Red Hat Fuse 7.1

Apache Camel Component Reference

Camel コンポーネントの設定リファレンス

Last Updated: 2023-10-09

Red Hat Fuse 7.1 Apache Camel Component Reference

Camel コンポーネントの設定リファレンス

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

Apache Camel には 100 を超えるコンポーネントがあり、各コンポーネントは高度な設定が可能です。本ガイドでは、各コンポーネントの設定について説明します。

目次

第1章 コンポーネントの概要

本章では、Apache Camel で利用可能なすべてのコンポーネントの概要を説明します。

1.1. コンテナーのタイプ

Red Hat Fuse は、Camel アプリケーションをデプロイするためのさまざまなコンテナータイプを提供します。

- Spring Boot
- Apache Karaf
- JBoss Enterprise Application Platform (JBoss EAP)

さらに、Camel アプリケーションは **コンテナーレス** として実行できます。つまり、Camel アプリケーションは特別なコンテナーを使用せずに JVM で直接実行されます。

場合によっては、Fuse が 1 つのコンテナーで Camel コンポーネントをサポートする可能性がありますが、それ以外はサポートされない場合があります。これにはさまざまな理由がありますが、あるコンポーネントがすべてのコンテナータイプに適しているとは限りません。たとえば、**camel-ejb** コンポーネントは Java EE(JBoss EAP)向けに厳密に設計されており、他のコンテナータイプではサポートできません。

1.2. サポートされるコンポーネント

以下のキーに注意してください。

Symbol	説明
✔	サポート対象
▮	サポートされないか、まだサポート対象ではない
非推奨	今後のリリースで削除される可能性があります。

表1.1「Apache Camel Component Support Matrix」 コンテナーでサポートされる Camel コンポーネントについての包括的な情報を提供します。

表1.1 Apache Camel Component Support Matrix

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
activemq-camel	エンドポイント	✔	✔	✔	✔
camel-ahc	エンドポイント	✔	✔	✔	✔
camel-ahc-ws	エンドポイント	✔	✔	✔	✔

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-ahc-wss	エンドポイント	✔	✔	✔	✔
camel-amqp	エンドポイント	✔	✔	✔	✔
camel-apns	エンドポイント	✔	✔	✔	✔
camel-asn1	データフォーマット	✔	✔	✔	✔
camel-asterisk	エンドポイント	✔	✔	✔	✔
camel-atmos	エンドポイント	✔	✔	▮	▮
camel-atmosphere-websocket	エンドポイント	✔	✔	✔	✔
camel-atom	エンドポイント	✔	✔	✔	✔
camel-atomix	エンドポイント	✔	✔	✔	✔
camel-avro	エンドポイント	✔	✔	✔	✔
camel-avro	データフォーマット	✔	✔	✔	✔
camel-aws	エンドポイント	✔	✔	✔	✔
camel-azure	エンドポイント	✔	✔	✔	✔
camel-bam	エンドポイント	**非推奨**	✔	✔	▮
camel-barcode	データフォーマット	✔	✔	✔	✔
camel-base64	データフォーマット	✔	✔	✔	✔
camel-bean	エンドポイント	✔	✔	✔	✔
camel-bean	言語	✔	✔	✔	✔
camel-bean-validator	エンドポイント	✔	✔	✔	✔
camel-beanio	データフォーマット	✔	✔	✔	✔
camel-beanstalk	エンドポイント	✔	✔	✔	▮

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-binding	エンドポイント	**非推奨**	✔	✔	✔
camel-bindy	エンドポイント	✔	✔	✔	✔
camel-bindy	データフォーマット	✔	✔	✔	✔
camel-blueprint	エンドポイント	✔	▮	✔	▮
camel-bonita	エンドポイント	✔	▮	▮	▮
camel-boon	データフォーマット	✔	✔	✔	✔
camel-box	エンドポイント	✔	✔	✔	✔
camel-braintree	エンドポイント	✔	✔	✔	✔
camel-browse	エンドポイント	✔	✔	✔	✔
camel-cache	エンドポイント	**非推奨**	✔	✔	▮
camel-caffeine	エンドポイント	✔	✔	✔	✔
camel-castor	データフォーマット	**非推奨**	✔	✔	✔
camel-cdi	エンドポイント	✔	▮	**非推奨**	✔
camel-chronicle-engine	エンドポイント	✔	✔	✔	✔
camel-chunk	エンドポイント	✔	✔	✔	✔
camel-class	エンドポイント	✔	✔	✔	✔
camel-cm-sms	エンドポイント	✔	✔	✔	✔
camel-cmis	エンドポイント	✔	✔	✔	▮
camel-coap	エンドポイント	✔	✔	✔	✔
camel-cometd	エンドポイント	✔	✔	✔	✔
camel-constant	言語	✔	✔	✔	✔
camel-context	エンドポイント	**非推奨**	▮	▮	▮

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-consul	エンドポイント	✔	✔	✔	✔
camel-controlbus	エンドポイント	✔	✔	✔	✔
camel-couchbase	エンドポイント	✔	✔	✔	✔
camel-couchdb	エンドポイント	✔	✔	✔	✔
camel-cql	エンドポイント	✔	✔	✔	✔
camel-crypto	エンドポイント	✔	✔	✔	✔
camel-crypto	データフォーマット	✔	✔	✔	✔
camel-crypto-cms	エンドポイント	✔	✔	✔	✔
camel-csv	データフォーマット	✔	✔	✔	✔
camel-cxf	エンドポイント	✔	✔	✔	✔
camel-cxf-transport	エンドポイント	✔	✔	✔	✔
camel-dataformat	エンドポイント	✔	✔	✔	✔
camel-dataset	エンドポイント	✔	✔	✔	✔
camel-digitalocean	エンドポイント	✔	✔	✔	✔
camel-direct	エンドポイント	✔	✔	✔	✔
camel-direct-vm	エンドポイント	✔	✔	✔	✔
camel-disruptor	エンドポイント	✔	✔	✔	✔
camel-dns	エンドポイント	✔	✔	✔	✔
camel-docker	エンドポイント	✔	✔	✔	✔
camel-dozer	エンドポイント	✔	✔	✔	✔
camel-drill	エンドポイント	✔	✔	✔	▮

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-dropbox	エンドポイント	✔	✔	✔	✔
camel-eclipse		**非推奨**	▮	▮	▮
camel-ehcache	エンドポイント	✔	✔	✔	✔
camel-ejb	エンドポイント	✔	▮	▮	✔
camel-el	言語	**非推奨**	▮	▮	▮
camel-elasticsearch	エンドポイント	✔	✔	✔	✔
camel-elasticsearch5	エンドポイント	✔	✔	✔	✔
camel-elasticsearch-rest	エンドポイント	✔	✔	✔	▮
camel-elsql	エンドポイント	✔	✔	✔	✔
camel-etcd	エンドポイント	✔	✔	✔	✔
camel-eventadmin	エンドポイント	✔	▮	✔	▮
camel-exchangeProperty	言語	✔	✔	✔	✔
camel-exec	エンドポイント	✔	✔	✔	✔
camel-facebook	エンドポイント	✔	✔	✔	✔
camel-fhir	データフォーマット	✔	✔	✔	▮
camel-file	エンドポイント	✔	✔	✔	✔
camel-file	言語	✔	✔	✔	✔
camel-flatpack	エンドポイント	✔	✔	✔	✔
camel-flatpack	データフォーマット	✔	✔	✔	✔

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-flink	エンドポイント	✔	✔	▮	✔
camel-fop	エンドポイント	✔	✔	✔	✔
camel-freemarker	エンドポイント	✔	✔	✔	✔
camel-ftp	エンドポイント	✔	✔	✔	✔
camel-gae	エンドポイント	**非推奨**	▮	▮	▮
camel-ganglia	エンドポイント	✔	✔	✔	▮
camel-geocoder	エンドポイント	✔	✔	✔	✔
camel-git	エンドポイント	✔	✔	✔	✔
camel-github	エンドポイント	✔	✔	✔	✔
camel-google-bigquery	エンドポイント	✔	✔	▮	✔
camel-google-calendar	エンドポイント	✔	✔	✔	✔
camel-google-drive	エンドポイント	✔	✔	✔	✔
camel-google-mail	エンドポイント	✔	✔	✔	✔
camel-google-pubsub	エンドポイント	✔	✔	✔	✔
camel-grape	エンドポイント	✔	✔	✔	▮
camel-groovy	言語	✔	✔	✔	✔
camel-groovy-dsl		**非推奨**	▮	▮	▮
camel-grpc	エンドポイント	✔	✔	✔	✔
camel-guava-eventbus	エンドポイント	✔	✔	✔	✔
camel-guice	エンドポイント	**非推奨**	✔	▮	▮

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-gzip	データフォーマット	✔	✔	✔	✔
camel-hawtdb	エンドポイント	**非推奨**	✔	✔	▮
camel-hazelcast	エンドポイント	✔	✔	✔	✔
camel-hbase	エンドポイント	✔	✔	▮	✔
camel-hdfs	エンドポイント	**非推奨**	✔	▮	▮
camel-hdfs2	エンドポイント	✔	✔	✔	✔
camel-header	言語	✔	✔	✔	✔
camel-headersmap		✔	✔	✔	✔
camel-hessian	データフォーマット	**非推奨**	✔	✔	✔
camel-hipchat	エンドポイント	✔	✔	✔	✔
camel-hl7	データフォーマット	✔	✔	✔	✔
camel-http	エンドポイント	**非推奨**	✔	✔	▮
camel-http4	エンドポイント	✔	✔	✔	✔
camel-hystrix	エンドポイント	✔	✔	✔	✔
camel-ibatis	エンドポイント	**非推奨**	▮	▮	▮
camel-ical	データフォーマット	✔	✔	✔	✔
camel-iec60870	エンドポイント	✔	✔	✔	✔
camel-ignite	エンドポイント	▮	▮	▮	▮
camel-imap	エンドポイント	✔	✔	✔	✔
camel-infinispan	エンドポイント	▮	▮	▮	▮
camel-influxdb	エンドポイント	✔	✔	✔	✔
camel-irc	エンドポイント	✔	✔	✔	✔

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-ironmq	エンドポイント	▮	▮	▮	▮
camel-jacksonxml	データフォーマット	✔	✔	✔	✔
camel-jasypt	エンドポイント	✔	✔	✔	✔
camel-javaspace	エンドポイント	**非推奨**	▮	▮	▮
camel-jaxb	データフォーマット	✔	✔	✔	✔
camel-jbpm	エンドポイント	▮	▮	▮	▮
camel-jcache	エンドポイント	✔	✔	✔	✔
camel-jcifs	エンドポイント	✔	▮	✔	▮
camel-jclouds	エンドポイント	✔	▮	✔	✔
camel-jcr	エンドポイント	✔	✔	✔	✔
camel-jdbc	エンドポイント	✔	✔	✔	✔
camel-jetty	エンドポイント	**非推奨**	**非推奨**	**非推奨**	▮
camel-jetty8	エンドポイント	▮	▮	▮	▮
camel-jetty9	エンドポイント	✔	✔	✔	▮
camel-jgroups	エンドポイント	✔	✔	✔	✔
camel-jibx	データフォーマット	✔	✔	✔	✔
camel-jing	エンドポイント	✔	✔	✔	✔
camel-jira	エンドポイント	✔	▮	▮	▮
camel-jms	エンドポイント	✔	✔	✔	✔
camel-jmx	エンドポイント	✔	✔	✔	✔
camel-jolt	エンドポイント	✔	✔	✔	✔
camel-josql	エンドポイント	**非推奨**	✔	✔	▮

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-jpa	エンドポイント	✔	✔	✔	✔
camel-jsch	エンドポイント	✔	✔	✔	✔
camel-json-fastjson	データフォーマット	✔	✔	✔	✔
camel-json-gson	データフォーマット	✔	✔	✔	✔
camel-json-jackson	データフォーマット	✔	✔	✔	✔
camel-json-johnzon	データフォーマット	✔	✔	✔	✔
camel-json-validator	エンドポイント	✔	✔	✔	▮
camel-json-xstream	データフォーマット	✔	✔	✔	✔
camel-jsonpath	言語	✔	✔	✔	✔
camel-jt400	エンドポイント	✔	✔	✔	✔
camel-juel	エンドポイント	**非推奨**	✔	✔	▮
camel-jxpath	言語	**非推奨**	▮	▮	▮
camel-kafka	エンドポイント	✔	✔	✔	✔
camel-kestrel	エンドポイント	**非推奨**	✔	✔	▮
camel-krati	エンドポイント	**非推奨**	✔	✔	▮
camel-kubernetes	エンドポイント	✔	✔	✔	✔
camel-kura		✔	▮	✔	▮
camel-ldap	エンドポイント	✔	✔	✔	✔
camel-ldif	エンドポイント	✔	✔	✔	▮
camel-leveldb	エンドポイント	✔	✔	✔	✔

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-linkedin	エンドポイント	✔	✔	✔	✔
camel-log	エンドポイント	✔	✔	✔	✔
camel-lpr	エンドポイント	✔	✔	✔	✔
camel-lra		▮	▮	▮	▮
camel-lucene	エンドポイント	✔	✔	✔	✔
camel-lumberjack	エンドポイント	✔	✔	✔	✔
camel-lzf	データフォーマット	✔	✔	✔	✔
camel-master		✔	✔	✔	▮
camel-mail	エンドポイント	✔	✔	✔	✔
camel-metrics	エンドポイント	✔	✔	✔	✔
camel-milo	エンドポイント	✔	✔	✔	✔
camel-mime-multipart	データフォーマット	✔	✔	✔	✔
camel-mina	エンドポイント	**非推奨**	▮	✔	▮
camel-mina2	エンドポイント	✔	✔	✔	✔
camel-mllp	エンドポイント	✔	✔	✔	✔
camel-mock	エンドポイント	✔	✔	✔	✔
camel-mongodb	エンドポイント	✔	✔	✔	✔
camel-mongodb-gridfs	エンドポイント	✔	✔	✔	✔
camel-mongodb3	エンドポイント	✔	✔	✔	✔
camel-mqtt	エンドポイント	✔	✔	✔	✔
camel-msv	エンドポイント	✔	✔	✔	✔

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-mustache	エンドポイント	✔	✔	✔	✔
camel-mvel	エンドポイント	✔	✔	✔	✔
camel-mvel	言語	✔	✔	✔	✔
camel-mybatis	エンドポイント	✔	✔	✔	✔
camel-nagios	エンドポイント	✔	✔	✔	▮
camel-nats	エンドポイント	✔	✔	✔	✔
camel-netty	エンドポイント	**非推奨**	✔	✔	▮
camel-netty-http	エンドポイント	**非推奨**	✔	✔	▮
camel-netty4	エンドポイント	✔	✔	✔	✔
camel-netty4-http	エンドポイント	✔	✔	✔	▮
camel-ognl	言語	✔	✔	✔	✔
camel-olingo2	エンドポイント	✔	✔	✔	✔
camel-olingo4	エンドポイント	✔	✔	✔	✔
camel-openshift	エンドポイント	**非推奨**	✔	✔	▮
camel-openstack	エンドポイント	✔	✔	✔	✔
camel-opentracing		✔	✔	✔	✔
camel-optaplanner	エンドポイント	✔	✔	✔	✔
camel-paho	エンドポイント	✔	✔	✔	✔
camel-paxlogging	エンドポイント	✔	▮	✔	▮
camel-pdf	エンドポイント	✔	✔	✔	✔
camel-pgevent	エンドポイント	✔	✔	✔	▮

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-pgp	データフォーマット	✔	✔	✔	✔
camel-php	言語	**非推奨**	▮	▮	✔
camel-pop3	エンドポイント	✔	✔	✔	✔
camel-printer	エンドポイント	✔	✔	✔	✔
camel-properties	エンドポイント	✔	✔	✔	✔
camel-protobuf	データフォーマット	✔	✔	✔	✔
camel-pubnub	エンドポイント	✔	✔	✔	✔
camel-python	言語	**非推奨**	▮	▮	▮
camel-quartz	エンドポイント	**非推奨**	✔	✔	▮
camel-quartz2	エンドポイント	✔	✔	✔	✔
camel-quickfix	エンドポイント	✔	✔	✔	✔
camel-rabbitmq	エンドポイント	✔	✔	✔	✔
camel-reactive-streams	エンドポイント	✔	✔	✔	✔
camel-reactor		✔	✔	✔	✔
camel-ref	エンドポイント	✔	✔	✔	✔
camel-ref	言語	✔	✔	✔	✔
camel-rest	エンドポイント	✔	✔	✔	✔
camel-rest-api	エンドポイント	✔	✔	✔	✔
camel-rest-swagger	エンドポイント	✔	✔	✔	✔
camel-restlet	エンドポイント	✔	✔	✔	▮
camel-ribbon		✔	✔	▮	✔

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-rmi	エンドポイント	✔	✔	✔	✔
camel-routebox	エンドポイント	**非推奨**	✔	✔	▮
camel-rss	エンドポイント	✔	✔	✔	✔
camel-rss	データフォーマット	✔	✔	✔	✔
camel-ruby	言語	**非推奨**	▮	▮	▮
camel-rx	エンドポイント	**非推奨**	✔	✔	▮
camel-saga	エンドポイント	▮	▮	▮	▮
camel-salesforce	エンドポイント	✔	✔	✔	✔
camel-sap	エンドポイント	✔	✔	✔	✔
camel-sap-netweaver	エンドポイント	✔	✔	✔	✔
camel-saxon	エンドポイント	✔	✔	✔	✔
camel-scala	エンドポイント	**非推奨**	✔	✔	▮
camel-scheduler	エンドポイント	✔	✔	✔	✔
camel-schematron	エンドポイント	✔	✔	✔	✔
camel-scp	エンドポイント	✔	✔	✔	✔
camel-scr	エンドポイント	**非推奨**	✔	**非推奨**	▮
camel-script	エンドポイント	✔	✔	✔	✔
camel-seda	エンドポイント	✔	✔	✔	✔
camel-serialization	データフォーマット	✔	✔	✔	✔
camel-servicenow	エンドポイント	✔	✔	✔	✔

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-servlet	エンドポイント	✔	✔	✔	✔
camel-servletlistener	エンドポイント	**非推奨**	✔	✔	▮
camel-sftp	エンドポイント	✔	✔	✔	✔
camel-shiro	エンドポイント	✔	✔	✔	✔
camel-simple	言語	✔	✔	✔	✔
camel-sip	エンドポイント	✔	✔	✔	✔
camel-sjms	エンドポイント	✔	✔	✔	✔
camel-sjms2	エンドポイント	✔	✔	✔	✔
camel-slack	エンドポイント	✔	✔	✔	✔
camel-smpp	エンドポイント	✔	✔	✔	✔
camel-snakeyaml	エンドポイント	✔	✔	✔	✔
camel-snmp	エンドポイント	✔	✔	✔	✔
camel-soapjaxb	データフォーマット	✔	✔	✔	✔
camel-solr	エンドポイント	✔	✔	✔	▮
camel-spark	エンドポイント	✔	✔	▮	▮
camel-spark-rest	エンドポイント	✔	✔	▮	▮
camel-spel	言語	✔	✔	▮	✔
camel-splunk	エンドポイント	✔	✔	✔	✔
camel-spring	エンドポイント	✔	✔	✔	✔
camel-spring-batch	エンドポイント	✔	✔	✔	✔
camel-spring-boot	エンドポイント	✔	✔	▮	▮

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-spring-cloud		✔	✔	▮	▮
camel-spring-cloud-netflix		✔	✔	▮	▮
camel-spring-event	エンドポイント	✔	✔	▮	✔
camel-spring-integration	エンドポイント	✔	▮	▮	✔
camel-spring-javaconfig	エンドポイント	✔	✔	▮	✔
camel-spring-ldap	エンドポイント	✔	✔	✔	✔
camel-spring-redis	エンドポイント	✔	✔	✔	✔
camel-spring-security	エンドポイント	✔	✔	✔	✔
camel-spring-ws	エンドポイント	✔	✔	✔	✔
camel-sql	エンドポイント	✔	✔	✔	✔
camel-sql-stored	エンドポイント	✔	✔	✔	✔
camel-ssh	エンドポイント	✔	✔	✔	✔
camel-stax	エンドポイント	✔	✔	✔	✔
camel-stomp	エンドポイント	✔	✔	✔	✔
camel-stream	エンドポイント	✔	✔	✔	✔
camel-string	データフォーマット	✔	✔	✔	✔
camel-string-template	エンドポイント	✔	✔	✔	✔
camel-stub	エンドポイント	✔	✔	✔	✔

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-swagger	エンドポイント	**非推奨**	✔	**非推奨**	▮
camel-swagger-java	エンドポイント	✔	✔	✔	✔
camel-syslog	データフォーマット	✔	✔	✔	✔
camel-tagsoup	エンドポイント	✔	✔	✔	✔
camel-tarfile	データフォーマット	✔	✔	✔	✔
camel-telegram	エンドポイント	✔	✔	✔	✔
camel-thrift	エンドポイント	✔	✔	✔	✔
camel-thrift	データフォーマット	✔	✔	✔	✔
camel-tika	エンドポイント	✔	✔	✔	▮
camel-timer	エンドポイント	✔	✔	✔	✔
camel-tokenize	言語	✔	✔	✔	✔
camel-twilio	エンドポイント	✔	✔	✔	✔
camel-twitter	エンドポイント	✔	✔	✔	✔
camel-undertow	エンドポイント	✔	✔	✔	✔
camel-univocity-csv	データフォーマット	✔	✔	✔	✔
camel-univocity-fixed	データフォーマット	✔	✔	✔	✔
camel-univocity-tsv	データフォーマット	✔	✔	✔	✔
camel-urlrewrite	エンドポイント	**非推奨**	✔	✔	▮
camel-validator	エンドポイント	✔	✔	✔	✔
camel-velocity	エンドポイント	✔	✔	✔	✔

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-vertx	エンドポイント	✔	✔	✔	✔
camel-vm	エンドポイント	✔	✔	✔	✔
camel-weather	エンドポイント	✔	✔	✔	✔
camel-websocket	エンドポイント	✔	✔	✔	▮
camel-wordpress	エンドポイント	✔	✔	✔	▮
camel-xchange	エンドポイント	✔	✔	✔	▮
camel-xmlbeans	データフォーマット	**非推奨**	**非推奨**	**非推奨**	✔
camel-xmljson	データフォーマット	✔	✔	✔	✔
camel-xmlrpc	エンドポイント	▮	▮	▮	▮
camel-xmlrpc	データフォーマット	▮	▮	▮	▮
camel-xmlsecurity	エンドポイント	✔	✔	✔	✔
camel-xmpp	エンドポイント	✔	✔	✔	✔
camel-xpath	言語	✔	✔	✔	✔
camel-xquery	エンドポイント	✔	✔	✔	✔
camel-xquery	言語	✔	✔	✔	✔
camel-xslt	エンドポイント	✔	✔	✔	✔
camel-xstream	データフォーマット	✔	✔	✔	✔
camel-xtokenize	言語	✔	✔	✔	✔
camel-yaml-snakeyaml	データフォーマット	✔	✔	✔	✔
camel-yammer	エンドポイント	✔	✔	✔	✔
camel-yql	エンドポイント	✔	✔	✔	▮

コンポーネント	Type	コンテナーレス	Spring Boot	Karaf	JBoss EAP
camel-zendesk	エンドポイント	✔	✔	▮	✔
camel-zip	データフォーマット	✔	✔	✔	✔
camel-zipfile	データフォーマット	✔	✔	✔	✔
camel-zipkin	エンドポイント	✔	✔	✔	✔
camel-zookeeper	エンドポイント	✔	✔	✔	✔
camel-zookeeper-master	エンドポイント	✔	✔	✔	✔

第2章 ACTIVEMQ

ACTIVEMQ コンポーネント

ActiveMQ コンポーネントを使用すると、メッセージを JMS Queue または Topic に送信できます。または Apache ActiveMQ を使用して JMS Queue またはトピックからメッセージを消費できます。

このコンポーネントは 169章*JMS コンポーネント* をベースとしており、送信に Spring の **JmsTemplate** を使用し、消費するために **MessageListenerContainer** を使用して宣言型トランザクションに Spring の JMS サポートを使用します。169章*JMS コンポーネント* コンポーネントのすべてのオプションは、このコンポーネントにも適用されます。

このコンポーネントを使用するには、クラスパスに **activemq.jar** または **activemq-core.jar** と **camel-core.jar**、**camel-spring.jar**、**camel-jms.jar** などの Apache Camel 依存関係が含まれるようにします。

トランザクションおよびキャッシュ

パフォーマンスに影響を与える可能 **性があるため、JMS でトランザクションを使用している場合は、JMS ページのトランザクションとキャッシュレベル** のセクション を参照してください。

URI 形式

```
activemq:[queue:|topic:]destinationName
```

destinationName は ActiveMQ のキューまたはトピック名です。デフォルトでは、**destinationName** はキュー名として解釈されます。たとえば、キュー **FOO.BAR** に接続するには、以下を使用します。

```
activemq:FOO.BAR
```

必要に応じて、オプションの **queue:** プレフィックスを含めることができます。

```
activemq:queue:FOO.BAR
```

トピックに接続するには、**topic:** プレフィックスを含める必要があります。たとえば、トピック **Stocks.Prices** に接続するには、以下を使用します。

```
activemq:topic:Stocks.Prices
```

オプション

これらのオプションはすべてこのコンポーネントに適用されるため、169章*JMS コンポーネント* コンポーネントのオプションを参照してください。

CAMEL ON EAP デプロイメント

このコンポーネントは、Red Hat JBoss Enterprise Application Platform (JBoss EAP) コンテナー上で簡素化されたデプロイメントモデルを提供する Camel on EAP (Wildfly Camel) フレームワークによってサポートされます。

ActiveMQ Camel コンポーネントを設定して、埋め込みブローカーまたは外部ブローカーのいずれかと

連携できます。JBoss EAP コンテナーにブローカーを組み込むには、「ActiveMQ Resource Adapter の設定」を参照してください。https://wildflyext.gitbooks.io/wildfly-camel/content/components/camel-activemq.html

接続ファクトリーの設定

以下の テストケース は、ActiveMQ への接続に使用される brokerURL を指定しながら **activeMQComponent（）** メソッド を使用して CamelContext に ActiveMQComponent を追加する方法を説明します。

```
camelContext.addComponent("activemq", activeMQComponent("vm://localhost?
broker.persistent=false"));
```

SPRING XML を使用した接続ファクトリーの設定

ActiveMQComponent で ActiveMQ ブローカー URL を以下のように設定できます。

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

  <camelContext xmlns="http://camel.apache.org/schema/spring">
  </camelContext>

  <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
    <property name="brokerURL" value="tcp://somehost:61616"/>
  </bean>

</beans>
```

接続プールの使用

Camel を使用して ActiveMQ ブローカーに送信する場合は、プールされた接続ファクトリーを使用して、JMS 接続、セッション、およびプロデューサーの効率的なプールを処理することが推奨されます。これは、ActiveMQ Spring Support ページに記載されています。

Maven を使用して AMQ プールを取得できます。

```
    <dependency>
      <groupId>org.apache.activemq</groupId>
      <artifactId>activemq-pool</artifactId>
      <version>5.3.2</version>
    </dependency>
```

そして、以下のように **activemq** コンポーネントを設定します。

```
    <bean id="jmsConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
        <property name="brokerURL" value="tcp://localhost:61616" />
```

```
    </bean>

    <bean id="pooledConnectionFactory"
class="org.apache.activemq.pool.PooledConnectionFactory" init-method="start" destroy-
method="stop">
        <property name="maxConnections" value="8" />
        <property name="connectionFactory" ref="jmsConnectionFactory" />
    </bean>

    <bean id="jmsConfig" class="org.apache.camel.component.jms.JmsConfiguration">
        <property name="connectionFactory" ref="pooledConnectionFactory"/>
        <property name="concurrentConsumers" value="10"/>
    </bean>

    <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
        <property name="configuration" ref="jmsConfig"/>
    </bean>
```

注記

プールされた接続ファクトリーの **init** メソッドおよび **destroy** メソッドに注意してください。これは、接続プールが適切に開始し、シャットダウンする上で重要です。

その後、**PooledConnectionFactory** は、同時に使用される最大 8 つの接続を持つ接続プールを作成します。各接続は、多くのセッションで共有できます。**maxActive** という名前のオプションを使用して、接続ごとのセッションの最大数を設定できます。デフォルト値は **500** です。**ActiveMQ 5.7** 以降では、オプションの名前が **maxActiveSessionPerConnection** と命名されるように名前が変更されたため、目的を反映しました。**concurrentConsumers** が **maxConnections** よりも高い値に設定されていることに注意してください。これは、各コンシューマーがセッションを使用しているため、セッションが同じ接続を共有できるため、安全です。この例では、8 * 500 = 4000 アクティブなセッションを同時に指定できます。

ルートでの MESSAGELISTENER POJO の呼び出し

ActiveMQ コンポーネントは、JMS MessageListener から プロセッサー へのヘルパー 型コンバーター も提供します。つまり、41章*Bean コンポーネント* コンポーネントはルート内で JMS MessageListener Bean を直接呼び出すことができます。

たとえば、以下のように JMS で MessageListener を作成できます。

```
public class MyListener implements MessageListener {
    public void onMessage(Message jmsMessage) {
        // ...
    }
}
```

次に、以下のようにルートで使用します。

```
from("file://foo/bar").
  bean(MyListener.class);
```

つまり、Apache Camel コンポーネントを再利用し、それらを JMS **MessageListener** POJO\ に簡単に統合できます。

ACTIVEMQ 宛先オプションの使用

ActiveMQ 5.6 から利用可能

"destination." プレフィックスを使用して、エンドポイント URI で Destination Options を設定できます。たとえば、コンシューマーを排他的としてマークし、その事前にフェッチサイズを 50 に設定するには、以下のように実行できます。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="file://src/test/data?noop=true"/>
    <to uri="activemq:queue:foo"/>
  </route>
  <route>
    <!-- use consumer.exclusive ActiveMQ destination option, notice we have to prefix with destination.
-->
    <from uri="activemq:foo?
destination.consumer.exclusive=true&amp;destination.consumer.prefetchSize=50"/>
    <to uri="mock:results"/>
  </route>
</camelContext>
```

アドバイザリーメッセージの消費

ActiveMQ は、消費可能なトピックに配置される アドバイザリーメッセージ を生成できます。このようなメッセージは、低速なコンシューマーを検出したり、統計をビルドしたりする場合にアラートを送信するのに役立ちます（1 日あたりのメッセージ/生成の数など）。 以下の Spring DSL の例は、トピックからメッセージを読み取る方法を示しています。

```
<route>
 <from uri="activemq:topic:ActiveMQ.Advisory.Connection?mapJmsMessage=false" />
 <convertBodyTo type="java.lang.String"/>
 <transform>
     <simple>${in.body}&#13;</simple>
 </transform>
 <to uri="file://data/activemq/?fileExist=Append&ileName=advisoryConnection-
${date:now:yyyyMMdd}.txt" />
</route>
```

キューでメッセージを使用する場合は、data/activemq フォルダーに以下のファイルが表示されるはずです。

advisoryConnection-20100312.txt advisoryProducer-20100312.txt

かつ、文字列が含まれる：

```
     ActiveMQMessage {commandId = 0, responseRequired = false, messageId = ID:dell-charles-
3258-1268399815140
     -1:0:0:0:221, originalDestination = null, originalTransactionId = null, producerId = ID:dell-charles-
     3258-1268399815140-1:0:0:0, destination = topic://ActiveMQ.Advisory.Connection, transactionId
= null,
     expiration = 0, timestamp = 0, arrival = 0, brokerInTime = 1268403383468, brokerOutTime =
1268403383468,
     correlationId = null, replyTo = null, persistent = false, type = Advisory, priority = 0, groupID = null,
```

```
    groupSequence = 0, targetConsumerId = null, compressed = false, userID = null, content = null,
    marshalledProperties = org.apache.activemq.util.ByteSequence@17e2705, dataStructure =
ConnectionInfo
    {commandId = 1, responseRequired = true, connectionId = ID:dell-charles-3258-1268399815140-
2:50,
    clientId = ID:dell-charles-3258-1268399815140-14:0, userName = , password = *****,
    brokerPath = null, brokerMasterConnector = false, manageable = true, clientMaster = true},
    redeliveryCounter = 0, size = 0, properties = {originBrokerName=master, originBrokerId=ID:dell-
charles-
    3258-1268399815140-0:0, originBrokerURL=vm://master}, readOnlyProperties = true,
readOnlyBody = true,
    droppable = false}
```

コンポーネント JAR の取得

この依存関係が必要です。

- **activemq-camel**

ActiveMQ は、ActiveMQ プロジェクト でリリースされた 169章*JMS コンポーネント* コンポーネントの拡張機能です。

```
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>activemq-camel</artifactId>
  <version>5.6.0</version>
</dependency>
```

第3章 AHC コンポーネント

Camel バージョン 2.8 から利用可能

ahc: component は、外部 HTTP リソース（HTTP を使用して外部サーバーを呼び出すためのクライアントとして）を使用するために HTTP ベースのエンドポイントを提供します。
コンポーネントは Async Http Client ライブラリーを使用します。

Maven ユーザーは、このコンポーネントの **pom.xml** に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ahc</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

3.1. URI 形式

```
ahc:http://hostname[:port][/resourceUri][?options]
ahc:https://hostname[:port][/resourceUri][?options]
```

デフォルトでは、HTTP にはポート 80 を使用し、HTTPS には 443 を使用します。

URI にクエリーオプションを追加するには、**?option=value&option=value&...**

3.2. AHCENDPOINT オプション

AHC エンドポイントは、URI 構文を使用して設定します。

```
ahc:httpUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

3.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
httpUri	http://hostname:port/pathなど、使用する URI が**必要です**。		URI

3.2.2. クエリーパラメーター（13 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
bridgeEndpoint (producer)	オプションが true の場合、Exchange.HTTP_URI ヘッダーは無視され、リクエストにエンドポイントの URI を使用します。また、throwExceptionOnFailure を false に設定して、AhcProducer がすべての障害応答を返せるようにすることもできます。	false	boolean
bufferSize (producer)	Camel と AHC クライアント間でデータを転送する際に使用される初期インメモリーバッファーサイズ。	4096	int
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを定義します。このパラメーターはデフォルトで false です。	false	boolean
cookieHandler (producer)	HTTP セッションを維持するためのクッキーハンドラーの設定		CookieHandler
headerFilterStrategy (producer)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
throwExceptionOnFailure (producer)	リモートサーバーからの応答に失敗した場合に AhcOperationFailedException のスローを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
transferException (producer)	有効で Exchange がコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合（Jetty や Servlet Camel コンポーネントの使用など）。プロデューサー側では、例外が AhcOperationFailedException ではなくデシリアライズされ、そのままスローされます。原因となる例外はシリアライズされている必要があります。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
バインディング（詳細）	AHC と Camel との間のバインド方法を制御するカスタム AhcBinding を使用します。		AhcBinding
clientConfig (advanced)	AsyncHttpClient がカスタム com.ning.http.client.AsyncHttpClientConfig インスタンスを使用するように設定する。		AsyncHttpClientConfig
clientConfigOptions (advanced)	マップのキー/値を使用して AsyncHttpClientConfig を設定します。		マップ

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
clientConfigRealmOptions (security)	マップのキー/値を使用して AsyncHttpClientConfig レルムを設定します。		マップ
sslContextParameters (security)	レジストリーの org.apache.camel.util.jsse.SSLContextParameters への参照。この参照は、コンポーネントレベルで設定された SSLContextParameters を上書きします。「JSSE 設定ユーティリティーの使用」を参照してください。このオプションを設定すると、エンドポイントまたはコンポーネントレベルで clientConfig オプションで指定される SSL/TLS 設定オプションが上書きされます。		SSLContextParameters

3.3. AHCCOMPONENT オプション

AHC コンポーネントは、以下に挙げる 8 つのオプションをサポートします。

Name	説明	デフォルト	Type
クライアント （詳細）	カスタム AsyncHttpClient を使用する場合		AsyncHttpClient
バインディング （詳細）	AHC と Camel との間のバインド方法を制御するカスタム AhcBinding を使用します。		AhcBinding
clientConfig (advanced)	AsyncHttpClient がカスタム com.ning.http.client.AsyncHttpClientConfig インスタンスを使用するように設定する。		AsyncHttpClientConfig
sslContextParameters (security)	レジストリーの org.apache.camel.util.jsse.SSLContextParameters への参照。このオプションを設定すると、エンドポイントまたはコンポーネントレベルで clientConfig オプションで指定される SSL/TLS 設定オプションが上書きされます。		SSLContextParameters

Name	説明	デフォルト	Type
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用する場合の java シリアライゼーションを許可するかどうか。これはデフォルトでオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AhcComponent にオプションのいずれかを設定すると、これらのオプションが作成されている **AhcEndpoints** に伝播されます。ただし、**AhcEndpoint** はカスタムオプションを設定/上書きすることもできます。エンドポイントに設定されたオプションは、常に **AhcComponent** のオプションよりも優先されます。

3.4. メッセージヘッダー

Name	タイプ	説明
Exchange.HTTP_URI	**文字列**	呼び出す URI。エンドポイントに直接設定された既存の URI を上書きします。
Exchange.HTTP_PATH	**文字列**	リクエスト URI のパス。ヘッダーは、リクエスト URI を HTTP_URI でビルドするために使用されます。パスが「/」で始まる場合は、http プロデューサーは Exchange.HTTP_BASE_URI ヘッダーまたは **exchange.getFromEndpoint（）.getEndpointUri（）;**に基づいて相対パスの検索を試みます。

Name	タイプ	説明
Exchange.HTTP_QUERY	**文字列**	**Camel 2.11 以降** では、URI パラメーター。エンドポイントに直接設定された既存の URI パラメーターを上書きします。
Exchange.HTTP_RESPONSE_CODE	**int**	外部サーバーからの HTTP 応答コード。OK の場合は 200 です。
Exchange.HTTP_CHARACTER_ENCODING	**文字列**	文字エンコーディング。
Exchange.CONTENT_TYPE	**文字列**	HTTP コンテンツタイプ。**テキスト/html** などのコンテンツタイプを提供するために IN メッセージと OUT メッセージの両方に設定されます。
Exchange.CONTENT_ENCODING	**文字列**	HTTP コンテンツのエンコーディング。**gzip** などのコンテンツエンコーディングを提供するために IN メッセージと OUT メッセージの両方に設定されます。

3.5. メッセージボディー

Camel は外部サーバーからの HTTP 応答を OUT ボディーに保存します。IN メッセージからのヘッダーはすべて OUT メッセージにコピーされるため、ヘッダーはルーティング時に保持されます。さらに Camel は HTTP 応答ヘッダーと OUT メッセージヘッダーを追加します。

3.6. レスポンスコード

Camel は HTTP レスポンスコードに従って処理されます。

- 応答コードは 100..299 の範囲にあり、Camel は成功レスポンスとして認識します。

- 応答コードは 300..399 の範囲にあり、Camel はこれをリダイレクト応答として認識し、情報と共に **AhcOperationFailedException** を発生させます。
- 応答コードは 400 以上で、Camel はこれを外部サーバーの失敗として認識し、情報と共に **AhcOperationFailedException** を発生させます。
  throwExceptionOnFailure

  オプション **throwExceptionOnFailure** を **false** に設定して、**AhcOperationFailedException** が失敗した応答コードに対してスローされないようにします。これにより、リモートサーバーから応答を取得できます。

3.7. AHCOPERATIONFAILEDEXCEPTION

この例外には、以下の情報が含まれます。

- HTTP ステータスコード
- HTTP ステータス行（ステータスコードのテキスト）
- リダイレクトを返した場合のリダイレクト場所のリダイレクト
- サーバーがボディーを応答として提供した場合、応答本体は **java.lang.String** として応答します。

3.8. GET または POST を使用した呼び出し

以下のアルゴリズムを使用して、**GET** または **POST** HTTP メソッドのいずれかを使用する必要があるかどうかを判断します。
1ヘッダーで提供されるメソッドを使用します。
2.クエリー文字列がヘッダーで提供される場合の **GET**。
3.エンドポイントがクエリー文字列で設定されている場合の **GET**。
4.送信するデータがある場合（ボディーは null ではありません）。
5.そうでない場合は GET。

3.9. 呼び出し用の URI の設定

HTTP プロデューサーの URI はエンドポイント URI を直接形成できます。以下のルートでは、Camel は HTTP を使用して外部サーバー（ 古いホスト ）に呼び出します。

```
from("direct:start")
        .to("ahc:http://oldhost");
```

同等の Spring サンプルは次のとおりです。

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
```

```
    <to uri="ahc:http://oldhost"/>
  </route>
</camelContext>
```

HTTP エンドポイント URI を上書きするには、キー Exchange.HTTP_URI でヘッダーをメッセージに追加します。

```
from("direct:start")
    .setHeader(Exchange.HTTP_URI, constant("http://newhost"))
    .to("ahc:http://oldhost");
```

3.10. URI パラメーターの設定

ahc プロデューサーは、HTTP サーバーに送信される URI パラメーターをサポートします。URI パラメーターはエンドポイント URI に直接設定することも、メッセージの Exchange.HTTP_QUERY キーのあるヘッダーとして設定できます。

```
from("direct:start")
        .to("ahc:http://oldhost?order=123&detail=short");
```

ヘッダーで提供されるオプション：

```
from("direct:start")
            .setHeader(Exchange.HTTP_QUERY, constant("order=123&detail=short"))
        .to("ahc:http://oldhost");
```

3.11. HTTP メソッドを HTTP プロデューサーに設定する方法

HTTP コンポーネントは、メッセージヘッダーを設定して HTTP リクエストメソッドを設定する方法を提供します。以下に例を示します。

```
from("direct:start")
            .setHeader(Exchange.HTTP_METHOD, constant("POST"))
        .to("ahc:http://www.google.com")
            .to("mock:results");
```

同等の Spring サンプルは次のとおりです。

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <setHeader headerName="CamelHttpMethod">
```

```
      <constant>POST</constant>
    </setHeader>
    <to uri="ahc:http://www.google.com"/>
    <to uri="mock:results"/>
  </route>
</camelContext>
```

3.12. 文字セットの設定

POST を使用してデータを送信する場合は、Exchange プロパティーを使用し て文字セット を設定できます。

```
exchange.setProperty(Exchange.CHARSET_NAME, "iso-8859-1");
```

3.12.1. エンドポイント URI からの URI パラメーター

この例では、Web ブラウザーに入力した内容のみの完全な URI エンドポイントがあります。Web ブラウザーの場合と同様に、複数の URI パラメーターをセパレーターとして使用し、区切り文字として使用できます。Camel はここでは複雑ではありません。

```
// we query for Camel at the Google page
template.sendBody("ahc:http://www.google.com/search?q=Camel", null);
```

3.12.2. メッセージの URI パラメーター

```
Map headers = new HashMap();
headers.put(Exchange.HTTP_QUERY, "q=Camel&lr=lang_en");
// we query for Camel and English language at Google
template.sendBody("ahc:http://www.google.com/search", null, headers);
```

上記のヘッダー値では、先頭に ? を付け ず、パラメーターは通常 &amp ; char で区切ることができることに注意してください。

3.12.3. 応答コードの取得

AHC コンポーネントから HTTP 応答コードを取得するには、Exchange.HTTP_RESPONSE_CODE の Out メッセージヘッダーから値を取得します。

```
Exchange exchange = template.send("ahc:http://www.google.com/search", new Processor() {
        public void process(Exchange exchange) throws Exception {
           exchange.getIn().setHeader(Exchange.HTTP_QUERY,
constant("hl=en&q=activemq"));
        }
```

```
});
Message out = exchange.getOut();
int responseCode = out.getHeader(Exchange.HTTP_RESPONSE_CODE, Integer.class);
```

3.13. ASYNCHTTPCLIENT の設定

AsyncHttpClient クライアントは AsyncHttpClientConfig を使用してクライアントを設定します。詳細は、
Async Http Client のドキュメントを参照してください。

Camel 2.8 では、設定は AsyncHttpClientConfig.Builder によって提供されるビルダーパターンの使用に制限されます。Camel 2.8 では、AsyncHttpClientConfig は getter/setter をサポートしません。そのため、Spring Bean スタイルを使用した作成/設定（XML ファイルの <bean> タグなど）はサポートされません。

以下の例は、ビルダーを使用して AhcComponent で設定した AsyncHttpClientConfig を作成する方法を示しています。

Camel 2.9 では、AHC コンポーネントは Async HTTP ライブラリー 1.6.4 を使用します。この新しいバージョンは、プレーンな Bean スタイルの設定のサポートを追加しています。AsyncHttpClientConfigBean クラスは、AsyncHttpClientConfig で利用可能な設定オプションの getter および setter を提供します。AsyncHttpClientConfigBean のインスタンスは AHC コンポーネントに直接渡すか、clientConfig URI パラメーターを使用してエンドポイント URI で参照することができます。

Camel 2.9 では、設定オプションを URI に直接設定する機能があります。「clientConfig.」で始まる URI パラメーターを使用して、AsyncHttpClientConfig の設定可能なさまざまなプロパティーを設定します。エンドポイント URI で指定されるプロパティーは、「clientConfig」 URI パラメーターによって参照される設定に指定されたもので、"clientConfig." パラメーターを使用して優先順位が付けられたプロパティーが設定されます。参照した AsyncHttpClientConfig インスタンスは、以前に作成されたエンドポイントの設定に関係なく、エンドポイントごとに常にコピーされます。以下の例は、「clientConfig.」タイプの URI パラメーターを使用して AHC コンポーネントを設定する方法を示しています。

```
from("direct:start")
    .to("ahc:http://localhost:8080/foo?
clientConfig.maxRequestRetry=3&clientConfig.followRedirects=true")
```

3.14. SSL サポート(HTTPS)

JSSE 設定ユーティリティーの使用

Camel 2.9 の時点で、AHC コンポーネントは Camel JSSE 設定ユーティリティーを介して SSL/TLS 設定をサポートします。 このユーティリティーは、エンドポイントおよびコンポーネントレベルで記述し、設定する必要のあるコンポーネント固有のコードの量を大幅に削減します。 以下の例は、AHC コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

AhcComponent component = context.getComponent("ahc", AhcComponent.class);
component.setSslContextParameters(scp));
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
    <camel:keyManagers
       keyPassword="keyPassword">
      <camel:keyStore
         resource="/users/home/server/keystore.jks"
         password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="ahc:https://localhost/foo?sslContextParameters=#sslContextParameters"/>
...
```

3.15. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- エンドポイント
- はじめに
- Jetty
- HTTP
- HTTP4

第4章 AHC WEBSOCKET コンポーネント

Camel バージョン 2.14 から利用可能

ahc-ws コンポーネントは、Websocket 経由で外部サーバーと通信するクライアント向けの Websocket ベースのエンドポイントを提供します（クライアントは外部サーバーへの Websocket 接続を開きます）。
コンポーネントは、 Async Http Client ライブラリーを使用する AHC コンポーネントを使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ahc-ws</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

4.1. URI 形式

```
ahc-ws://hostname[:port][/resourceUri][?options]
ahc-wss://hostname[:port][/resourceUri][?options]
```

ahc-wss にはポート 80 を使用し、ahc-wss には 443 を使用します。

4.2. AHC-WS オプション

AHC-WS コンポーネントは AHC コンポーネントをベースとしているため、AHC コンポーネントのさまざまな設定オプションを使用できます。

AHC Websocket コンポーネントは、以下に示す 8 個のオプションをサポートします。

Name	説明	デフォルト	Type
クライアント（詳細）	カスタム AsyncHttpClient を使用する場合		AsyncHttpClient

Name	説明	デフォルト	Type
バインディング（詳細）	AHC と Camel との間のバインド方法を制御するカスタム AhcBinding を使用します。		AhcBinding
clientConfig (advanced)	AsyncHttpClient がカスタム com.ning.http.client.AsyncHttpClientConfig インスタンスを使用するように設定する。		AsyncHttpClientConfig
sslContextParameters (security)	レジストリーの org.apache.camel.util.jsse.SSLContextParameters への参照。このオプションを設定すると、エンドポイントまたはコンポーネントレベルで clientConfig オプションで指定される SSL/TLS 設定オプションが上書きされます。		SSLContextParameters
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用する場合の java シリアライゼーションを許可するかどうか。これはデフォルトでオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AHC Websocket エンドポイントは、URI 構文を使用して設定します。

```
ahc-ws:httpUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

4.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
httpUri	http://hostname:port/pathなど、使用する URI が**必要です**。		URI

4.2.2. クエリーパラメーター（18 パラメーター）：

Name	説明	デフォルト	Type
bridgeEndpoint (common)	オプションが true の場合、Exchange.HTTP_URI ヘッダーは無視され、リクエストにエンドポイントの URI を使用します。また、throwExceptionOnFailure を false に設定して、AhcProducer がすべての障害応答を返せるようにすることもできます。	false	boolean
bufferSize (common)	Camel と AHC クライアント間でデータを転送する際に使用される初期インメモリーバッファーサイズ。	4096	int
headerFilterStrategy (common)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
throwExceptionOnFailure (common)	リモートサーバーからの応答に失敗した場合に AhcOperationFailedException のスローを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
transferException (common)	有効で Exchange がコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object コンテンツタイプとして応答でシリアライズされた場合（Jetty や Servlet Camel コンポーネントの使用など）。プロデューサー側では、例外が AhcOperationFailedException ではなくデシリアライズされ、そのままスローされます。原因となる例外はシリアライズされている必要があります。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendMessageOnError (consumer)	web-socket リスナーがエラーを受信した場合にメッセージを送信するかどうか。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを定義します。このパラメーターはデフォルトで false です。	false	boolean
cookieHandler (producer)	HTTP セッションを維持するためのクッキーハンドラーの設定		CookieHandler
useStreaming (producer)	ストリーミングがデータを複数のテキストフラグメントとして送信できるようにするには、以下を行います。	false	boolean
バインディング （詳細）	AHC と Camel との間のバインド方法を制御するカスタム AhcBinding を使用します。		AhcBinding
clientConfig (advanced)	AsyncHttpClient がカスタム com.ning.http.client.AsyncHttpClientConfig インスタンスを使用するように設定する。		AsyncHttpClientConfig
clientConfigOptions (advanced)	マップのキー/値を使用して AsyncHttpClientConfig を設定します。		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
clientConfigRealmOptions (security)	マップのキー/値を使用して AsyncHttpClientConfig レルムを設定します。		マップ
sslContextParameters (security)	レジストリーの org.apache.camel.util.jsse.SSLContextParameters への参照。この参照は、コンポーネントレベルで設定された SSLContextParameters を上書きします。「JSSE 設定ユーティリティーの使用」を参照してください。このオプションを設定すると、エンドポイントまたはコンポーネントレベルで clientConfig オプションで指定される SSL/TLS 設定オプションが上書きされます。		SSLContextParameters

4.3. WEBSOCKET でのデータの書き込みと読み取り

ahc-ws エンドポイントは、エンドポイントがプロデューサーまたはコンシューマーとして設定されるかによって、ソケットからデータを書き込みできます。

4.4. データの書き込みまたは読み取りのための URI の設定

以下のルートでは、Camel は指定された Websocket 接続に書き込みます。

```
from("direct:start")
        .to("ahc-ws://targethost");
```

同等の Spring サンプルは次のとおりです。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <to uri="ahc-ws://targethost"/>
  </route>
</camelContext>
```

以下のルートでは、Camel は指定された Websocket 接続から読み取られます。

```
from("ahc-ws://targethost")
        .to("direct:next");
```

同等の Spring サンプルは次のとおりです。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="ahc-ws://targethost"/>
    <to uri="direct:next"/>
  </route>
</camelContext>
```

4.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- AHC
- Atmosphere-Websocket

第5章 AMQP コンポーネント

Camel バージョン 1.2 で利用可能

amqp: コンポーネントは、Qpid プロジェクトの JMS Client API を使用して AMQP 1.0 プロトコルをサポートします。AMQP 0.9（特に RabbitMQ）を使用する場合は、Camel RabbitMQ コンポーネントにも関心がある可能性があります。Camel 2.17.0 AMQP コンポーネントは AMQP 0.9 以上をサポートしていましたが、Camel 2.17.0 以降は AMQP 1.0 のみをサポートすることに注意してください。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-amqp</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version --
>
</dependency>
```

5.1. URI 形式

```
amqp:[queue:|topic:]destinationName[?options]
```

5.2. AMQP オプション

宛先名の後に、JMS コンポーネントのさまざまな設定オプションをすべて指定できます。

AMQP コンポーネントは、以下に示す 80 個のオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有 JMS 設定を使用するには、以下を実行します。		JmsConfiguration

Name	説明	デフォルト	Type
acceptMessageswhile Stopping (consumer)	停止中にコンシューマーがメッセージを受け入れるかどうかを指定します。実行時に JMS ルートを開始および停止する場合に、キューでキューにまだキューに追加されたメッセージがある場合も、このオプションの有効化を検討してください。このオプションが false で、JMS ルートを停止すると、メッセージを拒否し、JMS ブローカーは再配信を試みる必要があります。これは再度拒否される可能性があり、最終的に JMS ブローカーのデッドレターキューに移動される可能性があります。これを回避するには、このオプションを有効にすることが推奨されます。	false	boolean
allowReplyManagerQuick Stop (consumer)	Request-reply メッセージングの応答マネージャーで使用される DefaultMessageListenerContainer により、JmsConfigurationisAcceptMessagesWhileStopping が有効になっている場合に DefaultMessageListenerContainer.runningAllowed フラグがすぐに停止でき、org.apache.camel.CamelContext が現在停止中です。この迅速な停止機能は、通常の JMS コンシューマーでデフォルトで有効になっていますが、応答マネージャーを有効にするには、このフラグを有効にする必要があります。	false	boolean
acknowledgementMode (consumer)	整数として定義される JMS 確認モード。ベンダー固有の拡張機能を確認モードに設定することができます。通常モードでは、代わりに acknowledgementModeName を使用することが推奨されます。		int
EagerLoadingOfプロパティー （コンシューマー）	JMS プロパティーは必要でない可能性があるため、通常は非効率となるメッセージの読み込み直後に JMS プロパティーの Eager 読み込みを有効にしますが、場合によっては、基礎となる JMS プロバイダーの問題と JMS プロパティーの使用に早期にキャッチできます。	false	boolean
acknowledgementModeName (consumer)	JMS 確認名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	文字列
autoStartup (consumer)	コンシューマーコンテナーが自動起動すべきかどうかを指定します。	true	boolean

Name	説明	デフォルト	Type
cacheLevel (consumer)	ベースにある JMS リソースの ID でキャッシュレベルを設定します。詳細は、cacheLevelName オプションを参照してください。		int
cacheLevelName (consumer)	ベースにある JMS リソースの名前でキャッシュレベルを設定します。使用できる値は、CACHE_AUTO、CACHE_CONNECTION、CACHE_CONSUMER、CACHE_NONE、CACHE_SESSION です。デフォルト設定は CACHE_AUTO です。詳細は、Spring ドキュメント および Transactions Cache Levels を参照してください。	CACHE_AUTO	文字列
replyToCacheLevelName (producer)	JMS で要求/応答を実行するときに、リプライコンシューマーの名前でキャッシュレベルを設定します。このオプションは、（一時的ではなく）固定応答キューを使用している場合に限り該当します。Camel はデフォルトで CACHE_CONSUMER（排他的の場合）または shared w/ replyToSelectorName を使用します。replyToSelectorName なしで共有される CACHE_SESSION。IBM WebSphere などの JMS ブローカーによっては、reToCacheLevelName=CACHE_NONE が機能するように設定する必要がある場合があります。注記： 一時キューを使用する場合は CACHE_NONE は許可されず、CACHE_CONSUMER や CACHE_SESSION などの高い値を使用する必要があります。		文字列
clientId (common)	使用する JMS クライアント ID を設定します。この値は一意でなければならず、単一の JMS 接続インスタンスでのみ使用できることに注意してください。通常、永続トピックサブスクリプションにのみ必要です。Apache ActiveMQ を使用する場合は、代わりに Virtual Topics を使用することが推奨されます。		文字列
concurrentConsumers (consumer)	（JMS 上でリクエストや応答ではなく）JMS から消費する場合の同時コンシューマーのデフォルト数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。JMS で要求/応答を行う場合、オプション replyToConcurrentConsumers を使用して、リプライメッセージリスナーで同時コンシューマーの数を制御します。	1	int
replyToConcurrent Consumers (producer)	JMS で要求/応答を実行する際の同時コンシューマーのデフォルト数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。	1	int

Name	説明	デフォルト	Type
connectionFactory (common)	使用する接続ファクトリー。接続ファクトリーはコンポーネントまたはエンドポイントのいずれかで設定する必要があります。		ConnectionFactory
ユーザー名 （セキュリティー）	ConnectionFactory で使用するユーザー名。また、ユーザー名/パスワードを ConnectionFactory に直接設定することもできます。		文字列
パスワード （セキュリティー）	ConnectionFactory で使用するパスワード。また、ユーザー名/パスワードを ConnectionFactory に直接設定することもできます。		文字列
deliveryPersistent (producer)	永続配信がデフォルトで使用されるかどうかを指定します。	true	boolean
deliveryMode (producer)	使用する配信モードを指定します。使用できる値は、javax.jms.DeliveryMode で定義される値です。NON_PERSISTENT = 1 and PERSISTENT = 2.		整数
durableSubscriptionName (common)	永続トピックサブスクリプションを指定するための永続サブスクライバー名。clientId オプションも設定する必要があります。		文字列
exceptionListener (advanced)	基礎となる JMS 例外が通知される JMS 例外リスナーを指定します。		ExceptionListener
errorHandler (advanced)	メッセージの処理中にキャッチされない例外が発生した場合に呼び出される org.springframework.util.ErrorHandler を指定します。デフォルトでは、errorHandler が設定されていない場合、これらの例外は WARN レベルでログに記録されます。errorHandlerLoggingLevel および errorHandlerLogStackTrace オプションを使用して、ロギングレベルおよびスタックトレースをログに記録するかどうかを設定できます。これにより、カスタム errorHandler をコーディングする必要よりも、設定が非常に簡単になります。		ErrorHandler
errorHandlerLoggingLevel (logging)	キャッチされない例外についてのデフォルトの errorHandler ロギングレベルの設定を可能にします。	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	デフォルトの errorHandler によってスタックトレースをログに記録するかどうかを制御できます。	true	boolean

Name	説明	デフォルト	Type
explicitQosEnabled (producer)	メッセージの送信時に、サービスの deliveryMode、priority、または timeToLive qualities が使用される場合を設定します。このオプションは Spring の JmsTemplate に基づいています。deliveryMode、priority、および timeToLive オプションは、現在のエンドポイントに適用されます。これは、メッセージの粒度で操作する preserveMessageQos オプションとは対照的で、Camel In メッセージヘッダーからのみ QoS プロパティーを読み取ります。	false	boolean
exposeListenerSession (consumer)	メッセージの消費時にリスナーセッションを公開するかどうかを指定します。	false	boolean
idleTaskExecutionLimit (advanced)	実行内でメッセージを受信せずに、受信タスクのアイドル実行の制限を指定します。この制限に達すると、タスクはシャットダウンし、他の実行中のタスク（動的なスケジューリングの場合は maxConcurrentConsumers の設定）を受信し続けます。Spring には追加のドキュメントがあります。	1	int
idleConsumerLimit (advanced)	いつでもアイドル状態にできるコンシューマーの数の制限を指定します。	1	int
maxConcurrentConsumers (consumer)	（JMS 上でリクエストや応答ではなく）JMS から消費する場合の同時コンシューマーの最大数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。JMS で要求/応答を行う場合、オプション replyToMaxConcurrentConsumers を使用して、リプライメッセージリスナーで同時コンシューマーの数を制御します。		int
replyToMaxConcurrent Consumers (producer)	JMS 上でリクエスト/応答を使用する場合の同時コンシューマーの最大数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。		int
replyOnTimeoutToMax ConcurrentConsumers (producer)	JMS でリクエスト/リプライタイムアウトを使用する場合に、継続中のルーティングの同時コンシューマーの最大数を指定します。	1	int
maxMessagesPerTask (advanced)	タスクごとのメッセージ数。-1 は無制限です。同時コンシューマー（min max など）に範囲を使用する場合、このオプションを使用して値を eg 100 に設定して、作業が少なくなる場合にコンシューマーのスピードを制御することができます。	-1	int

Name	説明	デフォルト	Type
messageConverter (advanced)	javax.jms.Message への/からのマップ方法を制御するため、カスタムの Spring org.springframework.jms.support.converter.MessageConverter を使用します。		MessageConverter
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ（javax.jms.TextMessage など）に自動マッピングするかどうかを指定します。	true	boolean
messageIdEnabled (advanced)	送信時に、メッセージ ID を追加するかどうかを指定します。これは、JMS ブローカーへのヒントです。JMS プロバイダーがこのヒントを受け入れる場合、これらのメッセージにはメッセージ ID が null に設定されている必要があります。プロバイダーがヒントを無視する場合は、メッセージ ID を通常の一意の値に設定する必要があります。	true	boolean
messageTimestampEnabled (advanced)	メッセージの送信時にタイムスタンプをデフォルトで有効にするかどうかを指定します。これは、JMS ブローカーへのヒントです。JMS プロバイダーがこのヒントを受け入れる場合、これらのメッセージにはタイムスタンプがゼロに設定される必要があります。プロバイダーがヒントを無視する場合は、タイムスタンプを通常の値に設定する必要があります。	true	boolean
alwaysCopyMessage (producer)	true の場合、送信のためにプロデューサーに渡されるときに Camel は常にメッセージの JMS メッセージコピーを作成します。replyToDestinationSelectorName が設定されている場合など、メッセージのコピーは一部の状況で必要になります（replyToDestinationSelectorName が設定されている場合、Camel は alwaysCopyMessage オプションを true に設定します）。	false	boolean
useMessageIDAsCorrelationID (advanced)	InOut メッセージに対して JMSCorrelationID として常に JMSMessageID を使用するかどうかを指定します。	false	boolean
優先順位 （プロデューサー）	1 より大きい値は送信時にメッセージの優先度を指定します（0 は優先度が最も低く、9 が最も高い値になります）。このオプションを有効にするには、explicitQosEnabled オプションも有効にする必要があります。	4	int
pubSubNoLocal (advanced)	独自の接続によって公開されるメッセージの配信を抑制するかどうかを指定します。	false	boolean

Name	説明	デフォルト	Type
receiveTimeout (advanced)	メッセージの受信のタイムアウト（ミリ秒単位）。	1000	Long
recoveryInterval (advanced)	リカバリーを試行する間隔を指定します（例： 接続の更新時（ミリ秒単位）。デフォルトは 5000 ミリ秒で、5 秒です。	5000	Long
taskExecutor (consumer)	メッセージ消費にカスタムタスクエグゼキューターを指定できます。		TaskExecutor
timeToLive (producer)	メッセージを送信する場合、メッセージの存続時間（ミリ秒単位）を指定します。	-1	Long
トランザクション（トランザクション）	トランザクションモードを使用するかどうかを指定します。	false	boolean
lazyCreateTransaction Manager (transaction)	true の場合、transacted=true オプション時に transactionManager が挿入されていない場合、Camel は JmsTransactionManager を作成します。	true	boolean
transactionManager (transaction)	使用する Spring トランザクションマネージャー。		PlatformTransaction Manager
transactionName (transaction)	使用するトランザクションの名前。		文字列
transactionTimeout (transaction)	トランザクションモードを使用している場合は、トランザクションのタイムアウト値（秒単位）。	-1	int
testConnectionOn Startup (common)	起動時に接続をテストするかどうかを指定します。これにより、Camel が起動すると、すべての JMS コンシューマーが JMS ブローカーへの有効な接続になります。コネクションを許可できない場合、Camel は起動時に例外をスローします。これにより、Camel が接続の失敗で開始しないようになります。JMS プロデューサーもテストされています。	false	boolean

Name	説明	デフォルト	Type
asyncStartListener (advanced)	ルートの開始時に JmsConsumer メッセージリスナーを非同期に起動するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの起動中に Camel がブロックされます。このオプションを true に設定すると、ルートの起動が可能になりますが、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用すると、接続が確立されないと、WARN レベルで例外がログに記録され、コンシューマーはメッセージを受信できなくなります。次にルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止するときに JmsConsumer メッセージリスナーを非同期的に停止するかどうか。	false	boolean
forceSendOriginal Message (producer)	mapJmsMessage=false を使用する場合、ルート中にヘッダー（get または set）を操作すると、Camel は新しい JMS 宛先に送信する新しい JMS メッセージを送信します。このオプションを true に設定して、Camel が受信した元の JMS メッセージを送信するように強制します。	false	boolean
requestTimeout (producer)	InOut エクスチェンジパターン（ミリ秒単位）を使用する場合の応答を待機するタイムアウト。デフォルトは 20 秒です。ヘッダー CamelJmsRequestTimeout を含めると、このエンドポイントに設定されたタイムアウト値を上書きすることができます。そのため、メッセージごとに個別のタイムアウト値を設定できます。requestTimeoutCheckerInterval オプションも参照してください。	20000	Long
requestTimeoutChecker Interval (advanced)	JMS でリクエスト/リプライを行うときに Camel がタイムアウトしたエクスチェンジをチェックする頻度を設定します。デフォルトでは、Camel は 1 秒ごとに 1 回チェックします。ただし、タイムアウトの発生時により迅速に対応する必要がある場合は、この間隔を減らしてより頻繁に確認できます。タイムアウトは、requestTimeout オプションで決定されます。	1000	Long

Name	説明	デフォルト	Type
transferExchange (advanced)	ボディーとヘッダーだけでなく、ネットワーク上でエクスチェンジを転送できます。次のフィールドが転送されます。本文、Out ボディー、フォールト本文、In ヘッダー、Out ヘッダー、フォールトヘッダー、エクスチェンジプロパティー、エクスチェンジ例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。プロデューサーとコンシューマー側の両方でこのオプションを有効にする必要があります。そのため、Camel はペイロードが通常のペイロードではなく Exchange を認識します。	false	boolean
transferException (advanced)	有効にすると、リクエスト応答メッセージング(InOut)を使用し、エクスチェンジがコンシューマー側で失敗した場合、発生した例外は javax.jms.ObjectMessage として応答として返信されます。クライアントが Camel の場合、返された Exception が再スローされます。これにより、Camel JMS をルーティングのブリッジとして使用できます。たとえば、永続キューを使用して堅牢なルーティングを有効にすることができます。transferExchange も有効化されている場合には、このオプションが優先されます。例外がシリアライズ可能である必要があります。コンシューマー側の元の例外は、プロデューサーに戻されたときに org.apache.camel.RuntimeCamelException などの外部例外でラップできます。	false	boolean
transferFault (advanced)	有効にすると、Request Reply messaging(InOut)を使用し、Exchange がコンシューマー側で SOAP fault（例外ではない）で失敗した場合、MessageisFault（）のフォールトフラグは、キー org.apache.camel.component.jms.JmsConstantsJMS_TRANSFER_FAULTJMS_TRANSFER_FAULTJMS_TRANSFER_FAULTJMS_TRANSFER_FAULT として応答として返信されます。クライアントが Camel の場合、返されるフォールトフラグはリンク org.apache.camel.MessagesetFault(boolean)に設定されます。これは、cxf や spring-ws などの SOAP ベースの障害をサポートする Camel コンポーネントを使用する際に有効にすることができます。	false	boolean
jmsOperations (advanced)	org.springframework.jms.core.JmsOperations インターフェースの実装を使用できます。Camel はデフォルトで JmsTemplate を使用します。テストには使用できますが、Spring API ドキュメントに記載されているものだけを使用することはできません。		JmsOperations

Name	説明	デフォルト	Type
destinationResolver (advanced)	独自のリゾルバーを使用できるプラグ可能な org.springframework.jms.support.destination.DestinationResolver（たとえば、JNDI レジストリーから実際の宛先を検索するためなど）。		DestinationResolver
replyToType (producer)	JMS で要求/応答を実行するときに replyTo キューに使用するストラテジーを明示的に指定できます。使用できる値は、Temporary、Shared、または Exclusive です。デフォルトでは、Camel は一時キューを使用します。ただし、responseTo が設定されている場合は、デフォルトで Shared が使用されます。このオプションを使用すると、共有キューの代わりに排他キューを使用できます。詳細は、Camel JMS ドキュメント を参照してください。特に、クラスター環境で実行している場合の影響に関する注意事項と、Shared 応答キューのパフォーマンスは、その代替時間または排他的なものよりも低下します。		ReplyToType
preserveMessageQos (producer)	JMS エンドポイントの QoS 設定の代わりに、メッセージに指定された QoS 設定を使用してメッセージを送信する場合は、true に設定します。以下の 3 つのヘッダーは JMSPriority、JMSDeliveryMode、および JMSExpiration とみなされます。提供できるのは、すべてまたは一部のみです。指定しないと、Camel は代わりにエンドポイントから値を使用するようにフォールバックします。そのため、このオプションを使用すると、ヘッダーはエンドポイントの値を上書きします。一方、explicitQosEnabled オプションはエンドポイントに設定されたオプションのみを使用し、メッセージヘッダーの値を使用しません。	false	boolean
asyncConsumer (consumer)	JmsConsumer が Exchange を非同期的に処理するかどうか。有効にすると、JmsConsumer は JMS キューから次のメッセージを取得し、以前のメッセージは非同期で処理されます（非同期ルーティングエンジンにより）。つまり、メッセージは 100% を厳密に順番に処理できます。無効にすると（デフォルト）、JmsConsumer が JMS キューから次のメッセージを取得する前に Exchange が完全に処理されます。トランザクション処理が有効になっている場合、非同期Consumer=true は非同期的に実行されません。トランザクションは同期的に実行される必要があるため（Camel 3.0 は非同期トランザクションをサポートする可能性があります）。	false	boolean
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false で、メッセージボディーが null の場合、JMSException が発生します。	true	boolean

Name	説明	デフォルト	Type
includeSentJMSMessageID (producer)	InOnly を使用して JMS 宛先に送信する場合にのみ適用されます（例： fire and forget）。このオプションを有効にすると、Camel Exchange は、メッセージが JMS 宛先に送信されたときに JMS クライアントによって使用される実際の JMSMessageID でエンリッチされます。	false	boolean
includeAllJMSXプロパティー (advanced)	JMS から Camel メッセージへのマッピング時に、すべての JMSXxxx プロパティーを含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合は、このオプションは適用されません。	false	boolean
defaultTaskExecutor Type (consumer)	コンシューマーエンドポイントとプロデューサーエンドポイントの ReplyTo コンシューマーの両方に使用するデフォルトの TaskExecutor タイプを DefaultMessageListenerContainer に指定します。使用できる値 - SimpleAsync（Spring の SimpleAsyncTaskExecutor を使用）または ThreadPool（最適な値で Spring の ThreadPoolTaskExecutor を使用）、キャッシュされたスレッドプールのような値。設定しないと、コンシューマーエンドポイントにキャッシュされたスレッドプールを使用し、応答コンシューマーに SimpleAsync を使用する以前の動作がデフォルトになります。ThreadPool を使用すると、同時コンシューマーを動的に増減するエラスティック設定でスレッドごみ箱を減らすことが推奨されます。		DefaultTaskExecutor Type
jmsKeyFormatStrategy (advanced)	JMS 鍵をエンコードおよびデコードするためのプラグ可能なストラテジー。これにより、JMS 仕様に準拠します。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーは、ドットとハイフン（. および -）を安全にマーシャリングします。passthrough ストラテジーは、鍵をそのまま残します。JMS ヘッダーキーに不正な文字が含まれるかどうかは気にしない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy

Name	説明	デフォルト	Type
allowAdditionalHeaders (producer)	このオプションは、JMS 仕様に従って無効な値がある可能性のある追加のヘッダーを許可するために使用されます。たとえば、WMQ などの一部のメッセージシステムは、バイト配列またはその他の無効な型を持つ値が含まれる接頭辞 JMS_IBM_MQMD_ を使用するヘッダー名を持つものです。複数のヘッダー名をコンマで区切って指定し、ワイルドカードのマッチングに使用する接尾辞として使用できます。		文字列
queueBrowseStrategy (advanced)	キューの参照時にカスタム QueueBrowseStrategy を使用する		QueueBrowseStrategy
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信するときに Camel が javax.jms.Message オブジェクトの新規インスタンスを作成する際に呼び出される指定の MessageCreatedStrategy を使用します。		MessageCreatedStrategy
waitForProvisionCorrelationToBeUpdated Counter (advanced)	request/reply over JMS を実行し、オプション useMessageIDAsCorrelationID が有効になっている場合に、プロビジョニング ID が実際の相関 ID に更新されるまで待機する回数。	50	int
waitForProvisionCorrelationToBeUpdated ThreadSleepingTime (advanced)	プロビジョニングの相関 ID の更新を待機する間に毎回スリープ状態になる間隔（ミリ秒単位）。	100	Long
correlationProperty (producer)	この JMS プロパティーを使用して、JMSCorrelationID プロパティーの代わりに InOut 交換パターン（リクエスト-reply）でメッセージを関連付けます。これにより、JMSCorrelationID JMS プロパティーを使用して、メッセージを関連付けないシステムとメッセージを交換できます。JMSCorrelationID を使用した場合、Camel によって使用または設定されません。同じ名前のメッセージのヘッダーに指定されていない場合、ここで名前付きプロパティーの値が生成されます。		文字列
subscriptionDurable (consumer)	サブスクリプションを永続化するかどうかを設定します。使用する永続サブスクリプション名は、subscriptionName プロパティーから指定できます。デフォルトは false です。通常、永続サブスクリプションを登録するには、これを true に設定します（メッセージリスナークラス名がサブスクリプション名として適切である場合）。トピックをリッスンする場合にのみ（pub-sub ドメイン）、このメソッドは pubSubDomain フラグも切り替えます。	false	boolean

Name	説明	デフォルト	Type
subscriptionShared (consumer)	サブスクリプションを共有するかどうかを設定します。使用する共有サブスクリプション名は、subscriptionName プロパティーから指定できます。デフォルトは false です。共有サブスクリプションを登録するには、true に設定します。通常、subscriptionName の値と併用します（メッセージリスナークラス名がサブスクリプション名として適切である場合）。共有サブスクリプションも永続的である可能性があるため、このフラグは subscriptionDurable と組み合わせることもできます。トピックをリッスンする場合にのみ（pub-sub ドメイン）、このメソッドは pubSubDomain フラグも切り替えます。JMS 2.0 と互換性のあるメッセージブローカーが必要です。	false	boolean
subscriptionName (consumer)	作成するサブスクリプションの名前を設定します。共有サブスクリプションまたは永続サブスクリプションがあるトピック（pub-sub ドメイン）の場合に適用できます。サブスクリプション名は、クライアントの JMS クライアント ID 内で一意である必要があります。default は、指定されたメッセージリスナーのクラス名です。注記： 共有サブスクリプション（JMS 2.0 が必要）を除き、サブスクリプションごとに 1 つの同時コンシューマー（このメッセージリスナーコンテナーのデフォルト）のみが許可されます。		文字列
streamMessageType Enabled (producer)	StreamMessage タイプが有効かどうかを指定します。ファイル、InputStream などのストリーミングタイプのメッセージペイロードは、BytesMessage または StreamMessage として送信されます。このオプションは、使用される種類を制御します。デフォルトでは、BytesMessage を使用して、メッセージペイロード全体をメモリーに読み込みます。このオプションを有効にすると、メッセージペイロードはチャンクのメモリーに読み取られ、各チャンクはデータがなくなるまで StreamMessage に書き込まれます。	false	boolean
formatDateHeadersTo Iso8601 (producer)	ISO 8601 標準に従って日付ヘッダーをフォーマットするかどうかを設定します。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AMQP エンドポイントは、URI 構文を使用して設定されます。

```
amqp:destinationType:destinationName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

5.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
destinationType	使用する宛先の種類	queue	文字列
destinationName	宛先として使用するキューまたはトピックの名前。		文字列

5.2.2. クエリーパラメーター（91 パラメーター）：

Name	説明	デフォルト	Type
clientId (common)	使用する JMS クライアント ID を設定します。この値は一意でなければならず、単一の JMS 接続インスタンスでのみ使用できることに注意してください。通常、永続トピックサブスクリプションにのみ必要です。Apache ActiveMQ を使用する場合は、代わりに Virtual Topics を使用することが推奨されます。		文字列
connectionFactory (common)	使用する接続ファクトリー。接続ファクトリーはコンポーネントまたはエンドポイントのいずれかで設定する必要があります。		ConnectionFactory

Name	説明	デフォルト	Type
disableReplyTo (common)	Camel がメッセージの JMSReplyTo ヘッダーを無視するかどうかを指定します。true の場合、Camel は JMSReplyTo ヘッダーで指定された宛先に返信を返しません。Camel がルートから消費し、コードの別のコンポーネントがリプライメッセージを処理するため、Camel が自動的に応答メッセージを送りないようにするには、このオプションを使用できます。Camel を異なるメッセージブローカー間でプロキシーとして使用し、あるシステムから別のシステムへメッセージをルーティングする場合は、このオプションを使用することもできます。	false	boolean
durableSubscriptionName (common)	永続トピックサブスクリプションを指定するための永続サブスクライバー名。clientId オプションも設定する必要があります。		文字列
jmsMessageType (common)	JMS メッセージの送信に、特定の javax.jms.Message 実装を強制的に使用できるようにします。使用できる値は、Bytes、Map、Object、Stream、Text です。デフォルトでは、Camel は In body タイプから使用する JMS メッセージタイプを決定します。このオプションを指定すると、指定できます。		JmsMessageType
testConnectionOnStartup (common)	起動時に接続をテストするかどうかを指定します。これにより、Camel が起動すると、すべての JMS コンシューマーが JMS ブローカーへの有効な接続になります。コネクションを許可できない場合、Camel は起動時に例外をスローします。これにより、Camel が接続の失敗で開始しないようになります。JMS プロデューサーもテストされています。	false	boolean
acknowledgementModeName (consumer)	JMS 確認名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	文字列
asyncConsumer (consumer)	JmsConsumer が Exchange を非同期的に処理するかどうか。有効にすると、JmsConsumer は JMS キューから次のメッセージを取得し、以前のメッセージは非同期で処理されます（非同期ルーティングエンジンにより）。つまり、メッセージは 100% を厳密に順番に処理できます。無効にすると（デフォルト）、JmsConsumer が JMS キューから次のメッセージを取得する前に Exchange が完全に処理されます。トランザクション処理が有効になっている場合、非同期Consumer=true は非同期的に実行されません。トランザクションは同期的に実行される必要があるため（Camel 3.0 は非同期トランザクションをサポートする可能性があります）。	false	boolean

Name	説明	デフォルト	Type
autoStartup (consumer)	コンシューマーコンテナーが自動起動すべきかどうかを指定します。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
cacheLevel (consumer)	ベースにある JMS リソースの ID でキャッシュレベルを設定します。詳細は、cacheLevelName オプションを参照してください。		int
cacheLevelName (consumer)	ベースにある JMS リソースの名前でキャッシュレベルを設定します。使用できる値は、CACHE_AUTO、CACHE_CONNECTION、CACHE_CONSUMER、CACHE_NONE、CACHE_SESSION です。デフォルト設定は CACHE_AUTO です。詳細は、Spring ドキュメント および Transactions Cache Levels を参照してください。	CACHE_AUTO	文字列
concurrentConsumers (consumer)	（JMS 上でリクエストや応答ではなく）JMS から消費する場合の同時コンシューマーのデフォルト数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。JMS で要求/応答を行う場合、オプション replyToConcurrentConsumers を使用して、リプライメッセージリスナーで同時コンシューマーの数を制御します。	1	int
maxConcurrentConsumers (consumer)	（JMS 上でリクエストや応答ではなく）JMS から消費する場合の同時コンシューマーの最大数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。JMS で要求/応答を行う場合、オプション replyToMaxConcurrentConsumers を使用して、リプライメッセージリスナーで同時コンシューマーの数を制御します。		int
replyTo (consumer)	Message.getJMSReplyTo（）の着信値を上書きする明示的な ReplyTo 宛先を提供します。		文字列

Name	説明	デフォルト	Type
replyToDeliveryPersistent (consumer)	返信に永続配信を使用するかどうかを指定します。	true	boolean
セレクター（コンシューマー）	使用する JMS セレクターを設定します。		文字列
subscriptionDurable (consumer)	サブスクリプションを永続化するかどうかを設定します。使用する永続サブスクリプション名は、subscriptionName プロパティーから指定できます。デフォルトは false です。通常、永続サブスクリプションを登録するには、これを true に設定します（メッセージリスナークラス名がサブスクリプション名として適切である場合）。トピックをリッスンする場合にのみ（pub-sub ドメイン）、このメソッドは pubSubDomain フラグも切り替えます。	false	boolean
subscriptionName (consumer)	作成するサブスクリプションの名前を設定します。共有サブスクリプションまたは永続サブスクリプションがあるトピック（pub-sub ドメイン）の場合に適用できます。サブスクリプション名は、クライアントの JMS クライアント ID 内で一意である必要があります。default は、指定されたメッセージリスナーのクラス名です。注記： 共有サブスクリプション（JMS 2.0 が必要）を除き、サブスクリプションごとに 1 つの同時コンシューマー（このメッセージリスナーコンテナーのデフォルト）のみが許可されます。		文字列
subscriptionShared (consumer)	サブスクリプションを共有するかどうかを設定します。使用する共有サブスクリプション名は、subscriptionName プロパティーから指定できます。デフォルトは false です。共有サブスクリプションを登録するには、true に設定します。通常、subscriptionName の値と併用します（メッセージリスナークラス名がサブスクリプション名として適切である場合）。共有サブスクリプションも永続的である可能性があるため、このフラグは subscriptionDurable と組み合わせることもできます。トピックをリッスンする場合にのみ（pub-sub ドメイン）、このメソッドは pubSubDomain フラグも切り替えます。JMS 2.0 と互換性のあるメッセージブローカーが必要です。	false	boolean

Name	説明	デフォルト	Type
acceptMessagesWhileStopping (consumer)	停止中にコンシューマーがメッセージを受け入れるかどうかを指定します。実行時に JMS ルートを開始および停止する場合に、キューでキューにまだキューに追加されたメッセージがある場合も、このオプションの有効化を検討してください。このオプションが false で、JMS ルートを停止すると、メッセージを拒否し、JMS ブローカーは再配信を試みる必要があります。これは再度拒否される可能性があり、最終的に JMS ブローカーのデッドレターキューに移動される可能性があります。これを回避するには、このオプションを有効にすることが推奨されます。	false	boolean
allowReplyManagerQuickStop (consumer)	Request-reply メッセージングの応答マネージャーで使用される DefaultMessageListenerContainer により、JmsConfigurationisAcceptMessagesWhileStopping が有効になっている場合に DefaultMessageListenerContainer.runningAllowed フラグがすぐに停止でき、org.apache.camel.CamelContext が現在停止中です。この迅速な停止機能は、通常の JMS コンシューマーでデフォルトで有効になっていますが、応答マネージャーを有効にするには、このフラグを有効にする必要があります。	false	boolean
consumerType (consumer)	使用するコンシューマータイプ。Simple、Default、または Custom のいずれかです。コンシューマータイプは、使用する Spring JMS リスナーを決定します。デフォルトは org.springframework.jms.listener.DefaultMessageListenerContainer を使用します。Simple は org.springframework.jms.listener.SimpleMessageListenerContainer を使用します。Custom を指定すると、messageListenerContainerFactory オプションによって定義された MessageListenerContainerFactory は、使用する org.springframework.jms.listener.AbstractMessageListenerContainer を決定します。	デフォルト	ConsumerType

Name	説明	デフォルト	Type
defaultTaskExecutorType (consumer)	コンシューマーエンドポイントとプロデューサーエンドポイントの ReplyTo コンシューマーの両方に使用するデフォルトの TaskExecutor タイプを DefaultMessageListenerContainer に指定します。使用できる値 - SimpleAsync（Spring の SimpleAsyncTaskExecutor を使用）または ThreadPool（最適な値で Spring の ThreadPoolTaskExecutor を使用）、キャッシュされたスレッドプールのような値。設定しないと、コンシューマーエンドポイントにキャッシュされたスレッドプールを使用し、応答コンシューマーに SimpleAsync を使用する以前の動作がデフォルトになります。ThreadPool を使用すると、同時コンシューマーを動的に増減するエラスティック設定でスレッドごみ箱を減らすことが推奨されます。		DefaultTaskExecutor Type
eagerLoadingOfProperties (consumer)	JMS プロパティーは必須ではありませんが、基礎となる JMS プロバイダーと JMS プロパティーの使用の初期段階で、メッセージが読み込まれるとすぐに JMS プロパティーとペイロードの Eager 読み込みを有効にします。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
exposeListenerSession (consumer)	メッセージの消費時にリスナーセッションを公開するかどうかを指定します。	false	boolean
replyToSameDestination Allowed（コンシューマー）	JMS コンシューマーがコンシューマーが消費に使用する同じ宛先にリプライメッセージを送信することができるかどうか。これにより、同じメッセージを独自に消費してバックエンドレスループを防ぐことができます。	false	boolean
taskExecutor (consumer)	メッセージ消費にカスタムタスクエグゼキューターを指定できます。		TaskExecutor
deliveryMode (producer)	使用する配信モードを指定します。使用できる値は、javax.jms.DeliveryMode で定義される値です。NON_PERSISTENT = 1 and PERSISTENT = 2.		整数

Name	説明	デフォルト	Type
deliveryPersistent (producer)	永続配信がデフォルトで使用されるかどうかを指定します。	true	boolean
explicitQosEnabled (producer)	メッセージの送信時に、サービスの deliveryMode、priority、または timeToLive qualities が使用される場合を設定します。このオプションは Spring の JmsTemplate に基づいています。deliveryMode、priority、および timeToLive オプションは、現在のエンドポイントに適用されます。これは、メッセージの粒度で操作する preserveMessageQos オプションとは対照的で、Camel In メッセージヘッダーからのみ QoS プロパティーを読み取ります。	false	ブール値
formatDateHeadersToIso8601 (producer)	ISO 8601 標準に従って JMS 日付プロパティーをフォーマットするかどうかを設定します。	false	boolean
preserveMessageQos (producer)	JMS エンドポイントの QoS 設定の代わりに、メッセージに指定された QoS 設定を使用してメッセージを送信する場合は、true に設定します。以下の 3 つのヘッダーは JMSPriority、JMSDeliveryMode、および JMSExpiration とみなされます。提供できるのは、すべてまたは一部のみです。指定しないと、Camel は代わりにエンドポイントから値を使用するようにフォールバックします。そのため、このオプションを使用すると、ヘッダーはエンドポイントの値を上書きします。一方、explicitQosEnabled オプションはエンドポイントに設定されたオプションのみを使用し、メッセージヘッダーの値を使用しません。	false	boolean
優先順位 （プロデューサー）	1 より大きい値は送信時にメッセージの優先度を指定します（0 は優先度が最も低く、9 が最も高い値になります）。このオプションを有効にするには、explicitQosEnabled オプションも有効にする必要があります。	4	int
replyToConcurrentConsumers (producer)	JMS で要求/応答を実行する際の同時コンシューマーのデフォルト数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。	1	int
replyToMaxConcurrent Consumers (producer)	JMS 上でリクエスト/応答を使用する場合の同時コンシューマーの最大数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。		int

Name	説明	デフォルト	Type
replyToOnTimeoutMaxConcurrentConsumers (producer)	JMS でリクエスト/リプライタイムアウトを使用する場合に、継続中のルーティングの同時コンシューマーの最大数を指定します。	1	int
replyToOverride (producer)	JMS メッセージで明示的な ReplyTo 宛先を提供します。これにより、ReplyTo の設定が上書きされます。メッセージをリモートキューへ転送し、ReplyTo 宛先からリプライメッセージを受信する場合に便利です。		文字列
replyToType (producer)	JMS で要求/応答を実行するときに replyTo キューに使用するストラテジーを明示的に指定できます。使用できる値は、Temporary、Shared、または Exclusive です。デフォルトでは、Camel は一時キューを使用します。ただし、responseTo が設定されている場合は、デフォルトで Shared が使用されます。このオプションを使用すると、共有キューの代わりに排他キューを使用できます。詳細は、Camel JMS ドキュメント を参照してください。特に、クラスター環境で実行している場合の影響に関する注意事項と、Shared 応答キューのパフォーマンスは、その代替時間または排他的なものよりも低下します。		ReplyToType
requestTimeout (producer)	InOut エクスチェンジパターン（ミリ秒単位）を使用する場合の応答を待機するタイムアウト。デフォルトは 20 秒です。ヘッダー CamelJmsRequestTimeout を含めると、このエンドポイントに設定されたタイムアウト値を上書きすることができます。そのため、メッセージごとに個別のタイムアウト値を設定できます。requestTimeoutCheckerInterval オプションも参照してください。	20000	Long
timeToLive (producer)	メッセージを送信する場合、メッセージの存続時間（ミリ秒単位）を指定します。	-1	Long
allowAdditionalHeaders (producer)	このオプションは、JMS 仕様に従って無効な値がある可能性のある追加のヘッダーを許可するために使用されます。たとえば、WMQ などの一部のメッセージシステムは、バイト配列またはその他の無効な型を持つ値が含まれる接頭辞 JMS_IBM_MQMD_ を使用するヘッダー名を持つものです。複数のヘッダー名をコンマで区切って指定し、ワイルドカードのマッチングに使用する接尾辞として使用できます。		文字列
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false で、メッセージボディーが null の場合、JMSException が発生します。	true	boolean

Name	説明	デフォルト	Type
alwaysCopyMessage (producer)	true の場合、送信のためにプロデューサーに渡されるときに Camel は常にメッセージの JMS メッセージコピーを作成します。replyToDestinationSelectorName が設定されている場合など、メッセージのコピーは一部の状況で必要になります（replyToDestinationSelectorName が設定されている場合、Camel は alwaysCopyMessage オプションを true に設定します）。	false	boolean
correlationProperty (producer)	InOut 交換パターンを使用する場合は、JMSCorrelationID JMS プロパティーの代わりにこの JMS プロパティーを使用してメッセージを関連付けます。設定されたメッセージは、このプロパティー JMSCorrelationID プロパティーの値でのみ相関し、Camel によって設定されません。		文字列
disableTimeToLive (producer)	このオプションを使用して、ライブ時間を強制的に無効にします。たとえば、JMS でリクエスト/応答を行う場合、Camel はデフォルトで送信されるメッセージの有効期間として requestTimeout の値を使用します。この問題は、送信側および受信側システムにクロックを同期して同期する必要があることです。これは常に簡単にアーカイブできる訳ではありません。したがって、disableTimeToLive=true を使用して、送信されたメッセージで時間がライブ値を設定しないようにすることができます。その後、メッセージは受信側システムで期限切れになりません。詳細は、以下のセクションの「ライブ時間について」を参照してください。	false	boolean
forceSendOriginalMessage (producer)	mapJmsMessage=false を使用する場合、ルート中にヘッダー（get または set）を操作すると、Camel は新しい JMS 宛先に送信する新しい JMS メッセージを送信します。このオプションを true に設定して、Camel が受信した元の JMS メッセージを送信するように強制します。	false	boolean
includeSentJMSMessageID (producer)	InOnly を使用して JMS 宛先に送信する場合にのみ適用されます（例： fire and forget）。このオプションを有効にすると、Camel Exchange は、メッセージが JMS 宛先に送信されたときに JMS クライアントによって使用される実際の JMSMessageID でエンリッチされます。	false	boolean

Name	説明	デフォルト	Type
replyToCacheLevelName (producer)	JMS で要求/応答を実行するときに、リプライコンシューマーの名前でキャッシュレベルを設定します。このオプションは、（一時的ではなく）固定応答キューを使用している場合に限り該当します。Camel はデフォルトで CACHE_CONSUMER（排他的の場合）または shared w/ replyToSelectorName を使用します。replyToSelectorName なしで共有される CACHE_SESSION。IBM WebSphere などの JMS ブローカーによっては、reToCacheLevelName=CACHE_NONE が機能するように設定する必要がある場合があります。注記： 一時キューを使用する場合は CACHE_NONE は許可されず、CACHE_CONSUMER や CACHE_SESSION などの高い値を使用する必要があります。		文字列
replyToDestinationSelector Name (producer)	共有キューの使用時に（一時的な応答キューを使用していない場合は）他のユーザーからの独自のリプライをフィルタリングできるように、固定名を使用して JMS セレクターを設定します。		文字列
streamMessageTypeEnabled (producer)	StreamMessage タイプが有効かどうかを指定します。ファイル、InputStream などのストリーミングタイプのメッセージペイロードは、BytesMessage または StreamMessage として送信されます。このオプションは、使用される種類を制御します。デフォルトでは、BytesMessage を使用して、メッセージペイロード全体をメモリーに読み込みます。このオプションを有効にすると、メッセージペイロードはチャンクのメモリーに読み取られ、各チャンクはデータがなくなるまで StreamMessage に書き込まれます。	false	boolean
allowSerializedHeaders (advanced)	シリアライズされたヘッダーを含めるかどうかを制御します。transferExchange が true の場合にのみ適用されます。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。	false	boolean

Name	説明	デフォルト	Type
asyncStartListener (advanced)	ルートの開始時に JmsConsumer メッセージリスナーを非同期に起動するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの起動中に Camel がブロックされます。このオプションを true に設定すると、ルートの起動が可能になりますが、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用すると、接続が確立されないと、WARN レベルで例外がログに記録され、コンシューマーはメッセージを受信できなくなります。次にルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止するときに JmsConsumer メッセージリスナーを非同期的に停止するかどうか。	false	boolean
destinationResolver (advanced)	独自のリゾルバーを使用できるプラグ可能な org.springframework.jms.support.destination.DestinationResolver（たとえば、JNDI レジストリーから実際の宛先を検索するためなど）。		DestinationResolver
errorHandler (advanced)	メッセージの処理中にキャッチされない例外が発生した場合に呼び出される org.springframework.util.ErrorHandler を指定します。デフォルトでは、errorHandler が設定されていない場合、これらの例外は WARN レベルでログに記録されます。errorHandlerLoggingLevel および errorHandlerLogStackTrace オプションを使用して、ロギングレベルおよびスタックトレースをログに記録するかどうかを設定できます。これにより、カスタム errorHandler をコーディングする必要よりも、設定が非常に簡単になります。		ErrorHandler
exceptionListener (advanced)	基礎となる JMS 例外が通知される JMS 例外リスナーを指定します。		ExceptionListener
headerFilterStrategy (advanced)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
idleConsumerLimit (advanced)	いつでもアイドル状態にできるコンシューマーの数の制限を指定します。	1	int

Name	説明	デフォルト	Type
idleTaskExecutionLimit (advanced)	実行内でメッセージを受信せずに、受信タスクのアイドル実行の制限を指定します。この制限に達すると、タスクはシャットダウンし、他の実行中のタスク（動的なスケジューリングの場合は maxConcurrentConsumers の設定）を受信し続けます。Spring には追加のドキュメントがあります。	1	int
includeAllJMSXProperties (advanced)	JMS から Camel メッセージへのマッピング時に、すべての JMSXxxx プロパティーを含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合は、このオプションは適用されません。	false	boolean
jmsKeyFormatStrategy (advanced)	JMS 鍵をエンコードおよびデコードするためのプラグ可能なストラテジー。これにより、JMS 仕様に準拠します。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーは、ドットとハイフン（. および -）を安全にマーシャリングします。passthrough ストラテジーは、鍵をそのまま残します。JMS ヘッダーキーに不正な文字が含まれるかどうかは気にしない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		文字列
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ（javax.jms.TextMessage など）に自動マッピングするかどうかを指定します。	true	boolean
maxMessagesPerTask (advanced)	タスクごとのメッセージ数。-1 は無制限です。同時コンシューマー（min max など）に範囲を使用する場合、このオプションを使用して値を eg 100 に設定して、作業が少なくなる場合にコンシューマーのスピードを制御することができます。	-1	int
messageConverter (advanced)	javax.jms.Message への/からのマップ方法を制御するため、カスタムの Spring org.springframework.jms.support.converter.MessageConverter を使用します。		MessageConverter
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信するときに Camel が javax.jms.Message オブジェクトの新規インスタンスを作成する際に呼び出される指定の MessageCreatedStrategy を使用します。		MessageCreatedStrategy

Name	説明	デフォルト	Type
messageIdEnabled (advanced)	送信時に、メッセージ ID を追加するかどうかを指定します。これは、JMS ブローカーへのヒントです。JMS プロバイダーがこのヒントを受け入れる場合、これらのメッセージにはメッセージ ID が null に設定されている必要があります。プロバイダーがヒントを無視する場合は、メッセージ ID を通常の一意の値に設定する必要があります。	true	boolean
messageListenerContainerFactory (advanced)	メッセージを消費するために使用する org.springframework.jms.listener.AbstractMessageListenerContainer を決定するために使用される MessageListenerContainerFactory のレジストリー ID。これを設定すると、consumerType が自動的に Custom に設定されます。		MessageListenerContainerFactory
messageTimestampEnabled (advanced)	メッセージの送信時にタイムスタンプをデフォルトで有効にするかどうかを指定します。これは、JMS ブローカーへのヒントです。JMS プロバイダーがこのヒントを受け入れる場合、これらのメッセージにはタイムスタンプがゼロに設定される必要があります。プロバイダーがヒントを無視する場合は、タイムスタンプを通常の値に設定する必要があります。	true	boolean
pubSubNoLocal (advanced)	独自の接続によって公開されるメッセージの配信を抑制するかどうかを指定します。	false	boolean
receiveTimeout (advanced)	メッセージの受信のタイムアウト（ミリ秒単位）。	1000	Long
recoveryInterval (advanced)	リカバリーを試行する間隔を指定します（例： 接続の更新時（ミリ秒単位）。デフォルトは 5000 ミリ秒で、5 秒です。	5000	Long
requestTimeoutCheckerInterval (advanced)	JMS でリクエスト/リプライを行うときに Camel がタイムアウトしたエクスチェンジをチェックする頻度を設定します。デフォルトでは、Camel は 1 秒ごとに 1 回チェックします。ただし、タイムアウトの発生時により迅速に対応する必要がある場合は、この間隔を減らしてより頻繁に確認できます。タイムアウトは、requestTimeout オプションで決定されます。	1000	Long
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
transferException (advanced)	有効にすると、リクエスト応答メッセージング (InOut)を使用し、エクスチェンジがコンシューマー側で失敗した場合、発生した例外は javax.jms.ObjectMessage として応答として返信されます。クライアントが Camel の場合、返された Exception が再スローされます。これにより、Camel JMS をルーティングのブリッジとして使用できます。たとえば、永続キューを使用して堅牢なルーティングを有効にすることができます。transferExchange も有効化されている場合には、このオプションが優先されます。例外がシリアライズ可能である必要があります。コンシューマー側の元の例外は、プロデューサーに戻されたときに org.apache.camel.RuntimeCamelException などの外部例外でラップできます。	false	boolean
transferExchange (advanced)	ボディーとヘッダーだけでなく、ネットワーク上でエクスチェンジを転送できます。次のフィールドが転送されます。本文、Out ボディー、フォールト本文、In ヘッダー、Out ヘッダー、フォールトヘッダー、エクスチェンジプロパティー、エクスチェンジ例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。プロデューサーとコンシューマー側の両方でこのオプションを有効にする必要があります。そのため、Camel はペイロードが通常のペイロードではなく Exchange を認識します。	false	boolean
transferFault (advanced)	有効にすると、Request Reply messaging(InOut)を使用し、Exchange がコンシューマー側で SOAP fault（例外ではない）で失敗した場合、MessageisFault（）のフォールトフラグは、キー org.apache.camel.component.jms.JmsConstantsJMS_TRANSFER_FAULTJMS_TRANSFER_FAULTJMS_TRANSFER_FAULTJMS_TRANSFER_FAULT として応答として返信されます。クライアントが Camel の場合、返されるフォールトフラグはリンク org.apache.camel.MessagesetFault(boolean)に設定されます。これは、cxf や spring-ws などの SOAP ベースの障害をサポートする Camel コンポーネントを使用する際に有効にすることができます。	false	boolean
useMessageIDAs CorrelationID (advanced)	InOut メッセージに対して JMSCorrelationID として常に JMSMessageID を使用するかどうかを指定します。	false	boolean

Name	説明	デフォルト	Type
waitForProvisionCorrelationToBeUpdatedCounter (advanced)	request/reply over JMS を実行し、オプション useMessageIDAsCorrelationID が有効になっている場合に、プロビジョニング ID が実際の相関 ID に更新されるまで待機する回数。	50	int
waitForProvisionCorrelationToBeUpdatedThreadSleeping Time (advanced)	プロビジョニングの相関 ID の更新を待機する間に毎回スリープ状態になる間隔（ミリ秒単位）。	100	Long
errorHandlerLoggingLevel (logging)	キャッチされない例外についてのデフォルトの errorHandler ロギングレベルの設定を可能にします。	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	デフォルトの errorHandler によってスタックトレースをログに記録するかどうかを制御できます。	true	boolean
パスワード（セキュリティー）	ConnectionFactory で使用するパスワード。また、ユーザー名/パスワードを ConnectionFactory に直接設定することもできます。		文字列
ユーザー名（セキュリティー）	ConnectionFactory で使用するユーザー名。また、ユーザー名/パスワードを ConnectionFactory に直接設定することもできます。		文字列
トランザクション（トランザクション）	トランザクションモードを使用するかどうかを指定します。	false	boolean
lazyCreateTransaction Manager (transaction)	true の場合、transacted=true オプション時に transactionManager が挿入されていない場合、Camel は JmsTransactionManager を作成します。	true	boolean
transactionManager (transaction)	使用する Spring トランザクションマネージャー。		PlatformTransaction Manager
transactionName (transaction)	使用するトランザクションの名前。		文字列
transactionTimeout (transaction)	トランザクションモードを使用している場合は、トランザクションのタイムアウト値（秒単位）。	-1	int

5.3. 用途

AMQP コンポーネントは JMS コンポーネントから継承されるため、前者の使用は後者とほぼ同じです。

AMQP コンポーネントの使用

```
// Consuming from AMQP queue
from("amqp:queue:incoming").
  to(...);

// Sending message to the AMQP topic
from(...).
  to("amqp:topic:notify");
```

5.4. AMQP コンポーネントの設定

Camel 2.16.1 以降、AMQPComponent#amqp10Component(String connectionURI) ファクトリーメソッドを使用して、事前設定されたトピック接頭辞を持つ AMQP 1.0 コンポーネントを返すこともできます。

AMQP 1.0 コンポーネントの作成

```
 AMQPComponent amqp =
AMQPComponent.amqp10Component("amqp://guest:guest@localhost:5672");
```

Camel 2.17 以降、 AMQPComponent#amqp10Component(String connectionURI) ファクトリーメソッドが AMQPComponent#amqpComponent(String connectionURI) の代わりに非推奨になっていることに注意してください。

AMQP 1.0 コンポーネントの作成

```
AMQPComponent amqp = AMQPComponent.amqpComponent("amqp://localhost:5672");

AMQPComponent authorizedAmqp =
AMQPComponent.amqpComponent("amqp://localhost:5672", "user", "password");
```

Camel 2.17 以降では、AMQP コンポーネントを自動的に設定するために、 org.apache.camel.component.amqp.AMQPConnectionDetails のインスタンスをレジストリーに追加することもできます。たとえば、Spring Boot の場合は Bean を定義するだけです。

AMQP コネクションの詳細自動設定

```
@Bean
AMQPConnectionDetails amqpConnection() {
  return new AMQPConnectionDetails("amqp://localhost:5672");
}

@Bean
AMQPConnectionDetails securedAmqpConnection() {
  return new AMQPConnectionDetails("amqp://lcoalhost:5672", "username", "password");
}
```

同様に、Camel-CDI を使用する際に CDI プロデューサーメソッドを使用することもできます。

CDI の AMQP コネクションの詳細自動設定

```
@Produces
AMQPConnectionDetails amqpConnection() {
  return new AMQPConnectionDetails("amqp://localhost:5672");
}
```

Camel プロパティー に依存して AMQP コネクションの詳細を読み取ることもできます。Factory メソッド AMQPConnectionDetails.discoverAMQP（）は 、以下のスニペットに示すように Kubernetes のような規則の Camel プロパティーの読み取りを試みます。

AMQP コネクションの詳細自動設定

```
export AMQP_SERVICE_HOST = "mybroker.com"
export AMQP_SERVICE_PORT = "6666"
export AMQP_SERVICE_USERNAME = "username"
export AMQP_SERVICE_PASSWORD = "password"

...

@Bean
AMQPConnectionDetails amqpConnection() {
  return AMQPConnectionDetails.discoverAMQP();
}
```

AMQP 固有のオプションの有効化

たとえば、amqp.traceFrames を有効にする必要がある場合は、以下の例のように URI にオプショ

ンを追加して実行できます。

```
AMQPComponent amqp = AMQPComponent.amqpComponent("amqp://localhost:5672?
amqp.traceFrames=true");
```

リファレンスは、QPS ID JMS クライアント設定を参照してください。

5.5. トピックの使用

camel-amqp で機能するトピックを使用するには、以下のように topic: // をトピック プレフィックスとして使用するようにコンポーネントを設定する必要があります。

```
<bean id="amqp" class="org.apache.camel.component.amqp.AmqpComponent">
  <property name="connectionFactory">
    <bean class="org.apache.qpid.jms.JmsConnectionFactory" factory-
method="createFromURL">
      <property name="remoteURI" value="amqp://localhost:5672" />
      <property name="topicPrefix" value="topic://" />  <!-- only necessary when connecting to
ActiveMQ over AMQP 1.0 -->
    </bean>
  </property>
 </bean>
```

AMQPComponent#amqpComponent（）メソッドおよび AMQP ConnectionDetails の両方は、トピック接頭辞でコンポーネントを事前設定するため、明示的に設定する必要がないことに注意してください。

5.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第6章 APNS コンポーネント

Camel バージョン 2.8 から利用可能

apns コンポーネントは、iOS デバイスに通知を送信するために使用されます。apns コンポーネントは javapns ライブラリーを使用します。
コンポーネントは、Apple Push Notification Servers(APNS)への通知の送信や、サーバーからのフィードバックの消費をサポートします。

コンシューマーには、デフォルトでポーリング用に 3600 秒が設定されます。これは、Apple Push Notification サーバーからのフィードバックストリームを時間から時間にだけ使用するのがベストプラクティスであるためです。たとえば、サーバーがあふれるのを避けるために 1 時間ごとです。

フィードバックストリームは、アクティブでないデバイスに関する情報を提供します。モバイルアプリケーションが頻繁に使用されていない場合にのみ、この情報を取得する必要があるのは 1 時間ごとのみです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-apns</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

6.1. URI 形式

通知を送信するには、以下の手順を実施します。

```
apns:notify[?options]
```

フィードバックをお寄せいただくには、以下の手順を実施します。

```
apns:consumer[?options]
```

6.2. オプション

APNS コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
apnsService (common)	使用する ApnsService が **必要です**。org.apache.camel.component.apns.factory.ApnsServiceFactory を使用して ApnsService をビルドできます。		ApnsService
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

APNS エンドポイントは、URI 構文を使用して設定します。

```
apns:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

6.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	エンドポイントの名前		文字列

6.2.2. クエリーパラメーター（20 パラメーター）：

Name	説明	デフォルト	Type
トークン （共通）	通知するデバイスに関連するトークンを静的に宣言する場合は、このプロパティーを設定します。トークンはコンマで区切ります。		文字列

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int

Name	説明	デフォルト	Type
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

URI にクエリーオプションを追加するには、?option=value&option=value&...

6.2.3. コンポーネント

ApnsComponent は、com.notnoop.apns.ApnsService で設定する必要があります。サービスは、org.apache.camel.component.apns.factory.ApnsServiceFactory を使用して作成および設定できます。例については、こちらを参照してください。テストソースコード にも、さらに必要です。

6.2.3.1. SSL の設定

セキュアな接続を使用するには、コンポーネントの設定に使用される org.apache.camel.util.jsse.SSLContextParameters のインスタンスを org.apache.camel.component.apns.factory.ApnsServiceFactory にインジェクトする必要があります。サンプルのテストリソースを参照してください。SSL の例

6.3. データフォーマットの交換

Camel が非アクティブなデバイスに対応するフィードバックデータを取得すると、List of InactiveDevice オブジェクトが取得されます。取得される一覧の各 InactiveDevice オブジェクトは In ボディーとして設定され、コンシューマーエンドポイントによって処理されます。

6.4. メッセージヘッダー

Camel Apns はこれらのヘッダーを使用します。

プロパティー	デフォルト	説明
CamelApnsTokens		デフォルトでは空です。
CamelApnsMessageType	**文字列、ペイロード、APNS_NOTIFICATION**	メッセージタイプに PAYLOAD を選択すると、メッセージは APNS ペイロードと見なされ、そのまま送信されます。STRING を選択すると、メッセージは APNS ペイロードとして変換されます。**Camel 2.16** 以降 APNS_NOTIFICATION は、メッセージボディーを com.notnoop.apns.ApnsNotification タイプとして送信するために使用されます。

6.5. APNSSERVICEFACTORY ビルダーコールバック

ApnsServiceFactory には、デフォルトの ApnsServiceBuilder インスタンスの設定（または置き換え）に使用できる空のコールバックメソッドが同梱されます。メソッドの署名は以下のようになります。

```
protected ApnsServiceBuilder configureServiceBuilder(ApnsServiceBuilder serviceBuilder);
```

また、以下のように使用できます。

```
ApnsServiceFactory proxiedApnsServiceFactory = new ApnsServiceFactory(){

  @Override
  protected ApnsServiceBuilder configureServiceBuilder(ApnsServiceBuilder serviceBuilder) {
    return serviceBuilder.withSocksProxy("my.proxy.com", 6666);
  }

};
```

6.6. サンプル

6.6.1. Camel Xml ルート

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

    <!-- Replace by desired values -->
    <bean id="apnsServiceFactory"
class="org.apache.camel.component.apns.factory.ApnsServiceFactory">

        <!-- Optional configuration of feedback host and port -->
        <!-- <property name="feedbackHost" value="localhost" /> -->
        <!-- <property name="feedbackPort" value="7843" /> -->

        <!-- Optional configuration of gateway host and port -->
        <!-- <property name="gatewayHost" value="localhost" /> -->
        <!-- <property name="gatewayPort" value="7654" /> -->

        <!-- Declaration of certificate used -->
            <!-- from Camel 2.11 onwards you can use prefix: classpath:, file: to refer to load the
certificate from classpath or file. Default it classpath -->
        <property name="certificatePath" value="certificate.p12" />
        <property name="certificatePassword" value="MyCertPassword" />
```

```
        <!-- Optional connection strategy - By Default: No need to configure -->
        <!-- Possible options: NON_BLOCKING, QUEUE, POOL or Nothing -->
        <!-- <property name="connectionStrategy" value="POOL" /> -->
        <!-- Optional pool size -->
        <!-- <property name="poolSize" value="15" /> -->

        <!-- Optional connection strategy - By Default: No need to configure -->
        <!-- Possible options: EVERY_HALF_HOUR, EVERY_NOTIFICATION or Nothing
(Corresponds to NEVER javapns option) -->
        <!-- <property name="reconnectionPolicy" value="EVERY_HALF_HOUR" /> -->
    </bean>

    <bean id="apnsService" factory-bean="apnsServiceFactory" factory-method="getApnsService" />

    <!-- Replace this declaration by wanted configuration -->
    <bean id="apns" class="org.apache.camel.component.apns.ApnsComponent">
        <property name="apnsService" ref="apnsService" />
    </bean>

    <camelContext id="camel-apns-test" xmlns="http://camel.apache.org/schema/spring">
            <route id="apns-test">
                <from uri="apns:consumer?initialDelay=10&amp;delay=3600&amp;timeUnit=SECONDS"
/>
                <to uri="log:org.apache.camel.component.apns?showAll=true&amp;multiline=true" />
                <to uri="mock:result" />
            </route>
    </camelContext>

</beans>
```

6.6.2. Camel Java ルート

camel コンテキストの作成およびプログラムによる apns コンポーネントの宣言

```
    protected CamelContext createCamelContext() throws Exception {
        CamelContext camelContext = super.createCamelContext();

        ApnsServiceFactory apnsServiceFactory = new ApnsServiceFactory();
        apnsServiceFactory.setCertificatePath("classpath:/certificate.p12");
        apnsServiceFactory.setCertificatePassword("MyCertPassword");

        ApnsService apnsService = apnsServiceFactory.getApnsService(camelContext);

        ApnsComponent apnsComponent = new ApnsComponent(apnsService);
        camelContext.addComponent("apns", apnsComponent);

        return camelContext;
    }
```

[[APNS-ApnsProducer-iOStargetdevicedynamicallyconfiguredviaheader:"CamelApnsTokens"]] ApnsProducer - iOS

target device dynamically configured via header: "CamelApnsTokens"

```
protected RouteBuilder createRouteBuilder() throws Exception {
    return new RouteBuilder() {
        public void configure() throws Exception {
            from("direct:test")
                .setHeader(ApnsConstants.HEADER_TOKENS, constant(IOS_DEVICE_TOKEN))
                .to("apns:notify");
            }
    }
}
```

ApnsProducer: iOS ターゲットデバイスが uri 経由で静的に設定される

```
protected RouteBuilder createRouteBuilder() throws Exception {
    return new RouteBuilder() {
        public void configure() throws Exception {
            from("direct:test").
            to("apns:notify?tokens=" + IOS_DEVICE_TOKEN);
        }
    };
}
```

ApnsConsumer

```
from("apns:consumer?initialDelay=10&delay=3600&timeUnit=SECONDS")
    .to("log:com.apache.camel.component.apns?showAll=true&multiline=true")
    .to("mock:result");
```

6.7. 関連項目

- コンポーネント
- Endpoint * APNS の使用に関するブログ(french)

第7章 ASN.1 FILE DATAFORMAT

Camel バージョン 2.20 で利用可能

ASN.1 Data Format Data Format [Intoduction to ASN.1](https://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx)は、複雑または非常にシンプルかに関係なく、アプリケーションの Bouncy Castle の bcprov-jdk15on ライブラリーと jASN.1 の java コンパイラーに基づいた Camel Framework のデータフォーマットの実装です。メッセージは、プレーンな Java POJO にアンマーシャリング（単純な Java POJO に結合）できます。Camel のルーティングエンジンやデータ変換の助けにより、POJO と協調してカスタマイズしたフォーマットを適用し、他の Camel コンポーネントを呼び出して、アップストリームシステムにメッセージを変換および送信することができます。

7.1. ASN.1 データフォーマットのオプション

ASN.1 ファイルデータ形式は、以下に示す 3 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
usingIterator	**false**	**ブール値**	asn1 ファイルに 1 つ以上のエントリーがある場合、このオプションを true に設定すると、Splitter EIP との作業が可能になり、ストリーミングモードでイテレーターを使用してデータを分割できます。
clazzName		**文字列**	アンマーシャリング時に使用するクラスの名前
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

7.2. アンマーシャリング

ASN.1 の構造化メッセージをアンマーシャリングする方法は 3 つあります。（通常はバイナリーファイル）

この最初の例では、BER ファイルペイロードを OutputStream にアンマーシャリングし、モックエンドポイントに送信します。

```
from("direct:unmarshal").unmarshal(asn1).to("mock:unmarshal");
```

2 つ目の例では、Split EIP を使用して BER ファイルペイロードをバイトアレイにアンマーシャリングします。Split EIP を適用する理由は、通常は各 BER ファイルまたは（ASN.1 構造化ファイル）には、処理する複数のレコードが含まれ、Split EIP は実際に ASN1Primitive's インスタンス（Bouncy Castle の ASN.1 の使用による）バイトアレイとしてファイルに各レコードを取得するのに役立ちます。 bcprov-jdk15on ライブラリーのサポート）Byte 配列は、ASN1Primitive のパブリック静的メソッド(ASN1Primitive.fromByteArray)を利用して ASN1Primitive に変換できます。usingIterator=trueを設定する必要があることに注意してください。

```
from("direct:unmarshal").unmarshal(asn1).split(body(Iterator.class)).streaming().to("mock:unmarshal");
```

最後の例では、Split EIP を使用して、BER ファイルペイロードをプレーンな古い Java オブジェクトにアンマーシャリングします。Split EIP を適用する理由は、すでに前述の例で紹介されています。そのような理由を念頭に置いてください。この例では、クラスの完全修飾名またはデータフォーマットで <YourObject>.class 参照を設定する必要もあります。ここで注意すべき重要なことは、jasn1 コンパイラーでオブジェクトを生成する必要があることです。このコンパイラーは、ASN.1 構造の Java オブジェクト表現を生成する優れたツールです。jasn1 コンパイラーのリファレンスの使用方法は、[JASN.1 プロジェクトページ](https://www.openmuc.org/asn1/)を参照してください。また、maven の exec プラグインを利用してコンパイラーがどのように呼び出されるかも確認してください。たとえば、このデータ形式のユニットテストでは、ASN.1 構造(TestSMSBerCdr.asn1)の例は src/test/resources/asn1_structure に追加されます。jasn1 コンパイラーが呼び出され、java オブジェクトの表現が ${basedir}/target/generated/src/test/java で生成 されます。この例では、モックエンドポイントまたはエンドポイントのどれでも POJO インスタンスを取得します。

```
from("direct:unmarshaldsl")
        .unmarshal()
        .asn1("org.apache.camel.dataformat.asn1.model.testsmscbercdr.SmsCdr")
        .split(body(Iterator.class)).streaming()
.to("mock:unmarshaldsl");
```

7.3. 依存関係

camel ルートで ASN.1 データ形式を使用するには、このデータ形式を実装する camel-asn1 の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加できます。バージョン番号は最新の最新のリリースに置き換えてください（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-asn1</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第8章 アスタリスクコンポーネント

Camel バージョン 2.18 から利用可能

アスタリスク： コンポーネントを使用すると、asterisk-javaを使用して Asterisk PBX Server http://www.asterisk.org/ で簡単に動作できます。

このコンポーネントは 、Asterisk Manager インターフェースとのインターフェースに役立ちます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-asterisk</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

8.1. URI 形式

```
asterisk:name[?options]
```

8.2. オプション

Asterisk コンポーネントにはオプションがありません。

Asterisk エンドポイントは、URI 構文を使用して設定します。

```
asterisk:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

8.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	**必要な** 論理名		文字列

8.2.2. クエリーパラメーター（8 パラメーター）：

Name	説明	デフォルト	Type
hostname (common)	**必須** サーバーのホスト名		文字列
パスワード (common)	**必要な** ログインパスワード		文字列
username (common)	**必要な** ログインユーザー名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
アクション （プロデューサー）	キューのステータス、センッピア、エクステンションの状態の取得などの実行するアクション。		AsteriskAction
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

8.3. アクション

サポートされるアクションは以下のとおりです。

- QUEUE_STATUS、Queue ステータス
- SIP_PEERS、SIP ピアの一覧表示
- EXTENSION_STATE、拡張ステータスの確認

第9章 ATMOS コンポーネント

Camel バージョン 2.15 から利用可能

Camel-Atmos は、Atmos クライアントを使用して ViPR オブジェクトデータサービスと連携できる Apache Camel のコンポーネントです。

```
from("atmos:foo/get?remotePath=/path").to("mock:test");
```

9.1. オプション

Atmos コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
fullTokenId (security)	Atmos クライアントに渡すトークン ID		文字列
secretKey (security)	Atmos クライアントに渡すシークレットキー		文字列
URI （詳細）	接続する Atmos クライアントのサーバーの URI。		文字列
sslValidation (security)	Atmos クライアントが SSL 検証を実行するかどうか。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atmos エンドポイントは URI 構文を使用します。

```
atmos:name/operation
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

9.1.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
name	Atmos name		文字列
operation	実行に **必要な** 操作		AtmosOperation

9.1.2. クエリーパラメーター（12 パラメーター）：

Name	説明	デフォルト	Type
enableSslValidation (common)	Atmos SSL validation	false	boolean
fullTokenId (common)	Atmos client fullTokenId		文字列
localPath (common)	ファイルを配置するローカルパス		文字列
newRemotePath (common)	ファイルの移動時の Atmos に新しいパス		文字列
クエリー （共通）	Atmos での検索クエリー		文字列
remotePath (common)	ファイルを Atmos に配置する場所		文字列
secretKey (common)	Atmos shared secret		文字列
URI （共通）	Atomos server uri		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

9.2. 依存関係

camel ルートで Atmos を使用するには、このデータ形式を実装する camel-atmos に依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atmos</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

9.3. 統合

atmos インテグレーションを確認すると、コンシューマー GetConsumer（ScheduledPollConsumer のタイプ）があります。

- **Get**

プロデューサーには 4 つのタイプがあり、

- Get
- del
- Move
- put

9.4. 例

以下の例はテストから取得されます。

```
from("atmos:foo/get?remotePath=/path").to("mock:test");
```

ここでは、コンシューマーの例です。remotePath は、データを読み取り、Camel エクスチェンジをプロデューサーアンダートに関するパスを表します。このコンポーネントは、これと他の操作すべてに atmos クライアント API を使用します。

```
from("direct:start")
.to("atmos://get?remotePath=/dummy/dummy.txt")
.to("mock:result");
```

ここでは、プロデューサーのサンプルです。remotePath は、Bevis オブジェクトデータサービスで操作が実行されるパスを表します。プロデューサーでは、Operation(Get,Del,Move,Put)を ViPR オブジェクトデータサービスで実行し、結果は camel エクスチェンジのヘッダーに設定されます。

操作に関して、以下のヘッダーが camel exhange に設定されます。

```
DOWNLOADED_FILE, DOWNLOADED_FILES, UPLOADED_FILE, UPLOADED_FILES,
FOUND_FILES, DELETED_PATH, MOVED_PATH;
```

9.5. 関連項目

- Configuring Camel (Camel の設定)

- **コンポーネント**
- **エンドポイント**
- **はじめに**

第10章 ATMO336 WEBSOCKET COMPONENT

Camel バージョン 2.14 から利用可能

atmo336-websocket: コンポーネントは、（外部クライアントから websocket 接続を受け入れるサーブレットとして）Websocket 経由で外部クライアントと通信するサーブレットに Websocket ベースのエンドポイントを提供します。
コンポーネントは SERVLET コンポーネントを使用し、 Atmo336 ライブラリーを 使用してさまざまなサーブレットコンテナーで Websocket トランスポートをサポートします（例： Jetty、Tomcat、…）。

埋め込み Jetty サーバーを起動する Websocket コンポーネントとは異なり、このコンポーネントはコンテナーのサーブレットプロバイダーを使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atmosphere-websocket</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

10.1. ATMO336-WEBSOCKET オプション

Atmo336 Websocket コンポーネントは、以下に示す 8 個のオプションをサポートします。

Name	説明	デフォルト	Type
servletName (common)	使用するサーブレットのデフォルト名。デフォルトの名前は CamelServlet です。		文字列
httpRegistry (common)	カスタムの org.apache.camel.component.servlet.HttpRegistry を使用します。		HttpRegistry

Name	説明	デフォルト	Type
attachmentMultipart Binding (common)	マルチパート/フォームデータを Camel Exchange で添付として自動バインドするかどうか。attachmentMultipartBinding=true オプションおよび disableStreamCache=false オプションは併用できません。AttachmentMultipartBinding を使用するには disableStreamCache を削除します。サーブレットの使用時にこの設定を有効にするためにサーブレット固有の設定を必要とする可能性があるため、これはデフォルトで無効になります。	false	boolean
httpBinding (advanced)	カスタムの HttpBinding を使用して Camel メッセージと HttpClient 間のマッピングを制御する場合。		HttpBinding
httpConfiguration (advanced)	共有 HttpConfiguration をベース設定として使用します。		HttpConfiguration
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用する場合に java のシリアライズを許可するかどうか。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atmo336 Websocket エンドポイントは、URI 構文を使用して設定します。

```
atmosphere-websocket:servicePath
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

10.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
servicePath	Websocket エンドポイントに **必要な** 名前		文字列

10.1.2. クエリーパラメーター（37 パラメーター）：

Name	説明	デフォルト	Type
チャンク （共通）	このオプションが false の場合、サーブレットは HTTP ストリーミングを無効にし、応答に content-length ヘッダーを設定します。	true	boolean
disableStreamCache (common)	Servlet からの raw 入力ストリームがキャッシュされているかどうかを決定します（Camel はストリームをメモリー内/オーバーフローでファイル、ストリームキャッシング）キャッシュに読み取ります。デフォルトでは、Camel は Servlet 入力ストリームをキャッシュして、複数回ロードし、Camel がストリームからすべてのデータを取得できるようにします。ただし、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。たとえば、ファイルまたは他の永続ストアに直接ストリーミングする場合などに、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。DefaultHttpBinding は、ストリームの読み取りを複数回サポートするために、このオプションが false の場合は、リクエスト入力ストリームをストリームキャッシュにコピーし、メッセージボディーに配置します。Servlet を使用してエンドポイントをブリッジ/プロキシーする場合、このオプションを有効にしてパフォーマンスを向上することを検討してください。メッセージペイロードを複数回読み取る必要がない場合は、このオプションを有効にします。http/http4 プロデューサーは、デフォルトでは応答本体ストリームをキャッシュします。このオプションを true に設定すると、プロデューサーは応答ボディーストリームをキャッシュしませんが、応答ストリームをメッセージボディーとして使用します。	false	boolean
headerFilterStrategy (common)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
sendToAll (common)	all（ブロードキャスト）に送信するか、または単一のレシーバーに送信するか。	false	boolean

Name	説明	デフォルト	Type
transferException (common)	有効で Exchange がコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object のコンテンツタイプとして応答でシリアライズされたかどうか。プロデューサー側では、例外は HttpOperationFailedException ではなくデシリアライズされ、そのままスローされます。原因となる例外はシリアライズされている必要があります。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
useStreaming (common)	ストリーミングがデータを複数のテキストフラグメントとして送信できるようにするには、以下を行います。	false	boolean
httpBinding (common)	カスタムの HttpBinding を使用して Camel メッセージと HttpClient 間のマッピングを制御する場合。		HttpBinding
非同期 （コンシューマー）	非同期モードで動作するようにコンシューマーを設定する	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
httpMethodRestrict (consumer)	HttpMethod が一致する場合にのみ消費（GET/POST/PUT など）を許可するために使用されます。複数のメソッドはカンマで区切って指定できます。		文字列
matchOnUriPrefix (consumer)	完全一致がない場合、コンシューマーが URI プレフィックスに一致することでターゲットコンシューマーの検索を試行するかどうか。	false	boolean
responseBufferSize (consumer)	javax.servlet.ServletResponse でカスタムバッファーサイズを使用します。		整数
servletName (consumer)	使用するサーブレットの名前	Camel Servlet	文字列

Name	説明	デフォルト	Type
attachmentMultipartBinding (consumer)	マルチパート/フォームデータを Camel Exchange で添付として自動バインドするかどうか。attachmentMultipartBinding=true オプションおよび disableStreamCache=false オプションは併用できません。AttachmentMultipartBinding を使用するには disableStreamCache を削除します。サーブレットの使用時にこの設定を有効にするためにサーブレット固有の設定を必要とする可能性があるため、これはデフォルトで無効になります。	false	boolean
eagerCheckContentAvailable (consumer)	content-length ヘッダーが 0 の場合に、HTTP リクエストにコンテンツがあるかどうかをアクティブにチェックするかどうか。これは、HTTP クライアントがストリーミングデータを送信しない場合に有効にすることができます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
optionsEnabled (consumer)	このサーブレットコンシューマーに HTTP OPTIONS を有効にするかどうかを指定します。デフォルトでは OPTIONS はオフになっています。	false	boolean
traceEnabled (consumer)	このサーブレットコンシューマーに対して HTTP TRACE を有効にするかどうかを指定します。デフォルトでは、TRACE はオフになっています。	false	boolean
bridgeEndpoint (producer)	オプションが true の場合、HttpProducer は Exchange.HTTP_URI ヘッダーを無視し、リクエストにエンドポイントの URI を使用します。また、ththExceptionOnFailure オプションを false に設定して、HttpProducer がすべての障害応答を返せるようにすることもできます。	false	boolean
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを指定します。デフォルトでは connectionClose は false です。	false	boolean

Name	説明	デフォルト	Type
copyHeaders (producer)	このオプションが true の場合、コピーストラテジーに従って IN エクスチェンジヘッダーが OUT エクスチェンジヘッダーにコピーされます。これを false に設定し、HTTP 応答からのヘッダーのみを含めることができます（IN ヘッダーは伝播しません）。	true	boolean
httpMethod (producer)	使用する HTTP メソッドを設定します。HttpMethod ヘッダーが設定されている場合は、このオプションをオーバーライドできません。		HttpMethods
ignoreResponseBody (producer)	このオプションが true の場合、http プロデューサーは応答本体を読み取りせず、入力ストリームをキャッシュします。	false	boolean
preserveHostHeader (producer)	オプションが true の場合、HttpProducer は Host ヘッダーを現在のエクスチェンジ Host ヘッダーに含まれる値に設定します。これは、ダウンストリームサーバーが受け取る Host ヘッダーを使用してアップストリームクライアントが呼び出した URL を反映させたいリバースプロキシーアプリケーションで有用です。これにより、Host ヘッダーを使用するアプリケーションがプロキシーされるサービスの正確な URL を生成することができます。	false	boolean
throwExceptionOnFailure (producer)	リモートサーバーからの応答に失敗した場合に HttpOperationFailedException のスローを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
cookieHandler (producer)	HTTP セッションを維持するためのクッキーハンドラーの設定		CookieHandler
okStatusCodeRange (producer)	正常な応答とみなされるステータスコード。値は含まれます。コンマで区切られた複数の範囲を定義できます（例： 200-204,209,301-304）。各範囲は、1 つの数字またはダッシュを含む from から でなければなりません。	200-299	文字列
urlRewrite (producer)	**非推奨** の org.apache.camel.component.http.UrlRewrite への参照。ブリッジ/プロキシーエンドポイントの実行時に URL を書き換えることができるようになりました。詳細は、http://camel.apache.org/urlrewrite.htmlを参照してください。		UrlRewrite

Name	説明	デフォルト	Type
mapHttpMessageBody (advanced)	このオプションが true の場合、エクスチェンジの IN エクスチェンジボディーは HTTP ボディーにマッピングされます。false に設定すると HTTP マッピングが回避されます。	true	boolean
mapHttpMessageFormUrlEncodedBody (advanced)	このオプションが true の場合、エクスチェンジの IN Exchange Form Encoded body が HTTP にマッピングされます。これを false に設定すると、HTTP Form Encoded body マッピングを回避します。	true	boolean
mapHttpMessageHeaders (advanced)	このオプションが true の場合、エクスチェンジの IN エクスチェンジヘッダーは HTTP ヘッダーにマッピングされます。false に設定すると、HTTP ヘッダーのマッピングが回避されます。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
proxyAuthScheme (proxy)	使用するプロキシー認証スキーム		文字列
proxyHost (proxy)	使用するプロキシーホスト名		文字列
proxyPort (proxy)	使用するプロキシーポート		int
authHost (security)	NTML で使用する認証ホスト		文字列

10.2. URI 形式

```
atmosphere-websocket:///relative path[?options]
```

10.3. WEBSOCKET でのデータの読み取りおよび書き込み

atmopshere-websocket エンドポイントは、エンドポイントがプロデューサーまたはコンシューマーとして設定されるかどうかに応じて、データをソケットから書き込むか、またはソケットから読み取ることができます。

10.4. 読み取りまたは書き込みデータの URI の設定

以下のルートでは、Camel は指定された Websocket 接続から読み取られます。

```
from("atmosphere-websocket:///servicepath")
        .to("direct:next");
```

同等の Spring サンプルは次のとおりです。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="atmosphere-websocket:///servicepath"/>
    <to uri="direct:next"/>
  </route>
</camelContext>
```

以下のルートでは、Camel は指定された Websocket 接続から読み取られます。

```
from("direct:next")
        .to("atmosphere-websocket:///servicepath");
```

同等の Spring サンプルは次のとおりです。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:next"/>
    <to uri="atmosphere-websocket:///servicepath"/>
  </route>
</camelContext>
```

10.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

- SERVLET
- AHC-WS * Websocket

第11章 ATOM コンポーネント

Camel バージョン 1.2 で利用可能

atom: コンポーネントは、ポーリング Atom フィードに使用されます。

Camel はデフォルトで 60 秒ごとにフィードをポーリングします。
注記： コンポーネントは現在、ポーリング（時間がかかる）フィードのみをサポートしています。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atom</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

11.1. URI 形式

```
atom://atomUri[?options]
```

atomUri は、ポーリングする Atom フィードの URI です。

11.2. オプション

Atom コンポーネントにはオプションがありません。

Atom エンドポイントは、URI 構文を使用して設定します。

```
atom:feedUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

11.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
feedUri	ポーリングに **必要な** フィードの URI。		文字列

11.2.2. クエリーパラメーター（27 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
feedHeader (consumer)	フィードオブジェクトをヘッダーとして追加するかどうかを設定します。	true	boolean
Filter （コンシューマー）	エントリーのフィルタリングを使用するかどうかを設定します。	true	boolean
lastUpdate (consumer)	atom フィードからのエントリーのフィルタリングに使用するタイムスタンプを設定します。このオプションは splitEntries と併用されます。		Date
パスワード （コンシューマー）	HTTP フィードからのポーリング時に使用するパスワードを設定します。		文字列
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
sortEntries (consumer)	発行日でエントリーをソートするかどうかを設定します。splitEntries = true の場合にのみ機能します。	false	boolean
splitEntries (consumer)	エントリーを個別に送信するかどうか、またはフィード全体を 1 つのメッセージとして送信すべきかどうかを設定します。	true	boolean

Name	説明	デフォルト	Type
throttleEntries (consumer)	1 つのフィードポーリングで特定されるすべてのエントリーを即座に配信するかどうかを設定します。true の場合、consumer.delay ごとに 1 つのエントリーのみを処理します。splitEntries = true の場合にのみ該当します。	true	boolean
ユーザー名 （コンシューマー）	HTTP フィードからのポーリング時に使用するユーザー名を設定します。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int

Name	説明	デフォルト	Type
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

URI にクエリーオプションを追加するには、?option=value&option=value&...

11.3. データフォーマットの交換

Camel は、エントリーが含まれる返されたエクスチェンジに In ボディーを設定します。Split Entries フラグに応じて、 Camel は 1 つのエントリーまたは List< Entry > を返します。

オプション	値	動作
splitEntries	true	現在処理中のフィードの単一エントリーのみが設定され、Exchange.in.body(Entry)が設定されます。
splitEntries	false	フィードからのエントリー一覧全体が設定されます：exchange.in.body(List<Entry>)

Camel は In ヘッダーに Feed オブジェクト を設定できます（f feedHeader オプションを参照）、これを無効にします。

11.4. メッセージヘッダー

Camel atom はこれらのヘッダーを使用します。

ヘッダー	説明
Camel Atom Feed	org.apache.abdera.model.Feed オブジェクトを消費すると、このヘッダーに設定されます。

11.5. サンプル

この例では、James Strachan のブログをポーリングしています。

```
from("atom://http://macstrac.blogspot.com/feeds/posts/default").to("seda:feeds");
```

この例では、SEDA キューのような優れたブログだけにフィルターをかけたいとします。サンプルは、コンテナーで実行されない、または Spring を使用しない Camel スタンドアロンの設定方法も示しています。

11.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- RSS

第12章 ATOMIX MAP COMPONENT

Camel バージョン 2.20 で利用可能

camel atomix-map コンポーネントを使用すると、Atomix の Distributed Map コレクションと連携できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

12.1. URI 形式

```
atomix-map:mapName
```

12.2. オプション

Atomix Map コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有コンポーネントの設定		AtomixMapConfiguration
atomix (common)	共有 AtomixClient インスタンス		AtomixClient
ノード (common)	AtomixClient が接続するノード		リスト
configurationUri (common)	AtomixClient 設定へのパス		文字列
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix Map エンドポイントは、URI 構文を使用して設定します。

```
atomix-map:resourceName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

12.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceName	**必須**。分散リソース名		文字列

12.2.2. クエリーパラメーター（18 パラメーター）：

Name	説明	デフォルト	Type
atomix (common)	使用する Atomix インスタンス		Atomix
configurationUri (common)	Atomix 設定 URI。		文字列
defaultAction (common)	デフォルトの動作。	PUT	アクション
キー (common)	ヘッダーに何も設定されていない場合や、特定のキーのイベントをリッスンする場合に使用するキー。		オブジェクト
ノード (common)	クラスターを構成するノードのアドレス。		文字列
resultHeader (common)	悪いヘッダーには結果が含まれます。		文字列
トランスポート（共通）	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
TTL (common)	リソーススロット。		Long

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトリソース設定。		プロパティー
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		プロパティー
ephemeral （詳細）	ローカルメンバーがグループを PersistentMember として参加させるかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取ると、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	一貫性レベルの読み取り。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソースの設定。		マップ
resourceOptions (advanced)	ローカルリソースの設定		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

12.3. HEADERS

Name	Type	値	説明
CamelAtomixResourceAction	AtomixMap.Action	• PUT • PUT_IF_ABSENT • GET • CLEAR • SIZE • CONTAINS_KEY • CONTAINS_VALUE • IS_EMPTY • ENTRY_SET • REMOVE • REPLACE • 値	実行するアクション
CamelAtomixResourceKey	オブジェクト	-	操作に使用するキー
CamelAtomixResourceValue	オブジェクト	-	In Body がない場合に値が使用されます。
CamelAtomixResourceOldValue	オブジェクト	-	古い値
CamelAtomixResourceTTL	string / long	-	エントリー TTL

Name	Type	値	説明
Camel Atomix ResourceReadConsistency	ReadConsistency	• ATOMIC • ATOMIC_LEASE • SEQUENTIAL • LOCAL	一貫性レベルの読み取り

12.4. ATOMIX クラスターに接続するためのコンポーネントの設定

参加する Atomix クラスターのノードは、エンドポイントまたはコンポーネントレベル（推奨）で区切ります（推奨）。以下に例を示します。

- エンドポイント：

```
<beans xmlns="...">
    <camelContext xmlns="http://camel.apache.org/schema/spring">
        <from uri="direct:start"/>
            <to uri="atomix-map:myMap?nodes=node-1.atomix.cluster:8700,node-2.atomix.cluster:8700"/>
        </route>
    </camelContext>
</beans>
```

- コンポーネント：

```
<beans xmlns="...">
    <bean id="atomix-map" class="org.apache.camel.component.atomix.client.map.AtomixMapComponent">
        <property name="nodes" value="nodes=node-1.atomix.cluster:8700,node-2.atomix.cluster:8700"/>
    </bean>

    <camelContext xmlns="http://camel.apache.org/schema/spring">
        <from uri="direct:start"/>
            <to uri="atomix-map:myMap"/>
        </route>
    </camelContext>
</beans>
```

12.5. 使用例：

- 要素に 1 秒の TTL を設定します。

```
FluentProducerTemplate.on(context)
    .withHeader(AtomixClientConstants.RESOURCE_ACTION, AtomixMap.Action.PUT)
    .withHeader(AtomixClientConstants.RESOURCE_KEY, key)
    .withHeader(AtomixClientConstants.RESOURCE_TTL, "1s")
    .withBody(val)
    .to("direct:start")
    .send();
```

第13章 ATOMIX MESSAGING COMPONENT

Camel バージョン 2.20 で利用可能

camel atomix-messaging コンポーネントを使用すると、Atomix の Group Messaging で作業できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

13.1. URI 形式

```
atomix-messaging:group
```

Atomix Messaging コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有コンポーネントの設定		AtomixMessaging Configuration
atomix (common)	共有 AtomixClient インスタンス		AtomixClient
ノード (common)	AtomixClient が接続するノード		リスト
configurationUri (common)	AtomixClient 設定へのパス		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix Messaging エンドポイントは、URI 構文を使用して設定します。

-

```
atomix-messaging:resourceName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

13.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceName	**必須**。分散リソース名		文字列

13.1.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
atomix (common)	使用する Atomix インスタンス		Atomix
broadcastType (common)	ブロードキャストタイプ。	ALL	BroadcastType
channelName (common)	メッセージングチャネル名		文字列
configurationUri (common)	Atomix 設定 URI。		文字列
defaultAction (common)	デフォルトの動作。	DIRECT	アクション
memberName (common)	Atomix Group member name		文字列
ノード (common)	クラスターを構成するノードのアドレス。		文字列
resultHeader (common)	悪いヘッダーには結果が含まれます。		文字列
トランスポート（共通）	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトリソース設定。		プロパティー
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		プロパティー
ephemeral （詳細）	ローカルメンバーがグループを PersistentMember として参加させるかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取ると、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	一貫性レベルの読み取り。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソースの設定。		マップ
resourceOptions (advanced)	ローカルリソースの設定		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

第14章 ATOMIX MULTIMAP COMPONENT

Camel バージョン 2.20 で利用可能

camel atomix-multimap コンポーネントを使用すると、Atomix の Distributed MultiMap コレクションと連携できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

14.1. URI 形式

```
atomix-multimap:multiMapName
```

Atomix MultiMap コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （コンシューマー）	共有コンポーネントの設定		AtomixMultiMapConfiguration
atomix (consumer)	共有 AtomixClient インスタンス		AtomixClient
ノード （コンシューマー）	AtomixClient が接続するノード		リスト
configurationUri (consumer)	AtomixClient 設定へのパス		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix MultiMap エンドポイントは、URI 構文を使用して設定します。

```
atomix-multimap:resourceName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

14.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceName	**必須**。分散リソース名		文字列

14.1.2. クエリーパラメーター（18 パラメーター）：

Name	説明	デフォルト	Type
atomix (consumer)	使用する Atomix インスタンス		Atomix
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
configurationUri (consumer)	Atomix 設定 URI。		文字列
defaultAction (consumer)	デフォルトの動作。	PUT	アクション
キー（コンシューマー）	ヘッダーに何も設定されていない場合や、特定のキーのイベントをリッスンする場合に使用するキー。		オブジェクト
ノード（コンシューマー）	クラスターを構成するノードのアドレス。		文字列

Name	説明	デフォルト	Type
resultHeader (consumer)	悪いヘッダーには結果が含まれます。		文字列
Transport (consumer)	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
TTL （コンシューマー）	リソーススロット。		Long
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトリソース設定。		プロパティー
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		プロパティー
ephemeral （詳細）	ローカルメンバーがグループを PersistentMember として参加させるかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取ると、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	一貫性レベルの読み取り。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソースの設定。		マップ
resourceOptions (advanced)	ローカルリソースの設定		マップ

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

第15章 ATOMIX QUEUE COMPONENT

Camel バージョン 2.20 で利用可能

camel atomix-queue コンポーネントを使用すると、Atomix の Distributed Queue コレクションと連携できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

15.1. URI 形式

```
atomix-queue:queueName
```

Atomix Queue コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有コンポーネントの設定		AtomixQueue Configuration
atomix (common)	共有 AtomixClient インスタンス		AtomixClient
ノード (common)	AtomixClient が接続するノード		リスト
configurationUri (common)	AtomixClient 設定へのパス		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix Queue エンドポイントは、URI 構文を使用して設定します。

-

```
atomix-queue:resourceName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

15.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceName	**必須**。分散リソース名		文字列

15.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
atomix (common)	使用する Atomix インスタンス		Atomix
configurationUri (common)	Atomix 設定 URI。		文字列
defaultAction (common)	デフォルトの動作。	追加	アクション
ノード (common)	クラスターを構成するノードのアドレス。		文字列
resultHeader (common)	悪いヘッダーには結果が含まれます。		文字列
トランスポート（共通）	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトリソース設定。		プロパティー
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		プロパティー
ephemeral （詳細）	ローカルメンバーがグループを PersistentMember として参加させるかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取ると、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	一貫性レベルの読み取り。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソースの設定。		マップ
resourceOptions (advanced)	ローカルリソースの設定		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

第16章 ATOMIX SET COMPONENT

Camel バージョン 2.20 で利用可能

camel atomix-set コンポーネントを使用すると、Atomix の Distributed Set コレクションと連携できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

16.1. URI 形式

```
atomix-set:setName
```

Atomix Set コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有コンポーネントの設定		AtomixSetConfiguration
atomix (common)	共有 AtomixClient インスタンス		AtomixClient
ノード (common)	AtomixClient が接続するノード		リスト
configurationUri (common)	AtomixClient 設定へのパス		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix Set エンドポイントは、URI 構文を使用して設定します。

-

```
atomix-set:resourceName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

16.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceName	**必須**。分散リソース名		文字列

16.1.2. クエリーパラメーター（17 パラメーター）：

Name	説明	デフォルト	Type
atomix (common)	使用する Atomix インスタンス		Atomix
configurationUri (common)	Atomix 設定 URI。		文字列
defaultAction (common)	デフォルトの動作。	追加	アクション
ノード (common)	クラスターを構成するノードのアドレス。		文字列
resultHeader (common)	悪いヘッダーには結果が含まれます。		文字列
トランスポート（共通）	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
TTL (common)	リソーススロット。		Long

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトリソース設定。		プロパティー
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		プロパティー
ephemeral （詳細）	ローカルメンバーがグループを PersistentMember として参加させるかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取ると、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	一貫性レベルの読み取り。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソースの設定。		マップ
resourceOptions (advanced)	ローカルリソースの設定		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

第17章 ATOMIX VALUE COMPONENT

Camel バージョン 2.20 で利用可能

camel atomix-value コンポーネントを使用すると、Atomix の Distributed Value と連携できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-atomix</artifactId>
  <version>${camel-version}</version>
</dependency>
```

17.1. URI 形式

```
atomix-value:valueName
```

Atomix Value コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有コンポーネントの設定		AtomixValue Configuration
atomix (common)	共有 AtomixClient インスタンス		AtomixClient
ノード (common)	AtomixClient が接続するノード		リスト
configurationUri (common)	AtomixClient 設定へのパス		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Atomix Value エンドポイントは、URI 構文を使用して設定します。

```
atomix-value:resourceName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

17.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceName	**必須**。分散リソース名		文字列

17.1.2. クエリーパラメーター（17 パラメーター）：

Name	説明	デフォルト	Type
atomix (common)	使用する Atomix インスタンス		Atomix
configurationUri (common)	Atomix 設定 URI。		文字列
defaultAction (common)	デフォルトの動作。	SET	アクション
ノード (common)	クラスターを構成するノードのアドレス。		文字列
resultHeader (common)	悪いヘッダーには結果が含まれます。		文字列
トランスポート（共通）	Atomix トランスポートを設定します。	io.atomix.catalyst.transport.netty.NettyTransport	トランスポート
TTL (common)	リソーススロット。		Long

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
defaultResourceConfig (advanced)	クラスター全体のデフォルトリソース設定。		プロパティー
defaultResourceOptions (advanced)	ローカルのデフォルトリソースオプション。		プロパティー
ephemeral （詳細）	ローカルメンバーがグループを PersistentMember として参加させるかどうかを設定します。ephemeral に設定すると、ローカルメンバーは自動生成された ID を受け取ると、ローカルメンバーは無視されます。	false	boolean
readConsistency (advanced)	一貫性レベルの読み取り。		ReadConsistency
resourceConfigs (advanced)	クラスター全体のリソースの設定。		マップ
resourceOptions (advanced)	ローカルリソースの設定		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

第18章 AVRO コンポーネント

Camel バージョン 2.10 で利用可能

このコンポーネントは、avro のデータ形式を提供します。これにより、Apache Avro のバイナリーデータ形式を使用してメッセージのシリアライズおよびデシリアライズが可能になります。さらに、netty または http を介して avro を使用するためのプロデューサーおよびコンシューマーエンドポイントを提供することで、Apache Avro の rpc に対応します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-avro</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

18.1. APACHE AVRO の概要

Avro では、フォーマットなどの json を使用してメッセージタイプとプロトコルを定義し、指定された型およびメッセージの java コードを生成します。スキーマの挙動の例を以下に示します。

```
{"namespace": "org.apache.camel.avro.generated",
 "protocol": "KeyValueProtocol",

 "types": [
    {"name": "Key", "type": "record",
     "fields": [
        {"name": "key",   "type": "string"}
     ]
    },
    {"name": "Value", "type": "record",
     "fields": [
        {"name": "value",   "type": "string"}
     ]
    }
 ],

 "messages": {
    "put": {
       "request": [{"name": "key", "type": "Key"}, {"name": "value", "type": "Value"} ],
       "response": "null"
    },
    "get": {
       "request": [{"name": "key", "type": "Key"}],
```

```
        "response": "Value"
      }
    }
  }
```

Maven やant などを使用して、スキーマからクラスを簡単に生成できます。詳細は Apache Avro のドキュメント を参照してください。

ただし、スキーマの最初のアプローチは実施せず、既存のクラスのスキーマを作成できます。2.12 以降、既存のプロトコルインターフェースを使用して RCP 呼び出しを行うことができます。パラメーターおよび結果タイプには、プロトコル自体と POJO Bean またはプリミティブ/文字列クラスにインターフェースを使用する必要があります。以下は、上記のスキーマに対応するクラスの例です。

```
package org.apache.camel.avro.reflection;

public interface KeyValueProtocol {
    void put(String key, Value value);
    Value get(String key);
}

class Value {
    private String value;
    public String getValue() { return value; }
    public void setValue(String value) { this.value = value; }
}
```

注記： 既存のクラスは RPC（以下を参照）にのみ使用でき、データ形式では使用できません。

18.2. AVRO データフォーマットの使用

avro データフォーマットを使用することは、ルートでマーシャリングまたはアンマーシャリングするクラスを指定するのと同じくらい簡単です。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:in"/>
        <marshal>
            <avro instanceClass="org.apache.camel.dataformat.avro.Message"/>
        </marshal>
        <to uri="log:out"/>
    </route>
</camelContext>
```

または、コンテキスト内でデータ形式を指定し、ルートから参照することもできます。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <dataFormats>
      <avro id="avro" instanceClass="org.apache.camel.dataformat.avro.Message"/>
    </dataFormats>
    <route>
      <from uri="direct:in"/>
      <marshal ref="avro"/>
      <to uri="log:out"/>
    </route>
</camelContext>
```

同様に、avro データフォーマットを使用してマージできます。

18.3. CAMEL での AVRO RPC の使用

前述のように、Avro は、http や netty などの複数のトランスポートで RPC サポートを提供します。Camel は、これらの 2 つのトランスポートにコンシューマーとプロデューサーを提供します。

```
avro:[transport]:[host]:[port][?options]
```

現在、サポートされるトランスポートの値は http または netty です。

2.12 以降、URI でメッセージ名を右側に指定できます。

```
avro:[transport]:[host]:[port][/messageName][?options]
```

コンシューマーの場合、複数のルートを同じソケットに割り当てることができます。適切なルートへのディスパッチは、自動的に avro コンポーネントにより行われます。messageName の指定のないルート（ある場合）がデフォルトとして使用されます。

avro ipc に camel プロデューサーを使用する場合、「in」メッセージボディーには avro プロトコルで指定された操作のパラメーターが含まれている必要があります。応答は「out」メッセージのボディーに追加されます。

camel avro コンシューマーを avro ipc に使用する場合と同様に、リクエストパラメーターは作成されたエクスチェンジの「in」メッセージボディー内に配置され、エクスチェンジの処理後、"out" メッセージのボディーが応答として送信されます。

注記： デフォルトでは、コンシューマーパラメーターは配列にラップされます。パラメーターが 1 つしかない場合は、2.12 以降、singleParameter URI オプションを使用して、配列ラッピングなしで

「in」メッセージボディーでこれを受け取ることができます。

18.4. AVRO RPC URI オプション

Avro コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有 AvroConfiguration を使用してオプションを一度に設定するには、以下を実行します。		AvroConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Avro エンドポイントは URI 構文を使用して設定します。

```
avro:transport:host:port/messageName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

18.4.1. パスパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
transport	使用する **必要な** トランスポート		AvroTransport
port	使用する **必要な** ポート番号		int
host	**使用** するホスト名		文字列
messageName	送信するメッセージの名前。		文字列

18.4.2. クエリーパラメーター（10 パラメーター）：

Name	説明	デフォルト	Type
プロトコル (common)	使用する Avro プロトコル		プロトコル
protocolClassName (common)	FQN クラス名で定義される Avro プロトコル		文字列
protocolLocation (common)	Avro プロトコルの場所		文字列
reflectionProtocol (common)	指定されたプロトコルオブジェクトがリフレクションプロトコルである場合。protocolClassName プロトコルタイプが自動検出されるため、protocol パラメーターとのみ使用する必要があります。	false	boolean
singleParameter (common)	true の場合、consumer パラメーターはアレイにラップされません。プロトコルがメッセージに対して 1 パラメーターを指定すると失敗します。	false	boolean
uriAuthority (common)	使用する認証局（ユーザー名とパスワード）		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

18.5. AVRO RPC ヘッダー

Name	説明
CamelAvroMessageName	送信するメッセージの名前。コンシューマーで URI からのメッセージ名を上書きする場合（存在する場合）

18.6. 例

http で camel avro プロデューサーを使用する例：

```
<route>
    <from uri="direct:start"/>
    <to uri="avro:http:localhost:{{avroport}}?
protocolClassName=org.apache.camel.avro.generated.KeyValueProtocol"/>
    <to uri="log:avro"/>
</route>
```

上記の例では、CamelAvroMessageName ヘッダーを入力する必要があります。2.12 以降、以下の構文を使用して定数メッセージを呼び出すことができます。

```
<route>
    <from uri="direct:start"/>
    <to uri="avro:http:localhost:{{avroport}}/put?
protocolClassName=org.apache.camel.avro.generated.KeyValueProtocol"/>
    <to uri="log:avro"/>
</route>
```

netty 経由で camel avro コンシューマーを使用してメッセージを使用する例：

```
<route>
    <from uri="avro:netty:localhost:{{avroport}}?
protocolClassName=org.apache.camel.avro.generated.KeyValueProtocol"/>
    <choice>
      <when>
        <el>${in.headers.CamelAvroMessageName == 'put'}</el>
        <process ref="putProcessor"/>
      </when>
      <when>
        <el>${in.headers.CamelAvroMessageName == 'get'}</el>
        <process ref="getProcessor"/>
      </when>
    </choice>
</route>
```

2 .12 以降、同じタスクを実行するため、2 つの異なるルートを設定できます。

```
        <route>
            <from uri="avro:netty:localhost:{{avroport}}/put?
protocolClassName=org.apache.camel.avro.generated.KeyValueProtocol">
            <process ref="putProcessor"/>
        </route>
        <route>
            <from uri="avro:netty:localhost:{{avroport}}/get?
protocolClassName=org.apache.camel.avro.generated.KeyValueProtocol&singleParameter=true"/>
            <process ref="getProcessor"/>
        </route>
```

上記の例では、get は 1 つのパラメーターのみを取り、single Processor は Value クラスをボディーで直接受信します。一方、putProcessor は、String キーと Value 値が配列コンテンツとして設定されるサイズ 2 の配列を受け取ります。

第19章 AVRO DATAFORMAT

Camel バージョン 2.14 から利用可能

このコンポーネントは、avro のデータ形式を提供します。これにより、Apache Avro のバイナリーデータ形式を使用してメッセージのシリアライズおよびデシリアライズが可能になります。さらに、netty または http を介して avro を使用するためのプロデューサーおよびコンシューマーエンドポイントを提供することで、Apache Avro の rpc に対応します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-avro</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

19.1. APACHE AVRO の概要

Avro では、フォーマットなどの json を使用してメッセージタイプとプロトコルを定義し、指定された型およびメッセージの java コードを生成します。スキーマの挙動の例を以下に示します。

```
{"namespace": "org.apache.camel.avro.generated",
 "protocol": "KeyValueProtocol",

 "types": [
    {"name": "Key", "type": "record",
     "fields": [
        {"name": "key",   "type": "string"}
     ]
    },
    {"name": "Value", "type": "record",
     "fields": [
        {"name": "value",   "type": "string"}
     ]
    }
 ],

 "messages": {
    "put": {
       "request": [{"name": "key", "type": "Key"}, {"name": "value", "type": "Value"} ],
       "response": "null"
    },
    "get": {
       "request": [{"name": "key", "type": "Key"}],
```

```
        "response": "Value"
    }
  }
}
```

Maven やant などを使用して、スキーマからクラスを簡単に生成できます。詳細は Apache Avro のドキュメント を参照してください。

ただし、スキーマの最初のアプローチは実施せず、既存のクラスのスキーマを作成できます。2.12 以降、既存のプロトコルインターフェースを使用して RCP 呼び出しを行うことができます。パラメーターおよび結果タイプには、プロトコル自体と POJO Bean またはプリミティブ/文字列クラスにインターフェースを使用する必要があります。以下は、上記のスキーマに対応するクラスの例です。

```
package org.apache.camel.avro.reflection;

public interface KeyValueProtocol {
    void put(String key, Value value);
    Value get(String key);
}

class Value {
    private String value;
    public String getValue() { return value; }
    public void setValue(String value) { this.value = value; }
}
```

注記： 既存のクラスは RPC（以下を参照）にのみ使用でき、データ形式では使用できません。

19.2. AVRO データフォーマットの使用

avro データフォーマットを使用することは、ルートでマーシャリングまたはアンマーシャリングするクラスを指定するのと同じくらい簡単です。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:in"/>
        <marshal>
            <avro instanceClass="org.apache.camel.dataformat.avro.Message"/>
        </marshal>
        <to uri="log:out"/>
    </route>
</camelContext>
```

または、コンテキスト内でデータ形式を指定し、ルートから参照することもできます。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <dataFormats>
      <avro id="avro" instanceClass="org.apache.camel.dataformat.avro.Message"/>
    </dataFormats>
    <route>
      <from uri="direct:in"/>
      <marshal ref="avro"/>
      <to uri="log:out"/>
    </route>
</camelContext>
```

同様に、avro データフォーマットを使用してマージできます。

19.3. AVRO データフォーマットのオプション

Avro データフォーマットは、以下に示す 2 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
instanceClassName		**文字列**	マーシャリングとアンマーシャリングに使用するクラス名
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

第20章 AWS CLOUDWATCH COMPONENT

Camel バージョン 2.11 で利用可能

CW コンポーネントを使用すると、メッセージを Amazon CloudWatch メトリクスに送信できます。Amazon API の実装は AWS SDK によって提供されます。

前提条件

Amazon CloudWatch を使用するように有効な Amazon Web Services 開発者アカウントが必要で、サインアップする必要があります。詳細は、Amazon CloudWatch を参照してください。

20.1. URI 形式

```
aws-cw://namespace[?options]
```

メトリクスが存在しない場合は作成されます。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

20.2. URI オプション

AWS CloudWatch コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS CW のデフォルト設定		CwConfiguration
accessKey (producer)	Amazon AWS Access Key		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列
リージョン（プロデューサー）	CW クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS CloudWatch エンドポイントは、URI 構文を使用して設定します。

```
aws-cw:namespace
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

20.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
namespace	***必須***: メトリクス namespace		文字列

20.2.2. クエリーパラメーター（11 パラメーター）：

Name	説明	デフォルト	Type
amazonCwClient (producer)	AmazonCloudWatch をクライアントとして使用します。		AmazonCloudWatch
名前 （プロデューサー）	メトリクス名		文字列
proxyHost (producer)	CW クライアントをインスタンス化する際にプロキシーホストを定義します。		文字列
proxyPort (producer)	CW クライアントをインスタンス化する際にプロキシーポートを定義します。		整数
リージョン （プロデューサー）	CW クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
タイムスタンプ（プロデューサー）	メトリクスのタイムスタンプ		Date
ユニット（プロデューサー）	メトリックユニット		文字列
値（プロデューサー）	メトリックの値		double
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
accessKey (security)	Amazon AWS Access Key		文字列
secretKey (security)	Amazon AWS Secret Key		文字列

必要な CW コンポーネントオプション

Amazon の CloudWatch にアクセスするためにレジストリーまたは accessKey および secretKey で amazonCwClient を提供する必要があります。

20.3. 用途

20.3.1. CW プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
CamelAwsCwMetricName	**文字列**	Amazon CW メトリクス名。

ヘッダー	Type	説明
CamelAwsCwMetricValue	**double**	Amazon CW メトリクスの値。
CamelAwsCwMetricUnit	**文字列**	Amazon CW メトリクスユニット。
CamelAwsCwMetricNamespace	**文字列**	Amazon CW メトリクス namespace。
CamelAwsCwMetricTimestamp	**Date**	Amazon CW メトリクスのタイムスタンプ。
CamelAwsCwMetricDimensionName	**文字列**	**Camel 2.12:** Amazon CW メトリクスのディメンション名。
CamelAwsCwMetricDimensionValue	**文字列**	**Camel 2.12:** Amazon CW メトリクスのディメンション値。
CamelAwsCwMetricDimensions	**Map<String, String>**	**Camel 2.12:** ディメンション名とディメンション値のマップ。

20.3.2. Advanced AmazonCloudWatch configuration

AmazonCloudWatch インスタンス設定に対する制御を強化する必要がある場合は、独自のインスタンスを作成して URI から参照することができます。

```
from("direct:start")
.to("aws-cw://namepsace?amazonCwClient=#client");
```

#client はレジストリーの AmazonCloudWatch を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合は、以下のようになります。

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey",
"mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonCloudWatch client = new AmazonCloudWatchClient(awsCredentials,
clientConfiguration);

registry.bind("client", client);
```

20.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.10 以降）に置き換える必要があります。

20.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- AWS コンポーネント

第21章 AWS DYNAMODB COMPONENT

Camel バージョン 2.10 で利用可能

DynamoDB コンポーネントは、Amazon の DynamoDB サービスからのデータの保存および取得をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon DynamoDB を使用するためにサインアップする必要があります。詳細は、Amazon DynamoDB を参照してください。

21.1. URI 形式

```
aws-ddb://domainName[?options]
```

URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

21.2. URI オプション

AWS DynamoDB コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS DDB のデフォルト設定		DdbConfiguration
accessKey (producer)	Amazon AWS Access Key		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列
リージョン（プロデューサー）	DDB クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS DynamoDB エンドポイントは、URI 構文を使用して設定します。

```
aws-ddb:tableName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

21.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
tableName	**必須**。現在作業中のテーブルの名前が必要です。		文字列

21.2.2. クエリーパラメーター（13 パラメーター）：

Name	説明	デフォルト	Type
amazonDDBClient (producer)	AmazonDynamoDB をクライアントとして使用します。		AmazonDynamoDB
consistentRead (producer)	データの読み取り時に強力な整合性を適用するべきかどうかを決定します。	false	boolean
keyAttributeName (producer)	テーブル作成時の属性名		文字列
keyAttributeType (producer)	テーブル作成時の属性型		文字列
操作 （プロデューサー）	実行する操作	PutItem	DdbOperations
proxyHost (producer)	DDB クライアントをインスタンス化する際にプロキシーホストを定義します。		文字列

Name	説明	デフォルト	Type
proxyPort (producer)	DDB クライアントをインスタンス化する際にプロキシーポートを定義します。		整数
readCapacity (producer)	テーブルからリソースを読み取るために確保するプロビジョニングされたスループット		Long
リージョン （プロデューサー）	DDB クライアントが機能する必要があるリージョン		文字列
writeCapacity (producer)	テーブルのリソースを書き込むために確保するプロビジョニングされたスループット		Long
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
accessKey (security)	Amazon AWS Access Key		文字列
secretKey (security)	Amazon AWS Secret Key		文字列

必要な DDB コンポーネントオプション

Amazon の DynamoDB にアクセスするために、レジストリーまたは accessKey および secretKey で amazonDDBClient を提供する必要があります。

21.3. 用途

21.3.1. DDB プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
CamelAwsDdbBatchItems	**Map<String, KeysAndAttributes>**	プライマリーキーが取得するテーブル名と対応する項目のマッピング。

ヘッダー	Type	説明
CamelAwsDdbTableName	**文字列**	この操作の表名。
CamelAwsDdbKey	**キー**	テーブル内の各項目を一意に識別するプライマリーキー。**Camel 2.16.0 より、このヘッダーのタイプは Map<String, AttributeValue> であり、Key ではなく Map<String, AttributeValue> になります。**
CamelAwsDdbReturnValues	**文字列**	変更する前または後に属性名と値のペアを取得する場合は、このパラメーターを使用します（NONE、ALL_OLD、UPDATED_OLD、ALL_NEW、UPDATE、UPDATE）。
CamelAwsDdbUpdateCondition	**Map<String, ExpectedAttributeValue>**	条件変更の属性を指定します。
CamelAwsDdbAttributeNames	**collection<String>**	属性名が指定されていない場合、すべての属性が返されます。
CamelAwsDdbConsistentRead	**ブール値**	true に設定すると、一貫した読み取りが発行され、それ以外は最終的に一貫したものが使用されます。
CamelAwsDdbIndexName	**文字列**	設定されている場合は、Query 操作のセカンダリーインデックスとして使用されます。

ヘッダー	Type	説明
CamelAwsDdbItem	**Map<String, AttributeValue>**	アイテムの属性のマップ。アイテムを定義するプライマリーキー値を含める必要があります。
CamelAwsDdbExactCount	**ブール値**	true に設定すると、Amazon DynamoDB は一致するアイテムとその属性のリストではなく、クエリーパラメーターに一致するアイテムの合計数を返します。**Camel 2.16.0 より、このヘッダーは存在しなくなりました。**
CamelAwsDdbKeyConditions	**Map<String, Condition>**	From Camel 2.16.0.このヘッダーはクエリーの選択基準を指定し、2 つの古いヘッダー CamelAwsDdbHashKeyValue および CamelAwsDdbScanRangeKeyConditionをマージします。
CamelAwsDdbStartKey	**キー**	以前のクエリーを継続する項目のプライマリーキー。
CamelAwsDdbHashKeyValue	**AttributeValue**	複合プライマリーキーのハッシュコンポーネントの値。**Camel 2.16.0 より、このヘッダーは存在しなくなりました。**
CamelAwsDdbLimit	**整数**	返すアイテムの最大数。
CamelAwsDdbScanRangeKeyCondition	**状態**	クエリーに使用する属性値と比較演算子のコンテナー。**Camel 2.16.0 では、このヘッダーは存在しなくなりました。**

ヘッダー	Type	説明
CamelAwsDdbScanIndexForward	**ブール値**	インデックスの forward または Backward トラバーサルを指定します。
CamelAwsDdbScanFilter	**Map<String, Condition>**	スキャン結果を評価し、必要な値のみを返します。
CamelAwsDdbUpdateValues	**Map<String, AttributeValueUpdate>**	更新の属性名の新しい値およびアクションへのマッピング。

21.3.2. BatchGetItems 操作中に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsDdbBatchResponse	**Map<String,BatchResponse>**	テーブル名およびテーブルの各項目属性。
CamelAwsDdbUnprocessedKeys	**Map<String,KeysAndAttributes>**	テーブルのマッピングと、現在の応答で処理されなかった対応するキーが含まれます。

21.3.3. DeleteItem 操作時に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	操作によって返される属性の一覧です。

21.3.4. DeleteTable 操作時に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsDdbProvisionedThroughput		
ProvisionedThroughputDescription		このテーブルの ProvisionedThroughput プロパティーの値
CamelAwsDdbCreationDate	**Date**	このテーブルの DateTime の作成。
CamelAwsDdbTableItemCount	**Long**	このテーブルのアイテム数。
CamelAwsDdbKeySchema	**KeySchema**	このテーブルのプライマリーキーを識別する KeySchema。**Camel 2.16.0 より、このヘッダーのタイプは List<KeySchemaElement> で KeySchema ではありません。**

ヘッダー	Type	説明
CamelAwsDdbTableName	**文字列**	テーブル名。
CamelAwsDdbTableSize	**Long**	テーブルサイズ（バイト単位）。
CamelAwsDdbTableStatus	**文字列**	テーブルのステータス： CREATING、UPDATING、DELETING、ACTIVE

21.3.5. DescribeTable 操作中に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsDdbProvisionedThroughput	{{ProvisionedThroughputDescription}}	このテーブルの ProvisionedThroughput プロパティーの値
CamelAwsDdbCreationDate	**Date**	このテーブルの DateTime の作成。
CamelAwsDdbTableItemCount	**Long**	このテーブルのアイテム数。
CamelAwsDdbKeySchema	{{KeySchema}}	このテーブルのプライマリーキーを識別する KeySchema。**Camel 2.16.0 より、このヘッダーのタイプは List<KeySchemaElement> で KeySchema ではありません。**

ヘッダー	Type	説明
CamelAwsDdbTableName	**文字列**	テーブル名。
CamelAwsDdbTableSize	**Long**	テーブルサイズ（バイト単位）。
CamelAwsDdbTableStatus	**文字列**	テーブルのステータス： CREATING、UPDATING、DELETING、ACTIVE
CamelAwsDdbReadCapacity	**Long**	このテーブルの ReadCapacityUnits プロパティー。
CamelAwsDdbWriteCapacity	**Long**	このテーブルの WriteCapacityUnits プロパティー。

21.3.6. GetItem 操作時に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	操作によって返される属性の一覧です。

21.3.7. PutItem 操作中に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	操作によって返される属性の一覧です。

21.3.8. Query 操作時に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsDdbItems	**List<java.util.Map<String,AttributeValue>>**	操作によって返される属性の一覧です。
CamelAwsDdbLastEvaluatedKey	**キー**	クエリー操作が停止する項目の主なキー。以前の結果セットが含まれます。
CamelAwsDdbConsumedCapacity	**double**	操作中に消費されるテーブルのプロビジョニングされたスループットの容量単位数。
CamelAwsDdbCount	**整数**	応答のアイテム数。

21.3.9. スキャン操作時に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsDdbItems	**List<java.util.Map<String,AttributeValue>>**	操作によって返される属性の一覧です。
CamelAwsDdbLastEvaluatedKey	**キー**	クエリー操作が停止する項目の主なキー。以前の結果セットが含まれます。
CamelAwsDdbConsumedCapacity	**double**	操作中に消費されるテーブルのプロビジョニングされたスループットの容量単位数。
CamelAwsDdbCount	**整数**	応答のアイテム数。
CamelAwsDdbScannedCount	**整数**	フィルターの適用前の完全なスキャンの項目数。

21.3.10. UpdateItem 操作中に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	操作によって返される属性の一覧です。

21.3.11. 高度な AmazonDynamoDB 設定

AmazonDynamoDB インスタンス設定に対する制御を強化する必要がある場合は、独自のインスタンスを作成して URI から参照することができます。

```
from("direct:start")
.to("aws-ddb://domainName?amazonDDBClient=#client");
```

#client はレジストリーの AmazonDynamoDB を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合は、以下のようになります。

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey",
"mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonDynamoDB client = new AmazonDynamoDBClient(awsCredentials,
clientConfiguration);

registry.bind("client", client);
```

21.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws</artifactId>
    <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.10 以降）に置き換える必要があります。

21.5. 関連項目

- **Configuring Camel (Camel の設定)**
- **コンポーネント**
- **エンドポイント**
- **はじめに**
- **AWS コンポーネント**

第22章 AWS DYNAMODB STREAMS COMPONENT

Camel バージョン 2.17 から利用可能

DynamoDB Stream コンポーネントは、Amazon DynamoDB Stream サービスからのメッセージの受信をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon DynamoDB Streams を使用するためにサインアップする必要があります。詳細は、 AWS DynamoDBを参照してください。

22.1. URI 形式

```
aws-ddbstream://table-name[?options]
```

ストリームの使用前にストリームを作成する必要があります。
URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

22.2. URI オプション

AWS DynamoDB Streams コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS DDB ストリームのデフォルト設定		DdbStreamConfiguration
accessKey (consumer)	Amazon AWS Access Key		文字列
secretKey (consumer)	Amazon AWS Secret Key		文字列
リージョン（コンシューマー）	Amazon AWS リージョン		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS DynamoDB Streams エンドポイントは、URI 構文を使用して設定されます。

```
aws-ddbstream:tableName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

22.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
tableName	dynamodb テーブルに **必要な** 名前		文字列

22.2.2. クエリーパラメーター（28 パラメーター）：

Name	説明	デフォルト	Type
amazonDynamoDbStreams Client (consumer)	このエンドポイントのすべてのリクエストに使用する Amazon DynamoDB クライアント		AmazonDynamoDBStreams
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
iteratorType (consumer)	DynaboDB ストリームでレコードの取得を開始する場所を定義します。TRIM_HORIZON を使用すると、ストリームがリアルタイムに追いつく前にかなりの遅延が発生する可能性があることに注意してください。AT,AFTER_SEQUENCE_NUMBER を使用する場合は、sequenceNumberProvider を指定する必要があります。	最新バージョン	ShardIteratorType
maxResultsPerRequest (consumer)	各ポーリングでフェッチされる最大レコード数。		int
proxyHost (consumer)	DDBStreams クライアントをインスタンス化するときにプロキシーホストを定義します。		文字列
proxyPort (consumer)	DDBStreams クライアントをインスタンス化する際にプロキシーポートを定義します。		整数
リージョン （コンシューマー）	DDBStreams クライアントが機能する必要があるリージョン		文字列
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
sequenceNumberProvider (consumer)	2 つの ShardIteratorType.AT,AFTER_SEQUENCE_NUMBER イテレータータイプのいずれかを使用する場合のシーケンス番号のプロバイダー。レジストリーの参照またはリテラルシーケンス番号を指定できます。		SequenceNumberProvider
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler

Name	説明	デフォルト	Type
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
accessKey (security)	Amazon AWS Access Key		文字列
secretKey (security)	Amazon AWS Secret Key		文字列

必要な DynampDBStream コンポーネントオプション

プロキシーおよび関連するクレデンシャルが設定されたレジストリーに amazonDynamoDbStreamsClient を提供する必要があります。

22.3. シーケンス番号

リテラル文字列をシーケンス番号として提供したり、レジストリーに Bean を提供できます。Bean の使用例は、変更フィードに現在の場所を保存し、Camel の起動時に復元することです。

これは、AWS 呼び出しが HTTP 400 を返すため、describe-streams の結果の最大シーケンス番号よりも大きいシーケンス番号を提供するのはエラーです。

22.4. バッチコンシューマー

このコンポーネントは Batch Consumer を実装します。

これにより、たとえば、このバッチに存在するメッセージ数を把握し、Aggregator にこの数のメッセージを集約することもできます。

22.5. 用途

22.5.1. AmazonDynamoDBStreamsClient configuration

AmazonDynamoDBStreamsClient のインスタンスを作成して、レジストリーにバインドする必要があります。

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

Region region = Region.getRegion(Regions.fromName(region));
region.createClient(AmazonDynamoDBStreamsClient.class, null, clientConfiguration);
// the 'null' here is the AWSCredentialsProvider which defaults to an instance of
DefaultAWSCredentialsProviderChain

registry.bind("kinesisClient", client);
```

22.5.2. AWS 認証情報の指定

新しい ClientConfiguration インスタンスの作成時にデフォルトである DefaultAWSCredentialsProviderChain を使用して認証情報を取得することが推奨されますが、createClient(...)を呼び出す際に異なる AWSCredentialsProvider を指定することもできます。

22.6. DOWNTIME でのスコープ設定

22.6.1. AWS DynamoDB Streams の停止時間は 24 時間未満

コンシューマーは最後に確認されたシーケンス番号（ CAMEL-9515に実装されたもの）から再開します。したがって、停止にも DynamoDB 自体が含まれていない限り、迅速にイベントを受け取る必要があります。

22.6.2. AWS DynamoDB Streams の 24 時間以上停止

AWS が 24 時間以上変更を保持することを前提とすると、どのような軽減策が発生しても変更イベントが見逃されます。

22.7. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.7 以降）に置き換える必要があります。

22.8. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- AWS Component
  +

第23章 AWS EC2 コンポーネント

Camel バージョン 2.16 から利用可能

EC2 コンポーネントは、AWS EC2 インスタンスの作成、実行、起動、停止、および終了をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon EC2 を使用するにはサインアップする必要があります。詳細は、Amazon EC2 を参照してください。

23.1. URI 形式

```
aws-ec2://label[?options]
```

URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

23.2. URI オプション

AWS EC2 コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS EC2 のデフォルト設定		EC2Configuration
リージョン（プロデューサー）	EC2 クライアントが機能する必要があるリージョン		文字列
accessKey (producer)	Amazon AWS Access Key		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS EC2 エンドポイントは URI 構文を使用して設定されます。

```
aws-ec2:label
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

23.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
ラベル	**必要な** 論理名		文字列

23.2.2. クエリーパラメーター（8 パラメーター）：

Name	説明	デフォルト	Type
accessKey (producer)	Amazon AWS Access Key		文字列
amazonEc2Client (producer)	既存の設定済みの AmazonEC2Client をクライアントとして使用します。		AmazonEC2Client
操作 （プロデューサー）	実行する操作が **必要です**。createAndRunInstances、startInstances、stopInstances、exitInstances、describeInstances、describeInstances、rebootInstances、monitorInstances、unmonitorInstances、createTags または deleteTags を指定できます。		EC2Operations
proxyHost (producer)	EC2 クライアントをインスタンス化する際にプロキシーホストを定義します。		文字列

Name	説明	デフォルト	Type
proxyPort (producer)	EC2 クライアントをインスタンス化する際にプロキシーポートを定義します。		整数
リージョン （プロデューサー）	EC2 クライアントが機能する必要があるリージョン		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

必要な EC2 コンポーネントオプション

Amazon EC2 サービスにアクセスするためにレジストリーまたは accessKey および secretKey に amazonEc2Client を指定する必要があります。

23.3. 用途

23.3.1. EC2 プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
CamelAwsEC2ImageId	**文字列**	AWS Markets のイメージ ID
CamelAwsEC2InstanceType	com.amazonaws.services.ec2.model.InstanceType	作成および実行するインスタンスタイプ

ヘッダー	Type	説明
CamelAwsEC2Operation	**文字列**	実行する操作
CamelAwsEC2InstanceMinCount	**Int**	実行するインスタンスの最小数。
CamelAwsEC2InstanceMaxCount	**Int**	実行するインスタンスの最大数。
CamelAwsEC2InstanceMonitoring	ブール値	実行中のインスタンスを監視するかどうかを定義します。
CamelAwsEC2InstanceEbsOptimized	**ブール値**	作成インスタンスが EBS I/O に対して最適化されているかどうかを定義します。
CamelAwsEC2InstanceSecurityGroups	コレクション	インスタンスに関連付けるセキュリティーグループ
CamelAwsEC2Instancesids	**コレクション**	起動、停止、説明、および終了操作を実行するインスタンス IDS のコレクション。

ヘッダー	Type	説明
CamelAwsEC2InstancesTags	**コレクション**	EC2 リソースを追加または削除するタグのコレクション

依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョン（2.16 以降）に置き換える必要があります。

23.4. 関連項目

- **Configuring Camel (Camel の設定)**
- **コンポーネント**
- **エンドポイント**
- **はじめに**
- **AWS コンポーネント**

第24章 AWS KINESIS コンポーネント

Camel バージョン 2.17 から利用可能

Kinesis コンポーネントは、Amazon Kinesis サービスからのメッセージの受信および送信をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon Kinesis を使用するためにサインアップする必要があります。詳細は、「AWS Kinesis」を参照してください。

24.1. URI 形式

```
aws-kinesis://stream-name[?options]
```

ストリームの使用前にストリームを作成する必要があります。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

24.2. URI オプション

AWS Kinesis コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS S3 のデフォルト設定		KinesisConfiguration
accessKey (common)	Amazon AWS Access Key		文字列
secretKey (common)	Amazon AWS Secret Key		文字列
リージョン（共通）	Amazon AWS リージョン		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Kinesis エンドポイントは URI 構文を使用して設定されます。

```
aws-kinesis:streamName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

24.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
streamName	ストリームの **必須** 名		文字列

24.2.2. クエリーパラメーター（30 パラメーター）：

Name	説明	デフォルト	Type
amazonKinesisClient (common)	このエンドポイントに対するすべての要求に使用する Amazon Kinesis クライアント。		AmazonKinesis
proxyHost (common)	DDBStreams クライアントをインスタンス化するときにプロキシーホストを定義します。		文字列
proxyPort (common)	DDBStreams クライアントをインスタンス化する際にプロキシーポートを定義します。		整数
リージョン （共通）	Kinesis クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
iteratorType (consumer)	Kinesis ストリームでレコードの取得を開始する場所を定義します。	TRIM_HORIZON	ShardIteratorType
maxResultsPerRequest (consumer)	各ポーリングでフェッチされる最大レコード数。	1	int
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
sequenceNumber (consumer)	ポーリングを開始するシーケンス番号。iteratorType が AFTER_SEQUENCE_NUMBER または AT_SEQUENCE_NUMBER に設定されている場合に必要です。		文字列
shardClosed (consumer)	シャード (shard) が閉じられた場合の動作を定義します。使用できる値は ignore、silent、および fail です。ignore の場合は、メッセージがログに記録され、コンシューマーは最初から再起動します。サイレントではロギングはなく、コンシューマーは最初から起動します。失敗すると ReachedClosedStateException が発生します。	ignore	KinesisShardClosed StrategyEnum
shardId (consumer)	Kinesis ストリームでどの shardId からレコードを取得するかを定義します。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel

Name	説明	デフォルト	Type
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
accessKey (security)	Amazon AWS Access Key		文字列
secretKey (security)	Amazon AWS Secret Key		文字列

必要な Kinesis コンポーネントオプション

プロキシーおよび関連するクレデンシャルを使用して、レジストリーに amazonKinesisClient を提供する必要があります。

24.3. バッチコンシューマー

このコンポーネントは Batch Consumer を実装します。

これにより、たとえば、このバッチに存在するメッセージ数を把握し、Aggregator にこの数のメッセージを集約することもできます。

24.4. 用途

24.4.1. Kinesis コンシューマーによって設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsKinesisSequenceNumber	**文字列**	レコードのシーケンス番号。サイズは API によって定義されないため、String として表されます。数値型として使用する場合は、以下を使用します。
CamelAwsKinesisApproximateArrivalTimestamp	**文字列**	レコードの Arrival 時間として割り当てられる時間 AWS。
CamelAwsKinesisPartitionKey	**文字列**	データレコードが割り当てられているストリーム内のシャードを特定します。

24.4.2. AmazonKinesis 設定

AmazonKinesisClient のインスタンスを作成して、レジストリーにバインドする必要があります。

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

Region region = Region.getRegion(Regions.fromName(region));
region.createClient(AmazonKinesisClient.class, null, clientConfiguration);
// the 'null' here is the AWSCredentialsProvider which defaults to an instance of
DefaultAWSCredentialsProviderChain

registry.bind("kinesisClient", client);
```

次に、amazonKinesisClient URI オプションで AmazonKinesisClient を参照する必要があります。

-

```
from("aws-kinesis://mykinesisstream?amazonKinesisClient=#kinesisClient")
  .to("log:out?showAll=true");
```

24.4.3. AWS 認証情報の指定

新しい ClientConfiguration インスタンスの作成時にデフォルトである DefaultAWSCredentialsProviderChain を使用して認証情報を取得することが推奨されますが、createClient(...)を呼び出す際に異なる AWSCredentialsProvider を指定することもできます。

24.4.4. Kinesis に書き込むために Kinesis プロデューサーによって使用されるメッセージヘッダー。プロデューサーは、メッセージボディーが ByteBuffer であることを想定します。

ヘッダー	Type	説明
CamelAwsKinesisPartitionKey	**文字列**	Kinesis に渡してこのレコードを保存する PartitionKey。
CamelAwsKinesisSequenceNumber	**文字列**	このレコードのシーケンス番号を示すオプションのパラメーター。

24.4.5. レコードの保存に成功した場合に Kinesis プロデューサーで設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsKinesisSequenceNumber	**文字列**	応答構文で定義されているレコードのシーケンス番号
CamelAwsKinesisShardId	**文字列**	レコードが保存されたシャード ID

24.5. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.17 以降）に置き換える必要があります。

24.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- AWS コンポーネント

第25章 AWS KINESIS FIREHOSE コンポーネント

Camel バージョン 2.19 から利用可能

Kinesis Firehose コンポーネントは、Amazon Kinesis Firehose サービスへのメッセージの送信をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon Kinesis Firehose を使用するようにサインアップする必要があります。詳細は、「AWS Kinesis Firehose」を参照してください。

25.1. URI 形式

```
aws-kinesis-firehose://delivery-stream-name[?options]
```

ストリームの使用前にストリームを作成する必要があります。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

25.2. URI オプション

AWS Kinesis Firehose コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS Kinesis Firehose デフォルト設定		KinesisFirehose Configuration
accessKey (producer)	Amazon AWS Access Key		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列
リージョン（プロデューサー）	Amazon AWS リージョン		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Kinesis Firehose エンドポイントは、URI 構文を使用して設定します。

```
aws-kinesis-firehose:streamName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

25.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
streamName	ストリームの **必須** 名		文字列

25.2.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
amazonKinesisFirehoseClient (producer)	このエンドポイントのすべての要求に使用する Amazon Kinesis Firehose クライアント。		AmazonKinesisFirehose
proxyHost (producer)	DDBStreams クライアントをインスタンス化するときにプロキシーホストを定義します。		文字列
proxyPort (producer)	DDBStreams クライアントをインスタンス化する際にプロキシーポートを定義します。		整数
リージョン（プロデューサー）	Kinesis クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
accessKey (security)	Amazon AWS Access Key		文字列
secretKey (security)	Amazon AWS Secret Key		文字列

必要な Kinesis Firehose コンポーネントオプション

プロキシーおよび関連するクレデンシャルを使用して、レジストリーに amazonKinesisClient を提供する必要があります。

25.3. 用途

25.3.1. Amazon Kinesis Firehose 設定

AmazonKinesisClient のインスタンスを作成して、レジストリーにバインドする必要があります。

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

Region region = Region.getRegion(Regions.fromName(region));
region.createClient(AmazonKinesisClient.class, null, clientConfiguration);
// the 'null' here is the AWSCredentialsProvider which defaults to an instance of
DefaultAWSCredentialsProviderChain

registry.bind("kinesisFirehoseClient", client);
```

次に、amazonKinesisFirehoseClient URI オプションで AmazonKinesisFirehoseClient を参照する必要があります。

```
from("aws-kinesis-firehose://mykinesisdeliverystream?
amazonKinesisFirehoseClient=#kinesisClient")
  .to("log:out?showAll=true");
```

25.3.2. AWS 認証情報の指定

新しい ClientConfiguration インスタンスの作成時にデフォルトである DefaultAWSCredentialsProviderChain を使用して認証情報を取得することが推奨されますが、createClient(...)を呼び出す際に異なる AWSCredentialsProvider を指定することもできます。

25.3.3. レコードの保存に成功した場合に Kinesis プロデューサーで設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsKinesisFirehoseRecordId	文字列	「レスポンス構文」で定義されているレコード ID

25.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel（2.19 以降）の実際のバージョンに置き換える必要があります。

25.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- エンドポイント
- はじめに
- AWS コンポーネント

第26章 AWS KMS コンポーネント

Camel バージョン 2.21 で利用可能

KMS コンポーネントは、AWS KMS インスタンスの作成、実行、起動、停止、および終了をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon KMS を使用するようにサインアップする必要があります。詳細は、Amazon KMS を参照してください。

26.1. URI 形式

```
aws-kms://label[?options]
```

URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

26.2. URI オプション

AWS KMS コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS MQ のデフォルト設定		KMSConfiguration
accessKey (producer)	Amazon AWS Access Key		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列
リージョン（プロデューサー）	MQ クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS KMS エンドポイントは URI 構文を使用します。

```
aws-kms:label
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

26.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
ラベル	**必要な** 論理名		文字列

26.2.2. クエリーパラメーター（8 パラメーター）：

Name	説明	デフォルト	Type
accessKey (producer)	Amazon AWS Access Key		文字列
kmsClient (producer)	既存の設定済みの AWS KMS をクライアントとして使用するには、以下を実行します。		AWSKMS
操作 （プロデューサー）	実行する操作が **必要です**		KMSOperations
proxyHost (producer)	KMS クライアントをインスタンス化する際にプロキシーホストを定義します。		文字列
proxyPort (producer)	KMS クライアントをインスタンス化する際にプロキシーポートを定義します。		整数
リージョン （プロデューサー）	KMS クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
secretKey (producer)	Amazon AWS Secret Key		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

必要な KMS コンポーネントオプション

Amazon KMS サービスにアクセスするために、レジストリーまたは accessKey および secretKey で amazonKmsClient を指定する必要があります。

26.3. 用途

26.3.1. MQ プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
CamelAwsKMSLimit	**整数**	listKeys 操作の実行中に返すキーの制限数
CamelAwsKMSOperation	**文字列**	実行する操作
CamelAwsKMSDescription	**文字列**	createKey 操作の実行時に使用するキーの説明
CamelAwsKMSKeyId	**文字列**	キー ID

依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.16 以降）に置き換える必要があります。

26.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- AWS コンポーネント

第27章 AWS LAMBDA コンポーネント

Camel バージョン 2.20 で利用可能

Lambda コンポーネントは、AWS Lambda 機能の作成、取得、一覧表示、削除、および起動をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon Lambda を使用するにはサインアップする必要があります。詳細は、Amazon Lambda を参照してください。

Lambda 関数を作成するときは、少なくとも AWSLambdaBasicExecuteRole ポリシーが割り当てられた IAM ロールを指定する必要があります。

Warning

lambda は地域サービスです。S3 バケットとは異なり、指定のリージョンで作成された Lambda 機能は、他のリージョンでは使用できません。

27.1. URI 形式

```
aws-lambda://functionName[?options]
```

URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

27.2. URI オプション

AWS Lambda コンポーネントは、以下に挙げる 5 つのオプションをサポートします。

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
Configuration (advanced)	AWS Lambda のデフォルト設定		LambdaConfiguration
accessKey (producer)	Amazon AWS Access Key		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列
リージョン （プロデューサー）	Amazon AWS リージョン		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Lambda エンドポイントは、URI 構文を使用して設定します。

```
aws-lambda:function
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

27.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
function	Lambda 関数の **必須** 名。		文字列

27.2.2. クエリーパラメーター（8 パラメーター）：

Name	説明	デフォルト	Type
操作 （プロデューサー）	実行する操作が **必要です**。これは、listFunctions、getFunction、createFunction、deleteFunction、または invokeFunction にすることができます。		LambdaOperations
リージョン （プロデューサー）	Amazon AWS リージョン		文字列

Name	説明	デフォルト	Type
awsLambdaClient (advanced)	既存の設定済みの AwsLambdaClient をクライアントとして使用します。		AWSLambda
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
proxyHost (proxy)	Lambda クライアントをインスタンス化するときにプロキシーホストを定義します。		文字列
proxyPort (proxy)	Lambda クライアントをインスタンス化するときにプロキシーポートを定義します。		整数
accessKey (security)	Amazon AWS Access Key		文字列
secretKey (security)	Amazon AWS Secret Key		文字列

必要な Lambda コンポーネントオプション

Amazon Lambda サービスにアクセスするには、レジストリーで awsLambdaClient または accessKey および secretKey を提供する必要があります。

27.3. 用途

27.3.1. Lambda プロデューサーによって評価されるメッセージヘッダー

操作	ヘッダー	Type	説明	必須

操作	ヘッダー	Type	説明	必須
All	**CamelAwsLambdaOperation**	**文字列**	実行する操作。クエリーパラメーターとして渡される操作をオーバーライド	対応
createFunction	**CamelAwsLambdaS3Bucket**	**文字列**	デプロイメントパッケージを含む .zip ファイルが保存される Amazon S3 バケット名。このバケットは、Lambda 関数を作成するのと同じ AWS リージョンに存在する必要があります。	非対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaS3Key**	**文字列**	アップロードする Amazon S3 オブジェクト（デプロイメントパッケージ）キー名。	非対応
createFunction	**CamelAwsLambdaS3ObjectVersion**	文字列	アップロードする Amazon S3 オブジェクト（デプロイメントパッケージ）のバージョン。	非対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaZipFile**	**文字列**	zip ファイルのローカルパス（デプロイメントパッケージ）。zip ファイルの内容は、メッセージボディーにも配置できます。	非対応
createFunction	**CamelAwsLambdaRole**	**文字列**	IAM ロールの Amazon Resource Name(ARN)は、他の Amazon Web Services(AWS)リソースにアクセスするために関数を実行するときに想定します。	対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaRuntime**	文字列	アップロードしている Lambda 関数のランタイム環境（nodejs、nodejs4.3、nodejs6.10、java8、python2.7、python3.6、dotnetcore1.0、odejs4.3-edge）	対応
createFunction	**CamelAwsLambdaHandler**	**文字列**	実行を開始するために Lambda が呼び出すコードの関数。Node.js の場合は、関数の module-name.export 値です。Java の場合は、package.class-name::handler または package.class-name にすることができます。	対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaDescription**	**文字列**	ユーザーによって提供される説明。	非対応
createFunction	**CamelAwsLambdaTargetArn**	**文字列**	Amazon SQS キューまたは Amazon SNS トピックのターゲット ARN(Amazon Resource Name)が含まれる親オブジェクト。	非対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaMemorySize**	**整数**	この機能に設定したメモリーサイズ（MB 単位）。64 MB の倍数でなければなりません。	非対応
createFunction	**CamelAwsLambdaKMSKeyArn**	**文字列**	関数の環境変数の暗号化に使用される KMS キーの Amazon Resource Name(ARN)。指定しないと、AWS Lambda はデフォルトのサービスキーを使用します。	非対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaPublish**	**ブール値**	このブール値パラメーターを使用して、AWS Lambda に対して Lambda 関数を作成し、バージョンをアトミック操作として公開することができます。	非対応
createFunction	**CamelAwsLambdaTimeout**	**整数**	Lambda が関数を終了する関数の実行時間。デフォルトは 3 秒です。	非対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaTracingConfig**	**文字列**	関数のトレース設定（Active または PassThrough）。	非対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaEnvironmentVariables**	**Map<String, String>**	ご使用の環境設定を表すキーと値のペアです。	非対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaEnvironmentTags**	**Map<String, String>**	新しい関数に割り当てられたタグ（キーと値のペア）の一覧。	非対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaSecurityGroupIds**	**List<String>**	Lambda 関数が VPC のリソースにアクセスする場合、VPC 内の 1 つ以上のセキュリティーグループ ID の一覧。	非対応

操作	ヘッダー	Type	説明	必須
createFunction	**CamelAwsLambdaSubnetIds**	**List<String>**	Lambda 関数が VPC のリソースにアクセスする場合、VPC 内の 1 つ以上のサブネット ID の一覧。	非対応

依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョン（2.16 以降）に置き換える必要があります。

27.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- AWS コンポーネント

第28章 AWS MQ コンポーネント

Camel バージョン 2.21 で利用可能

EC2 コンポーネントは、AWS MQ インスタンスの作成、実行、起動、停止、および終了をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon MQ を使用するにはサインアップする必要があります。詳細は、Amazon MQ を参照してください。

28.1. URI 形式

```
aws-mq://label[?options]
```

URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

28.2. URI オプション

AWS MQ コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS MQ のデフォルト設定		MQConfiguration
accessKey (producer)	Amazon AWS Access Key		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列
リージョン（プロデューサー）	MQ クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS MQ エンドポイントは、URI 構文を使用して設定します。

```
aws-mq:label
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

28.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
ラベル	**必要な** 論理名		文字列

28.2.2. クエリーパラメーター（8 パラメーター）：

Name	説明	デフォルト	Type
accessKey (producer)	Amazon AWS Access Key		文字列
amazonMqClient (producer)	既存の設定済みの AmazonMQClient をクライアントとして使用します。		AmazonMQ
操作 （プロデューサー）	実行する操作が **必要です**。It can be listBrokers,createBroker,deleteBroker		MQOperations
proxyHost (producer)	MQ クライアントをインスタンス化するときにプロキシーホストを定義します。		文字列
proxyPort (producer)	MQ クライアントをインスタンス化するときにプロキシーポートを定義します。		整数
リージョン （プロデューサー）	MQ クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
secretKey (producer)	Amazon AWS Secret Key		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

必要な EC2 コンポーネントオプション

Amazon EC2 サービスにアクセスするためにレジストリーまたは accessKey および secretKey に amazonEc2Client を指定する必要があります。

28.3. 用途

28.3.1. MQ プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
CamelAwsMQMaxResults	**文字列**	listBrokers 操作から取得する必要のある結果の数
CamelAwsMQBrokerName	**文字列**	ブローカー名
CamelAwsMQOperation	**文字列**	実行する操作
CamelAwsMQBrokerId	**文字列**	ブローカーID

ヘッダー	Type	説明
CamelAwsMQBrokerDeploymentMode	**文字列**	createBroker 操作のブローカーのデプロイメントモード

依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.16 以降）に置き換える必要があります。

28.4. 関連項目

- **Configuring Camel (Camel の設定)**
- **コンポーネント**
- **エンドポイント**
- **はじめに**

- **AWS コンポーネント**

第29章 AWS S3 STORAGE SERVICE コンポーネント

Camel バージョン 2.8 から利用可能

S3 コンポーネントは、Amazon の S3 サービスから objetc の保存および取得をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon S3 を使用するにはサインアップする必要があります。詳細は、Amazon S3 を参照してください。

29.1. URI 形式

```
aws-s3://bucketNameOrArn[?options]
```

バケットが存在しない場合は作成されます。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

たとえば、バケット helloBucket から hello.txt ファイルを読み取るには、以下のスニペットを使用します。

```
from("aws-s3:helloBucket?
accessKey=yourAccessKey&secretKey=yourSecretKey&prefix=hello.txt")
  .to("file:/var/downloaded");
```

29.2. URI オプション

AWS S3 Storage Service コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS S3 のデフォルト設定		S3Configuration
accessKey (common)	Amazon AWS Access Key		文字列

Name	説明	デフォルト	Type
secretKey (common)	Amazon AWS Secret Key		文字列
リージョン （共通）	バケットのあるリージョン。このオプションは com.amazonaws.services.s3.model.CreateBucketRequest で使用されます。		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS S3 Storage Service エンドポイントは URI 構文を使用します。

```
aws-s3:bucketNameOrArn
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

29.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
bucketNameOrArn	**必要な** バケット名または ARN。		文字列

29.2.2. クエリーパラメーター（50 パラメーター）：

Name	説明	デフォルト	Type
amazonS3Client (common)	link:registry.htmlRegistry の com.amazonaws.services.sqs.AmazonS3 への参照。		AmazonS3
pathStyleAccess (common)	S3 クライアントがパススタイルアクセスを使用するかどうか。	false	boolean
ポリシー （共通）	com.amazonaws.services.s3.AmazonS3setBucketPolicy（）メソッドに設定されるこのキューのポリシー。		文字列

Name	説明	デフォルト	Type
proxyHost (common)	SQS クライアントをインスタンス化するときにプロキシーホストを定義します。		文字列
proxyPort (common)	クライアント定義内で使用されるプロキシーポートを指定します。		整数
リージョン （共通）	S3 クライアントが機能する必要があるリージョン		文字列
useIAMCredentials (common)	S3 クライアントが EC2 インスタンスでクレデンシャルをロードすることを期待すべきか、または静的クレデンシャルが渡されることを期待すべきかどうかを設定します。	false	boolean
encryptionMaterials (common)	対称/非対称クライアントの使用時に使用する暗号化資料		EncryptionMaterials
useEncryption (common)	暗号化を使用する必要があるかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
deleteAfterRead (consumer)	取得後に S3 からオブジェクトを削除します。削除は、エクスチェンジがコミットされた場合にのみ実行されます。ロールバックが発生すると、オブジェクトは削除されません。このオプションが false の場合、同じオブジェクトがポーリングで繰り返し取得されます。そのため、ルートで Idempotent Consumer EIP を使用して重複を除外する必要があります。リンク S3ConstantsBUCKET_NAME とリンク S3ConstantsKEY ヘッダーを使用するか、リンク S3ConstantsKEY ヘッダーのみを使用してフィルターを設定できます。	true	boolean
fileName (consumer)	指定のファイル名を持つバケットからオブジェクトを取得します。		文字列

Name	説明	デフォルト	Type
includeBody (consumer)	true の場合、エクスチェンジボディーはファイルの内容へのストリームに設定されます。false の場合、ヘッダーは S3 オブジェクトメタデータで設定されますが、ボディーは null になります。このオプションは、autocloseBody オプションと密接に関係します。includeBody を true に設定し、autocloseBody を false に設定すると、S3Object ストリームを閉じるのは呼び出し元によって異なります。autocloseBody を true に設定すると、S3Object ストリームが自動的に閉じられます。	true	boolean
maxConnections（コンシューマー）	S3 クライアント設定の maxConnections パラメーターを設定します。	60	int
maxMessagesPerPoll (consumer)	各ポーリングのポーリング制限としてメッセージの最大数を取得します。デフォルトは無制限ですが、0 または負の値を使用して無制限として無効にします。	10	int
prefix (consumer)	対象のオブジェクトのみを消費するために com.amazonaws.services.s3.model.ListObjectsRequest で使用されるプレフィックス。		文字列
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
autocloseBody（コンシューマー）	このオプションが true で includeBody が true の場合、エクスチェンジの完了時に S3Object.close（）メソッドが呼び出されます。このオプションは includeBody オプションと密接に関係しています。includeBody を true に設定し、autocloseBody を false に設定すると、S3Object ストリームを閉じるのは呼び出し元によって異なります。autocloseBody を true に設定すると、S3Object ストリームが自動的に閉じられます。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
deleteAfterWrite (producer)	S3 ファイルのアップロード後にファイルオブジェクトを削除します。	false	boolean
multiPartUpload (producer)	true の場合、Camel はマルチパート形式のファイルをアップロードし、パートサイズは partSize のオプションによって決定されます。	false	boolean
操作 （プロデューサー）	ユーザーがアップロードのみを実行したくない場合に実行する操作		S3Operations
partSize (producer)	マルチパートのアップロードで使用される partSize を設定します。デフォルトのサイズは 25M です。	26214400	Long
serverSideEncryption (producer)	AWS 管理のキーを使用してオブジェクトを暗号化する際にサーバー側の暗号化アルゴリズムを設定します。たとえば、AES256 を使用します。		文字列
storageClass (producer)	com.amazonaws.services.s3.model.PutObjectRequest リクエストに設定するストレージクラス。		文字列
awsKMSKeyId (producer)	KMS が有効になっている場合に使用する KMS キーの ID を定義します。		文字列
useAwsKMS (producer)	KMS を使用する必要があるかどうかを定義します。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
accelerateModeEnabled (advanced)	Accelerate モードが true または false かどうかを定義します。	false	boolean

Name	説明	デフォルト	Type
chunkedEncodingDisabled (advanced)	無効な Chunked Encoding が true または false になるかどうかを定義します。	false	boolean
dualstackEnabled (advanced)	Dualstack が有効かどうかを定義します。	false	boolean
forceGlobalBucketAccessEnabled (advanced)	強制グローバルバケットアクセスが true または false であるかどうかを定義します。	false	boolean
payloadSigningEnabled (advanced)	Payload 署名が有効かどうかを定義します。true または false	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel

Name	説明	デフォルト	Type
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
accessKey (security)	Amazon AWS Access Key		文字列
secretKey (security)	Amazon AWS Secret Key		文字列

必要な S3 コンポーネントオプション

Amazon の S3Client にアクセスするためにレジストリーまたは accessKey および secretKey で amazonS3Client を提供する必要があります。

29.3. バッチコンシューマー

このコンポーネントは Batch Consumer を実装します。

これにより、たとえば、このバッチに存在するメッセージ数を把握し、Aggregator にこの数のメッセージを集約することもできます。

29.4. 用途

29.4.1. S3 プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
CamelAwsS3BucketName	**文字列**	このオブジェクトが保管されるバケット名、または現在の操作に使用されるバケット名。
CamelAwsS3BucketDestinationName	**文字列**	Camel 2.18: 現在の操作に使用されるバケット宛先名
CamelAwsS3ContentLength	**Long**	このオブジェクトの内容の長さ。
CamelAwsS3ContentType	**文字列**	このオブジェクトのコンテンツタイプ。
CamelAwsS3ContentControl	**文字列**	Camel 2.8.2: このオブジェクトのコンテンツ管理。
CamelAwsS3ContentDisposition	**文字列**	Camel 2.8.2: このオブジェクトのコンテンツの配置。

ヘッダー	Type	説明
CamelAwsS3ContentEncoding	**文字列**	Camel 2.8.2: このオブジェクトのコンテンツのエンコーディング。
CamelAwsS3ContentMD5	**文字列**	Camel 2.8.2: このオブジェクトの md5 チェックサム。
CamelAwsS3DestinationKey	**文字列**	Camel 2.18: 現在の操作に使用される Destination キー
CamelAwsS3Key	**文字列**	このオブジェクトを保存するキー、または現在の操作に使用されるキー
CamelAwsS3LastModified	**java.util.Date**	Camel 2.8.2: このオブジェクトの最後に変更されたタイムスタンプ。
CamelAwsS3Operation	**文字列**	Camel 2.18: 実行する操作。使用できる値は copyObject、listBuckets、deleteBucket、downloadLink です。
CamelAwsS3StorageClass	**文字列**	Camel 2.8.4: このオブジェクトのストレージクラス。
CamelAwsS3CannedAcl	**文字列**	Camel 2.11.0: オブジェクトに適用されるカナリア acl。許可される値は **com.amazonaws.services.s3.model.CannedAccessControlList** を参照してください。

ヘッダー	Type	説明
CamelAwsS3Acl	**com.amazonaws.services.s3.model.AccessControlList**	Camel 2.11.0: 適切に構築された Amazon S3 Access Control List オブジェクト。詳細は **com.amazonaws.services.s3.model.AccessControlList** を参照してください。
CamelAwsS3Headers	**Map<String,String>**	Camel 2.15.0: カスタム objectMetadata ヘッダーを取得または設定するサポート。
CamelAwsS3ServerSideEncryption	文字列	Camel 2.16: AWS 管理のキーを使用してオブジェクトを暗号化する際にサーバー側の暗号化アルゴリズムを設定します。たとえば、AES256 を使用します。
CamelAwsS3VersionId	**文字列**	現在の操作から保存または返されるオブジェクトのバージョン ID

29.4.2. S3 プロデューサーによって設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsS3ETag	**文字列**	新たにアップロードしたオブジェクトの ETag 値。
CamelAwsS3VersionId	**文字列**	新たにアップロードしたオブジェクトの **任意** のバージョン ID。

ヘッダー	Type	説明
CamelAwsS3DownloadLinkExpiration	**文字列**	URL ダウンロードリンクの有効期限（ミリ秒単位）。リンクは **CamelAwsS3DownloadLink** 応答ヘッダーに保存されます。

29.4.3. S3 コンシューマーによって設定されるメッセージヘッダー

ヘッダー	Type	説明
CamelAwsS3Key	**文字列**	このオブジェクトが保存されるキー。
CamelAwsS3BucketName	**文字列**	このオブジェクトが含まれるバケットの名前。
CamelAwsS3ETag	**文字列**	RFC 1864 に従って、関連付けられたオブジェクトの 16 進エンコードされた 128 ビット MD5 ダイジェスト。このデータは整合性チェックとして使用され、呼び出し元によって受信されるデータが Amazon S3 によって送信されたデータと同じであることを確認します。
CamelAwsS3LastModified	**Date**	Last-Modified ヘッダーの値。Amazon S3 が最後に関連付けられたオブジェクトに対する変更を記録した日時を示します。
CamelAwsS3VersionId	**文字列**	関連付けられた Amazon S3 オブジェクトのバージョン ID（ある場合）。バージョン ID は、オブジェクトのバージョン管理が有効な Amazon S3 バケットにオブジェクトをアップロードする場合にのみ、オブジェクトに割り当てられます。
CamelAwsS3ContentType	**文字列**	関連付けられたオブジェクトに保存されているコンテンツタイプを示す Content-Type HTTP ヘッダー。このヘッダーの値は標準の MIME タイプです。

ヘッダー	Type	説明
CamelAwsS3ContentMD5	**文字列**	RFC 1864 に従って、関連付けられたオブジェクトの base64 でエンコードされた 128 ビット MD5 ダイジェスト（ヘッダーを含まないコンテンツ）このデータは、Amazon S3 が受信したデータが呼び出し元によって送信されるデータと同じかどうかを確認するためにメッセージ整合性チェックとして使用されます。
CamelAwsS3ContentLength	**Long**	関連付けられたオブジェクトのサイズ（バイト単位）を示す Content-Length HTTP ヘッダー。
CamelAwsS3ContentEncoding	**文字列**	オブジェクトに適用されるコンテンツエンコーディングを指定する **任意** の Content-Encoding HTTP ヘッダーと、Content-Type フィールドによって参照されるメディアタイプを取得するために、どのようなデコードメカニズムを適用する必要があります。
CamelAwsS3ContentDisposition	**文字列**	**オプション** の Content-Disposition HTTP ヘッダー。オブジェクトの保存に推奨されるファイル名などのプレゼンテーション情報を指定します。
CamelAwsS3ContentControl	**文字列**	**任意** の Cache-Control HTTP ヘッダー。これにより、ユーザーは HTTP リクエスト/応答チェーンでキャッシュ動作を指定できます。
CamelAwsS3ServerSideEncryption	文字列	**Camel 2.16:** AWS 管理のキーを使用してオブジェクトを暗号化する際のサーバー側の暗号化アルゴリズム。

29.4.4. 高度な AmazonS3 設定

Camel アプリケーションがファイアウォールの内側で実行されている場合や、AmazonS3 インスタンス設定をより詳細に制御する必要がある場合は、独自のインスタンスを作成できます。

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey",
"mySecretKey");
```

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonS3 client = new AmazonS3Client(awsCredentials, clientConfiguration);

registry.bind("client", client);
```

そして、Camel aws-s3 コンポーネント設定で参照します。

```
from("aws-s3://MyBucket?amazonS3Client=#client&delay=5000&maxMessagesPerPoll=5")
.to("mock:result");
```

29.4.5. S3 コンポーネントでの KMS の使用

AWS インフラストラクチャーを使用して AWS KMS を使用してデータを暗号化/復号化するには、以下の例のように 2.21.x で導入されたオプションを使用できます。

```
from("file:tmp/test?fileName=test.txt")
    .setHeader(S3Constants.KEY, constant("testFile"))
    .to("aws-s3://mybucket?
amazonS3Client=#client&useAwsKMS=true&awsKMSKeyId=3f0637ad-296a-3dfe-a796-
e60654fb128c");
```

このようにして、S3 に依頼されます。KMS キー 3f0637ad-296a-3dfe-a796-e60654fb128c を使用してファイル test.txt を暗号化します。このファイルをダウンロードするように依頼すると、ダウンロード前に復号が直接行われます。

29.4.6. s3 コンポーネントでの「useIAMCredentials」の使用

AWS IAM 認証情報を使用するには、最初に Camel アプリケーションを起動する EC2 に、効果的に実行するために適切なポリシーが含まれる IAM ロールが割り当てられていることを確認する必要があります。この機能は、リモートインスタンスでのみ「true」に設定する必要があります。さらに明確にするために、IAM は AWS 固有のコンポーネントであるため、静的認証情報をローカルで使用する必要があります。ただし、AWS 環境は管理しやすくなるはずです。これを実装して理解した後に、AWS 環境についてクエリーパラメーター「useIAMCredentials」を「true」に設定できます。ローカル環境およびリモート環境に基づいてこれをオンまたはオフにするには、システム環境変数を使用してこのクエリーパラメーターを有効にすることを検討してください。たとえば、「isRemote」と呼ばれるシステム環境変数が true に設定されている場合、コードは「useIAMCredentials」クエリーパラメーターを「true」に設定する可能性があります（これを実行するための他の多くの方法があり、これは簡単な例として機能する必要があります）。静的認証情報の完全な必要はなくなるわけではありませんが、AWS 環境で IAM 認証情報を使用すると、リモート環境での IAM 認証情報の更新が不要になり、主要なセキュリティーブートストラップが追加されます（IAM 認証情報は 6 時間ごとに自動的に更新され、ポリシーが更新されると更新されます）。これは AWS で認証情報を管理する方法として推奨されるため、できるだけ多く使用する必要があります。

29.5. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.8 以降）に置き換える必要があります。

29.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- AWS コンポーネント

第30章 AWS SIMPLEDB コンポーネント

Camel バージョン 2.9 で利用可能

sdb コンポーネントは、Amazon の SDB サービスからのデータの保存および取得を サポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon SDB を使用するにはサインアップする必要があります。詳細は、Amazon SDB を参照してください。

30.1. URI 形式

```
aws-sdb://domainName[?options]
```

URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

30.2. URI オプション

AWS SimpleDB コンポーネントにはオプションがありません。

AWS SimpleDB エンドポイントは URI 構文を使用して設定します。

```
aws-sdb:domainName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

30.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
domainName	**必須**。現在作業しているドメイン名です。		文字列

30.2.2. クエリーパラメーター（10 パラメーター）：

Name	説明	デフォルト	Type
accessKey (producer)	Amazon AWS Access Key		文字列
amazonSDBClient (producer)	AmazonSimpleDB をクライアントとして使用します。		AmazonSimpleDB
consistentRead (producer)	データの読み取り時に強力な整合性を適用するべきかどうかを決定します。	false	boolean
maxNumberOfDomains (producer)	返されるドメイン名の最大数。範囲は 1 から 100 です。		整数
操作 （プロデューサー）	実行する操作	PutAttributes	SdbOperations
proxyHost (producer)	SDB クライアントをインスタンス化する際にプロキシーホストを定義します。		文字列
proxyPort (producer)	SDB クライアントをインスタンス化する際にプロキシーポートを定義します。		整数
リージョン （プロデューサー）	SDB クライアントが機能する必要があるリージョン		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

必要な SDB コンポーネントオプション

レジストリーで amazonSDBClient を指定するか、または Amazon の SDB にアクセスするために accessKey および secretKey を提供する必要があります。

30.3. 用途

30.3.1. SDB プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSdbAttributes	**collection<Attribute>**	操作する属性のリスト。
CamelAwsSdbAttributeNames	**collection<String>**	取得する属性の名前。
CamelAwsSdbConsistentRead	**ブール値**	データの読み取り時に強力な整合性を適用するべきかどうかを決定します。
CamelAwsSdbDeletableItems	**collection<DeletableItem>**	バッチで削除操作を実行する項目の一覧。
CamelAwsSdbDomainName	**文字列**	現在作業中のドメイン名。
CamelAwsSdbItemName	**文字列**	この項目の一意のキー
CamelAwsSdbMaxNumberOfDomains	**整数**	返されるドメイン名の最大数。範囲は 1* から 100 までです。
CamelAwsSdbNextToken	**文字列**	ドメイン/項目名の次のリストを開始する場所を指定する文字列。

ヘッダー	Type	説明
CamelAwsSdbOperation	**文字列**	URI オプションから操作を上書きする。
CamelAwsSdbReplaceableAttributes	**Collection<ReplaceableAttribute>**	アイテムに追加する属性のリスト。
CamelAwsSdbReplaceableItems	**Collection<ReplaceableItem>**	ドメインに追加する項目の一覧。
CamelAwsSdbSelectExpression	**文字列**	ドメインのクエリーに使用される式。
CamelAwsSdbUpdateCondition	**UpdateCondition**	指定した場合には、指定した属性が更新/削除されるかどうかを決定する更新条件。

30.3.2. DomainMetadata 操作中に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSdbTimestamp	**整数**	メタデータの計算時のデータと時間(UNIX)秒。

ヘッダー	Type	説明
CamelAwsSdbItemCount	**整数**	ドメインのすべての項目数。
CamelAwsSdbAttributeNameCount	**整数**	ドメインの一意の属性名の数。
CamelAwsSdbAttributeValueCount	**整数**	ドメイン内のすべての属性名と値のペアの数。
CamelAwsSdbAttributeNameSize	**Long**	ドメイン内のすべての一意の属性名（バイト単位）の合計サイズ。
CamelAwsSdbAttributeValueSize	**Long**	ドメインのすべての属性値の合計サイズ（バイト単位）。
CamelAwsSdbItemNameSize	**Long**	ドメイン内のすべての項目名の合計サイズ（バイト単位）。

30.3.3. GetAttributes 操作時に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSdbAttributes	**List<Attribute>**	操作によって返される属性の一覧です。

30.3.4. ListDomains 操作時に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSdbDomainNames	**List<String>**	式に一致するドメイン名の一覧。
CamelAwsSdbNextToken	**文字列**	指定された MaxNumberOfDomains を超えるドメインが依然として利用できることを示す不透明なトークンです。

30.3.5. Select 操作時に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSdbItems	**List<Item>**	select 式に一致する項目の一覧。
CamelAwsSdbNextToken	**文字列**	MaxNumberOfItems を超える項目が一致するか、応答サイズが 1 メガバイトを超えたか、または実行時間が 5 秒を超えることを示す不透明なトークン。

30.3.6. AmazonSimpleDB の高度な設定

AmazonSimpleDB インスタンス設定に対する制御を強化する必要がある場合は、独自のインスタンスを作成して URI から参照することができます。

```
from("direct:start")
.to("aws-sdb://domainName?amazonSDBClient=#client");
```

#client はレジストリーの AmazonSimpleDB を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合は、以下のようになります。

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey",
"mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonSimpleDB client = new AmazonSimpleDBClient(awsCredentials, clientConfiguration);

registry.bind("client", client);
```

30.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws</artifactId>
    <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.8.4 以降）に置き換える必要があります。

30.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- エンドポイント
- はじめに
- AWS コンポーネント

第31章 AWS SIMPLE EMAIL SERVICE コンポーネント

Camel バージョン 2.9 で利用可能

ses コンポーネントは、Amazon の SES サービスでの メール送信をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon SES を使用できるようにサインアップする必要があります。詳細は、Amazon SES を参照してください。

31.1. URI 形式

```
aws-ses://from[?options]
```

URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

31.2. URI オプション

AWS Simple Email Service コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS SES のデフォルト設定		SesConfiguration
accessKey (producer)	Amazon AWS Access Key		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列
リージョン（プロデューサー）	SES クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Simple Email Service エンドポイントは、URI 構文を使用して設定します。

```
aws-ses:from
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

31.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
from	**必須**: 送信者のメールアドレス。		文字列

31.2.2. クエリーパラメーター（11 パラメーター）：

Name	説明	デフォルト	Type
amazonSESClient (producer)	AmazonSimpleEmailService をクライアントとして使用します。		AmazonSimpleEmail Service
proxyHost (producer)	SES クライアントをインスタンス化するときにプロキシーホストを定義します。		文字列
proxyPort (producer)	SES クライアントをインスタンス化する際にプロキシーポートを定義します。		整数
リージョン（プロデューサー）	SES クライアントが機能する必要があるリージョン		文字列
replyToAddresses (producer)	メッセージの返信先メールアドレスのリスト。'CamelAwsSesReplyToAddresses' ヘッダーを使用して上書きします。		リスト

Name	説明	デフォルト	Type
returnPath (producer)	バウンス通知を転送するメールアドレス。'CamelAwsSesReturnPath' ヘッダーを使用して上書きします。		文字列
Subject (producer)	メッセージヘッダー 'CamelAwsSesSubject' が存在しない場合に使用されるサブジェクト。		文字列
to （プロデューサー）	宛先メールアドレスの一覧。'CamelAwsSesTo' ヘッダーでオーバーライドできます。		リスト
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
accessKey (security)	Amazon AWS Access Key		文字列
secretKey (security)	Amazon AWS Secret Key		文字列

必要な SES コンポーネントオプション

Amazon の SES にアクセスするためにレジストリーまたは accessKey および secretKey で amazonSESClient を提供する必要があります。

31.3. 用途

31.3.1. SES プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSesFrom	**文字列**	送信者のメールアドレス。

ヘッダー	Type	説明
CamelAwsSesTo	**List<String>**	この電子メールの宛先。
CamelAwsSesSubject	**文字列**	メッセージの件名。
CamelAwsSesReplyToAddresses	**List<String>**	メッセージの返信先メールアドレス。
CamelAwsSesReturnPath	**文字列**	バウンス通知を転送するメールアドレス。
CamelAwsSesHtmlEmail	**ブール値**	Camel 2.12.3 **以降**、電子メールコンテンツが HTML であるかどうかを示すフラグ。

31.3.2. SES プロデューサーによって設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSesMessageId	**文字列**	Amazon SES メッセージ ID。

31.3.3. 高度な AmazonSimpleEmailService 設定

AmazonSimpleEmailService インスタンス設定に対する制御を強化する必要がある場合は、独自のインスタンスを作成して URI から参照することができます。

```
from("direct:start")
.to("aws-ses://example@example.com?amazonSESClient=#client");
```

#client はレジストリーの AmazonSimpleEmailService を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合は、以下のようになります。

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey",
"mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);
AmazonSimpleEmailService client = new AmazonSimpleEmailServiceClient(awsCredentials,
clientConfiguration);

registry.bind("client", client);
```

31.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.8.4 以降）に置き換える必要があります。

31.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント

- **はじめに**

- **AWS コンポーネント**

第32章 AWS SIMPLE NOTIFICATION SYSTEM コンポーネント

Camel バージョン 2.8 から利用可能

SNS コンポーネントを使用すると、メッセージを Amazon Simple Notification Topic に送信できます。Amazon API の実装は AWS SDK によって提供されます。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon SNS を使用するにはサインアップする必要があります。詳細は、Amazon SNS を参照してください。

32.1. URI 形式

```
aws-sns://topicNameOrArn[?options]
```

トピックが存在しない場合は作成されます。
URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

32.2. URI オプション

AWS Simple Notification System コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS SNS のデフォルト設定		SnsConfiguration
accessKey (producer)	Amazon AWS Access Key		文字列
secretKey (producer)	Amazon AWS Secret Key		文字列
リージョン（プロデューサー）	SNS クライアントが機能する必要があるリージョン		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Simple Notification System エンドポイントは、URI 構文を使用して設定します。

```
aws-sns:topicNameOrArn
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

32.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
topicNameOrArn	**必要な** トピック名または ARN。		文字列

32.2.2. クエリーパラメーター（11 パラメーター）：

Name	説明	デフォルト	Type
amazonSNSClient (producer)	AmazonSNS をクライアントとして使用します。		AmazonSNS
headerFilterStrategy (producer)	カスタムの HeaderFilterStrategy を使用して、ヘッダーから Camel または Camel からヘッダーにマッピングします。		HeaderFilterStrategy
messageStructure (producer)	json などの使用するメッセージ構造。		文字列
ポリシー （プロデューサー）	このキューのポリシー		文字列
proxyHost (producer)	SNS クライアントをインスタンス化するときにプロキシーホストを定義します。		文字列

Name	説明	デフォルト	Type
proxyPort (producer)	SNS クライアントをインスタンス化するときにプロキシーポートを定義します。		整数
リージョン （プロデューサー）	SNS クライアントが機能する必要があるリージョン		文字列
Subject (producer)	メッセージヘッダー 'CamelAwsSnsSubject' が存在しない場合に使用されるサブジェクト。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
accessKey (security)	Amazon AWS Access Key		文字列
secretKey (security)	Amazon AWS Secret Key		文字列

必要な SNS コンポーネントオプション

レジストリーで amazonSNSClient を指定するか、または Amazon の SNS にアクセスするために accessKey および secretKey を提供する必要があります。

32.3. 用途

32.3.1. SNS プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSnsSubject	**文字列**	Amazon SNS メッセージサブジェクト。設定されていない場合は、**SnsConfiguration** からのサブジェクトが使用されます。

32.3.2. SNS プロデューサーによって設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSnsMessageId	**文字列**	Amazon SNS メッセージ ID。

32.3.3. 高度な AmazonSNS 設定

AmazonSNS インスタンス設定をより制御する必要がある場合は、独自のインスタンスを作成して URI から参照することができます。

```
from("direct:start")
.to("aws-sns://MyTopic?amazonSNSClient=#client");
```

#client はレジストリーの AmazonSNS を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合は、以下のようになります。

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey",
"mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);
AmazonSNS client = new AmazonSNSClient(awsCredentials, clientConfiguration);

registry.bind("client", client);
```

32.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws</artifactId>
    <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョン（2.8 以降）に置き換える必要があります。

32.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- AWS コンポーネント

第33章 AWS SIMPLE QUEUE SERVICE コンポーネント

Camel バージョン 2.6 で利用可能

sqs コンポーネントは、Amazon の SQS サービスへのメッセージの 送受信をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon SQS を使用するにはサインアップする必要があります。詳細は、Amazon SQS を参照してください。

33.1. URI 形式

```
aws-sqs://queueNameOrArn[?options]
```

キューが存在しない場合は作成されます。
URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

33.2. URI オプション

AWS Simple Queue Service コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS SQS のデフォルト設定		SqsConfiguration
accessKey (common)	Amazon AWS Access Key		文字列
secretKey (common)	Amazon AWS Secret Key		文字列
リージョン（共通）	service URL をビルドするために queueOwnerAWSAccountId で使用できるキューリージョンを指定します。		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Simple Queue Service エンドポイントは、URI 構文を使用して設定します。

```
aws-sqs:queueNameOrArn
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

33.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
queueNameOrArn	***必須*** キュー名または ARN。		文字列

33.2.2. クエリーパラメーター（46 パラメーター）：

Name	説明	デフォルト	Type
amazonAWSHost (common)	Amazon AWS クラウドのホスト名。	amazonaws.com	文字列
amazonSQSClient (common)	AmazonSQS をクライアントとして使用します。		AmazonSQS
headerFilterStrategy (common)	カスタムの HeaderFilterStrategy を使用して、ヘッダーから Camel または Camel からヘッダーにマッピングします。		HeaderFilterStrategy
queueOwnerAWSAccountId (common)	異なるアカウント所有者でキューを接続する必要がある場合は、キュー所有者の aws アカウント ID を指定します。		文字列

Name	説明	デフォルト	Type
リージョン （共通）	service URL をビルドするために queueOwnerAWSAccountId で使用できるキューリージョンを指定します。		文字列
attributeNames (consumer)	消費時に受け取る属性名のリスト。複数の名前はコンマで区切ることができます。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	複数のスレッドを使用して sqs キューをポーリングしてスループットを高めることが可能	1	int
defaultVisibilityTimeout (consumer)	デフォルトの可視性タイムアウト（秒単位）		整数
deleteAfterRead (consumer)	読み取り後に SQS からメッセージを削除します。	true	boolean
deleteIfFiltered (consumer)	エクスチェンジがフィルターを経由できない場合は DeleteMessage を SQS キューに送信するかどうか。'false' とエクスチェンジがルートの Camel フィルターアップストリームを介して行われない場合は、DeleteMessage を送信しないでください。	true	boolean
extendMessageVisibility (consumer)	有効にすると、スケジュールされたバックグラウンドタスクでは SQS のメッセージの表示が維持されます。これは、メッセージの処理に長い時間がかかる場合に必要です。true に設定した場合は、defaultVisibilityTimeout を設定する必要があります。詳細は、Amazon ドキュメント を参照してください。	false	boolean
maxMessagesPerPoll (consumer)	各ポーリングのポーリング制限としてメッセージの最大数を取得します。デフォルトは無制限ですが、0 または負の値を使用して無制限として無効にします。		int

Name	説明	デフォルト	Type
messageAttributeNames (consumer)	消費時に受信するメッセージ属性名のリスト。複数の名前はコンマで区切ることができます。		文字列
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
visibilityTimeout (consumer)	com.amazonaws.services.sqs.model.SetQueueAttributesRequest に設定される ReceiveMessage リクエストによって取得された後、受信したメッセージが後続の取得リクエストから表示されない期間（秒単位）。これは、defaultVisibilityTimeout とは異なる場合にのみ有効です。キューの可視性のタイムアウト属性を永続的に変更します。		整数
waitTimeSeconds (consumer)	ReceiveMessage アクション呼び出しが応答に含めるキューにメッセージが待機する期間（0 から 20）。		整数
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
delaySeconds (producer)	数秒間メッセージの送信を遅延します。		整数
messageDeduplicationId Strategy (producer)	FIFOキューの場合のみ。メッセージに messageDeduplicationId を設定するストラテジー。useExchangeId または useContentBasedDeduplication のいずれかをオプションとして使用できます。useContentBasedDeduplication オプションでは、メッセージに messageDeduplicationId が設定されません。	useExchangeId	MessageDeduplicationId Strategy

Name	説明	デフォルト	Type
messageGroupIdStrategy (producer)	FIFOキューの場合のみ。メッセージに messageGroupId を設定するストラテジー。useConstant、useExchangeId、usePropertyValue のいずれかをオプションとして使用できます。usePropertyValue オプションでは、CamelAwsMessageGroupId プロパティーの値が使用されます。		MessageGroupIdStrategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel

Name	説明	デフォルト	Type
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
proxyHost (proxy)	SQS クライアントをインスタンス化するときにプロキシーホストを定義します。		文字列
proxyPort (proxy)	SQS クライアントをインスタンス化するときにプロキシーポートを定義します。		整数
maximumMessageSize (queue)	SQS メッセージに含まれる最大MessageSize（バイト単位）。		整数
messageRetentionPeriod (queue)	messageRetentionPeriod（秒単位）は、このキューの SQS によってメッセージが保持されます。		整数
ポリシー（キュー）	このキューのポリシー		文字列
receiveMessageWaitTime Seconds (queue)	要求で WaitTimeSeconds を指定しない場合、キュー属性 ReceiveMessageWaitTimeSeconds を使用して待機する期間を決定します。		整数
redrivePolicy (queue)	DeadLetter キューに送信するポリシーを指定します。詳細は、Amazon ドキュメント を参照してください。		文字列

Name	説明	デフォルト	Type
accessKey (security)	Amazon AWS Access Key		文字列
secretKey (security)	Amazon AWS Secret Key		文字列

必要な SQS コンポーネントオプション

レジストリーで amazonSQSClient を指定するか、または Amazon の SQS にアクセスするために accessKey および secretKey を提供する必要があります。

33.3. バッチコンシューマー

このコンポーネントは Batch Consumer を実装します。

これにより、たとえば、このバッチに存在するメッセージ数を把握し、Aggregator にこの数のメッセージを集約することもできます。

33.4. 用途

33.4.1. SQS プロデューサーによって設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSqsMD5OfBody	**文字列**	Amazon SQS メッセージの MD5 チェックサム。
CamelAwsSqsMessageId	**文字列**	Amazon SQS メッセージ ID。

ヘッダー	Type	説明
CamelAwsSqsDelaySeconds	**整数**	Camel 2.11 以降、Amazon SQS メッセージが他の遅延秒単位で確認できます。

33.4.2. SQS コンシューマーによって設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelAwsSqsMD5OfBody	**文字列**	Amazon SQS メッセージの MD5 チェックサム。
CamelAwsSqsMessageId	**文字列**	Amazon SQS メッセージ ID。
CamelAwsSqsReceiptHandle	**文字列**	Amazon SQS メッセージ受信ハンドル。
CamelAwsSqsAttributes	**Map<String, String>**	Amazon SQS メッセージ属性。

33.4.3. 高度な AmazonSQS 設定

Camel アプリケーションがファイアウォールの内側で実行されている場合や、AmazonSQS インスタンス設定をより詳細に制御する必要がある場合は、独自のインスタンスを作成できます。

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey",
"mySecretKey");

ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
```

```
clientConfiguration.setProxyPort(8080);

AmazonSQS client = new AmazonSQSClient(awsCredentials, clientConfiguration);

registry.bind("client", client);
```

そして、Camel aws-sqs コンポーネント設定で参照します。

```
from("aws-sqs://MyQueue?amazonSQSClient=#client&delay=5000&maxMessagesPerPoll=5")
.to("mock:result");
```

33.5. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-aws</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.6 以上）に置き換える必要があります。

33.6. JMS 形式のセレクター

SQS はセレクターを許可しませんが、Camel Filter EIP を使用して適切な visibilityTimeout を設定するとこれを効果的に実行できます。SQS がメッセージをディスパッチすると、表示タイムアウトまで待機し、DeleteMessage が受信されない限り、メッセージを別のコンシューマーにディスパッチします。デフォルトでは、ルートが失敗で終了していない限り、Camel は常にルートの末尾で DeleteMessage を送信します。適切なフィルターを実行し、ルートが正常に完了した場合でも DeleteMessage を送信しないようにするには、Filter を使用します。

```
from("aws-sqs://MyQueue?
amazonSQSClient=#client&defaultVisibilityTimeout=5000&deleteIfFiltered=false")
.filter("${header.login} == true")
.to("mock:result");
```

上記のコードでは、エクスチェンジに適切なヘッダーがない場合は、フィルター AND 経由でも SQS キューから削除されません。5000 ミリ秒後、メッセージは他のコンシューマーに表示されます。

33.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- AWS コンポーネント

第34章 AWS SIMPLE WORKFLOW コンポーネント

Camel バージョン 2.13 から利用可能

Simple Workflow コンポーネントは、 Amazon の Simple Workflow サービスからのワークフローの管理をサポートします。

前提条件

有効な Amazon Web Services 開発者アカウントが必要で、Amazon Simple Workflow を使用するにはサインアップする必要があります。詳細は、「 Amazon Simple Workflow 」を参照してください。

34.1. URI 形式

```
aws-swf://<workflow|activity>[?options]
```

URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

34.2. URI オプション

AWS Simple Workflow コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	AWS SWF デフォルト設定		SWFConfiguration
accessKey (common)	Amazon AWS Access Key。		文字列
secretKey (common)	Amazon AWS Secret Key。		文字列
リージョン （共通）	Amazon AWS リージョン		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

AWS Simple Workflow エンドポイントは、URI 構文を使用して設定します。

```
aws-swf:type
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

34.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
type	**必要な** アクティビティーまたはワークフロー		文字列

34.2.2. クエリーパラメーター（30 パラメーター）：

Name	説明	デフォルト	Type
amazonSWClient (common)	指定の AmazonSimpleWorkflowClient をクライアントとして使用します。		AmazonSimpleWorkflow Client
dataConverter (common)	データのシリアライズ/デシリアライズに使用する com.amazonaws.services.simpleworkflow.flow.DataConverter のインスタンス。		DataConverter
domainName (common)	使用するワークフロードメイン。		文字列
eventName (common)	使用するワークフローまたはアクティビティーイベント名。		文字列
リージョン （共通）	Amazon AWS リージョン		文字列

Name	説明	デフォルト	Type
バージョン (common)	使用するワークフローまたはアクティビティーイベントのバージョン。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
ClientConfiguration パラメーター（詳細）	マップのキー/値を使用して ClientConfiguration を設定します。		マップ
startWorkflowOptions Parameters (advanced)	Map のキー/値を使用して StartWorkflowOptions を設定します。		マップ
sWClientParameters (advanced)	マップのキー/値を使用して AmazonSimpleWorkflowClient を設定します。		マップ
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
activityList (activity)	アクティビティーを消費するリスト名。		文字列
activitySchedulingOptions (activity)	アクティビティースケジューリングオプション		ActivityScheduling オプション
activityThreadPoolSize (activity)	アクティビティーに対するワークプールの最大スレッド数。	100	int

Name	説明	デフォルト	Type
activityTypeExecution Options (activity)	アクティビティー実行オプション		ActivityTypeExecution オプション
activityTypeRegistration Options (activity)	アクティビティー登録オプション		ActivityTypeRegistrationOptions
childPolicy （ワークフロー）	ワークフローの終了時に子ワークフローで使用するポリシー。		文字列
executionStartToClose Timeout (workflow)	実行開始を設定してタイムアウトを閉じます。	3600	文字列
操作 （ワークフロー）	ワークフロー操作	開始	文字列
SignalName （ワークフロー）	ワークフローに送信するシグナルの名前。		文字列
stateResultType (workflow)	ワークフローの状態がクエリーされる結果のタイプ。		文字列
taskStartToCloseTimeout (workflow)	タスク開始を設定してタイムアウトを閉じます。	600	文字列
terminationDetails (workflow)	ワークフローの終了詳細。		文字列
terminationReason (workflow)	ワークフローを終了する理由。		文字列
workflowList (workflow)	ワークフローを消費する一覧名。		文字列
workflowTypeRegistration Options (workflow)	ワークフロー登録オプション		WorkflowTypeRegistrationOptions
accessKey (security)	Amazon AWS Access Key。		文字列
secretKey (security)	Amazon AWS Secret Key。		文字列

必要な SWF コンポーネントオプション

Amazon の Simple Workflow Service にアクセスするには、レジストリーまたは accessKey で amazonSWClient を提供する必要があります。

34.3. 用途

34.3.1. SWF Workflow Producer によって評価されるメッセージヘッダー

ワークフロープロデューサーを使用すると、ワークフローと対話できます。新しいワークフロー実行の開始、その状態のクエリー、実行中のワークフローへのシグナルの送信、または終了（終了）してキャンセルすることができます。

ヘッダー	Type	説明
CamelSWFOperation	**文字列**	ワークフローで実行する操作。サポートされる操作：SIGNAL、CANCEL、TERMINATE、GET_STATE、START、DESCRIBE、GET_HISTORY
CamelSWFWorkflowId	**文字列**	使用するワークフロー ID。
CamelAwsDdbKeyCamelSWFRunId	**文字列**	使用する worfklow 実行 ID。
CamelSWFStateResultType	**文字列**	ワークフローの状態がクエリーされる結果のタイプ。
CamelSWFEventName	**文字列**	使用するワークフローまたはアクティビティーイベント名。

ヘッダー	Type	説明
CamelSWFVersion	**文字列**	使用するワークフローまたはアクティビティーイベントのバージョン。
CamelSWFReason	**文字列**	ワークフローを終了する理由。
CamelSWFDetails	**文字列**	ワークフローの終了詳細。
CamelSWFChildPolicy	**文字列**	ワークフローの終了時に子ワークフローで使用するポリシー。

34.3.2. SWF Workflow Producer によって設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelSWFWorkflowId	**文字列**	使用または新たに生成される worfklow ID。
CamelAwsDdbKey CamelSWFRunId	**文字列**	使用または生成される worfklow 実行 ID。

34.3.3. SWF ワークフローコンシューマーによって設定されるメッセージヘッダー

ワークフローコンシューマーは、ワークフローロジックを表します。開始時に、ワークフローのデシジョンタスクのポーリングを開始して処理します。デシジョンタスクを処理する他に、ワークフローコンシューマールートもシグナル（ワークフロープロデューサーから送信）または状態クエリーを受信します。ワークフローコンシューマーの主な目的は、アクティビティープロデューサーを使用して実行するアクティビティータスクをスケジュールすることです。実際のアクティビティータスクは、ワークフローコンシューマーによって開始したスレッドからのみスケジュールできます。

ヘッダー	Type	説明
CamelSWFAction	**文字列**	現在のイベントのタイプを示します。CamelSWFActionExecute、CamelSWFSignalReceivedAction、CamelSWFGetStateAction。
CamelSWFWorkflowReplaying	**boolean**	現在のデシジョンタスクが再生されているかどうかを示します。
CamelSWFWorkflowStartTime	**Long**	このデシジョンタスクの開始イベントの時間。

34.3.4. SWF Activity Producer によって設定されたメッセージヘッダー

アクティビティープロデューサーは、アクティビティータスクのスケジューリングを可能にします。アクティビティープロデューサーは、ワークフローコンシューマーによって開始したスレッドからのみ使用でき、ワークフローコンシューマーによって開始する同期エクスチェンジを処理できます。

ヘッダー	Type	説明
CamelSWFEventName	**文字列**	スケジュールするアクティビティー名。
CamelSWFVersion	**文字列**	スケジュールするアクティビティーバージョン。

34.3.5. SWF アクティビティーコンシューマーによって設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelSWFTaskToken	**文字列**	手動で完了したタスクの完了についてのタスク補完を報告するのに必要なタスクトークン。

34.3.6. 高度な amazonSWClient 設定

AmazonSimpleWorkflowClient インスタンス設定に対する制御を強化する必要がある場合は、独自のインスタンスを作成して URI から参照することができます。

#client はレジストリーの AmazonSimpleWorkflowClient を参照します。

たとえば、Camel アプリケーションがファイアウォールの内側で実行されている場合は、以下のようになります。

```
AWSCredentials awsCredentials = new BasicAWSCredentials("myAccessKey",
"mySecretKey");
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setProxyHost("http://myProxyHost");
clientConfiguration.setProxyPort(8080);

AmazonSimpleWorkflowClient client = new AmazonSimpleWorkflowClient(awsCredentials,
clientConfiguration);

registry.bind("client", client);
```

34.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws</artifactId>
    <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel の実際のバージョン（2.13 以降）に置き換える必要があります。

34.5. 関連項目

- Configuring Camel (Camel の設定)

- コンポーネント
- エンドポイント
- はじめに

AWS コンポーネント

第35章 AWS XRAY コンポーネント

Camel 2.21 で利用可能

camel-aws-xray コンポーネントは、AWS XRay を使用した Camel メッセージのトレースとタイミングに使用されます。

イベント（サブセグメント）は、Camel に送信される受信および送信メッセージに対してキャプチャーされます。

35.1. 依存関係

AWS XRay サポートを Camel に含めるには、Camel 関連の AWS XRay 関連のクラスが含まれるアーカイブをプロジェクトに追加する必要があります。さらに、AWS XRay ライブラリーも利用できるようにする必要があります。

AWS XRay と Camel の両方を含めるには、依存関係に以下の Maven インポートを使用します。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-xray-recorder-sdk-bom</artifactId>
      <version>1.3.1</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-aws-xray</artifactId>
    </dependency>

    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-xray-recorder-sdk-core</artifactId>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-xray-recorder-sdk-aws-sdk</artifactId>
    </dependency>
<dependencies>
```

-

35.2. 設定

AWS XRay トレーサーの設定プロパティーは次のとおりです。

オプション	デフォルト	説明
addExcludePatterns		パターンに一致する Camel メッセージのトレースを無効にする除外パターンを設定します。コンテンツは Set<String> で、キーは routeId に一致するパターンです。このパターンは Intercept のルールを使用します。
setTracingStrategy	NoopTracingStrategy	**BeanDefinition** や **ProcessDefinition** などの呼び出されたプロセッサー定義を追跡するために、カスタム Camel **InterceptStrategy** を提供できます。**TraceAnnotatedTracingStrategy** は、クラスレベルで **@XRayTrace** アノテーションが含まれる **.bean(...)** または **.process(...)** を介して呼び出されるクラスを追跡します。

現在、AWS XRay トレーサーを Camel アプリケーション用に分散トレーシングを提供するために設定できる方法は 1 つだけです。

35.2.1. explicit

camel-aws-xray コンポーネントと、AWS XRay トレーサーに関連付けられた特定の依存関係が含まれます。

AWS XRay サポートを明示的に設定するには、XRayTracer をインスタンス化し、Camel コンテキストを初期化します。オプションで トレーサー を指定したり、レジストリー または ServiceLoader を使用して暗黙的に検出することもできます。

```
XRayTracer xrayTracer = new XRayTracer();
// By default it uses a NoopTracingStrategy, but you can override it with a specific
InterceptStrategy implementation.
xrayTracer.setTracingStrategy(...);
// And then initialize the context
xrayTracer.init(camelContext);
```

XML で XRayTracer を使用するには、AWS XRay トレーサー Bean を定義するだけです。Camel は自動的に検出して使用します。

```
<bean id="tracingStrategy" class="..."/>
<bean id="aws-xray-tracer" class="org.apache.camel.component.aws.xray.XRayTracer" />
```

```
    <property name="tracer" ref="tracingStrategy"/>
  </bean>
```

デフォルトの **NoopTracingStrategy** の場合、エクスチェンジの作成および削除のみが追跡されますが、特定の **Bean** または **EIP** パターンの呼び出しは追跡されません。

35.2.2. 包括的なルート実行の追跡

複数のルート間でのエクスチェンジの実行を追跡するために、エクスチェンジの作成時に一意のトレース **ID** が生成され、対応する値がまだ利用できない場合にはヘッダーに保存されます。処理されたエクスチェンジの一貫性ビューを維持するために、このトレース **ID** は新しいエクスチェンジにコピーされます。

AWS XRay トレースはスレッドベースで動作するため、現在のサブセグメントは新しいスレッドにコピーされ、**AWS XRay** ドキュメント で説明されているように設定する必要があります。そのため、**Camel AWS XRay** コンポーネントは、渡された **AWS XRay Entity** を新しいスレッドに設定し、実行されたルートとは関係のない新しいセグメントを公開するのではなく、コンポーネントが使用する追加のヘッダーフィールドを提供します。

コンポーネントは、エクスチェンジのヘッダーにある以下の定数を使用します。

ヘッダー	説明
Camel-AWS-XRay-Trace-ID	呼び出されたルートの包括的なビューを提供する AWS XRay **TraceID** オブジェクトへの参照が含まれます。
Camel-AWS-XRay-Trace-Entity	新しいスレッドにコピーされる実際の AWS XRay **Segment** または **Subsegment** への参照が含まれます。このヘッダーは、新しいスレッドが生成され、実行されたタスクを、関連する新しいセグメントを作成するのではなく、実行中のルートの一部として公開する必要があります。

AWS XRay Entity （例： **Segment** および **Subsegment**）はシリアライズできないので、他の **JVM** プロセスに渡さないことに注意してください。

35.3. 例

このプロジェクトで付随するテスト内で **AWS XRay** トレースを設定する方法を実証する例があります。

第36章 WINDOWS AZURE SERVICES の CAMEL コンポーネント

Windows Azure Services の Camel コンポーネントは、Camel から Azure サービスへの接続を提供します。

Azure Service	Camel コンポーネント	Camel バージョン	コンポーネントの説明
Storage Blob サービス	Azure-Blob	2.9.0	Blob の保存および取得をサポートします。
ストレージキューサービス	Azure-Queue	2.9.0	キューへのメッセージの保存および取得をサポートします。

第37章 AZURE STORAGE BLOB SERVICE コンポーネント

Camel バージョン 2.19 から利用可能

Azure Blob コンポーネントは、Azure Storage Blob サービス/サービスへの Blob の保存および取得をサポートします。

前提条件

有効な Windows Azure Storage アカウントが必要です。詳細は、Azure ドキュメントポータル を参照してください。

37.1. URI 形式

```
azure-blob://accountName/containerName[/blobName][?options]
```

ほとんどの場合、BlobName が存在しない場合には Blob が作成されます。
URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

たとえば、camelazure ストレージアカウントの container1 にあるパブリックブロック Blob ブロックから Blob コンテンツをダウンロードするには、以下のスニペットを使用します。

```
from("azure-blob:/camelazure/container1/blockBlob").
to("file://blobdirectory");
```

37.2. URI オプション

Azure Storage Blob Service コンポーネントにはオプションがありません。

Azure Storage Blob Service エンドポイントは URI 構文を使用して設定されます。

```
azure-blob:containerOrBlobUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

37.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
containerOrBlobUri	**必要な** コンテナーまたは Blob のコンパクトな Uri		文字列

37.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
azureBlobClient (common)	blob サービスクライアント		CloudBlob
blobOffset (common)	アップロードまたはダウンロード操作の Blob オフセットを設定します。デフォルトは 0 です。	0	Long
blobType (common)	blob タイプを設定します。'blockblob' はデフォルトです。	blockblob	BlobType
closeStreamAfterRead (common)	読み取り後にストリームを閉じるか、または開いたままにします。デフォルトは true です。	true	boolean
認証情報 （共通）	ほとんどの場合は、ストレージ認証情報を設定します。		StorageCredentials
dataLength (common)	ダウンロードまたはページブロブのアップロード操作のデータ長の設定		Long
fileDir (common)	ダウンロードした Blob が保存されるファイルディレクトリーを設定します。		文字列
publicForRead (common)	このプロパティーが有効な場合は、ストレージリソースを公開してコンテンツを読み取ることができます。この場合、認証情報を設定する必要はありません。	false	boolean
streamReadSize (common)	Blob コンテンツの読み取り時に最小の読み取りサイズをバイト単位で設定します。		int

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
blobMetadata (producer)	Blob メタデータの設定		マップ
blobPrefix (producer)	Blob の一覧表示に使用できる接頭辞の設定		文字列
closeStreamAfterWrite (producer)	書き込み後にストリームを閉じるか、または開いたままにします。デフォルトは true です。	true	boolean
操作 （プロデューサー）	Blob サービス操作のプロデューサーへのヒント	listBlobs	BlobServiceOperations
streamWriteSize (producer)	ブロックおよびページブロックを書き込むバッファーのサイズを設定します。		int
useFlatListing (producer)	フラットまたは階層の Blob リストを使用するかどうかを指定します。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

必要な Azure Storage Blob Service コンポーネントのオプション

プライベート Blob にアクセスする必要がある場合は、containerOrBlob 名および認証情報を提供する必要があります。

37.3. 用途

37.3.1. Azure Storage Blob Service プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明

37.3.2. Azure Storage Blob サービスプロデューサーで設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelFileName	**文字列**	ダウンロードした Blob コンテンツのファイル名。

37.3.3. Azure Storage Blob Service プロデューサーコンシューマーによって設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelFileName	**文字列**	ダウンロードした Blob コンテンツのファイル名。

37.3.4. Azure Blob サービス操作

すべてのブロックタイプに共通する操作

操作	説明
getBlob	Blob の内容を取得します。この操作の出力を Blob 範囲に制限することができます。
deleteBlob	Blob を削除します。

操作	説明
listBlobs	Blob を一覧表示します。

ブロック Blob 操作

操作	説明
updateBlockBlob	新しいブロック Blob を作成するブロック Blob コンテンツを配置するか、既存のブロック Blob コンテンツを上書きします。
uploadBlobBlocks	最初に Blob ブロックのシーケンスを生成し、それらを Blob にコミットして、ブロック Blob コンテンツをアップロードします。メッセージ **CommitBlockListLater** プロパティーを有効にした場合、コミットは後で **commitBlobBlockList** 操作で実行できます。後に個別のブロック Blob を更新できます。
commitBlobBlockList	**CommitBlockListLater** プロパティーを有効にして **updateBlockBlob** 操作を使用して、以前に Blob サービスにアップロードしたブロックリストに Blob ブロックのシーケンスをコミットします。
getBlobBlockList	ブロック Blob 一覧を取得します。

Blob 操作の追加

操作	説明
createAppendBlob	追加ブロックを作成します。デフォルトでは、ブロックがすでに存在する場合はリセットされません。別の方法では、メッセージ **AppendBlobCreated** プロパティーを有効にし、**updateAppendBlob** 操作を使用して add Blob を作成することもできます。
updateAppendBlob	新しいコンテンツを Blob に追加します。この操作は、BlobCreated プロパティーがまだ存在していない場合や、メッセージ **AppendBlobCreated** プロパティーを有効にしている場合には、BlobCreated も作成します。

ページブロック操作

操作	説明

操作	説明
createPageBlob	ページブロックを作成します。デフォルトでは、ブロックがすでに存在する場合はリセットされません。PageBlobCreated プロパティーを有効にし、**updatePageBlob** 操作を使用してページブロブを作成し、その内容を設定することもできます。
updatePageBlob	ページブロック（PageBlobCreated プロパティーを有効にし、同じ名前で block がすでに存在していない限り）を作成し、この Blob の内容を設定します。
resizePageBlob	ページブロブのサイズを変更します。
clearPageBlob	ページブロブをクリアします。
getPageBlobRanges	ページの blob ページ範囲を取得します。

37.3.5. Azure Blob クライアントの設定

Camel アプリケーションがファイアウォールの背後で実行されている場合や、Azure Blob クライアント設定をさらに制御する必要がある場合は、独自のインスタンスを作成できます。

```
StorageCredentials credentials = new StorageCredentialsAccountAndKey("camelazure",
"thekey");

CloudBlob client = new CloudBlob("camelazure", credentials);

registry.bind("azureBlobClient", client);
```

そして、Camel azure-blob コンポーネント設定でこれを参照します。

```
from("azure-blob:/camelazure/container1/blockBlob?azureBlobClient=#client")
.to("mock:result");
```

37.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-azure</artifactId>
```

```
    <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel（2.19.0 以降）の実際のバージョンに置き換える必要があります。

37.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- Azure コンポーネント

第38章 AZURE STORAGE QUEUE SERVICE コンポーネント

Camel バージョン 2.19 から利用可能

Azure Queue コンポーネントは、Azure Storage Queue サービスとの間でメッセージの保存および取得をサポートします。

前提条件

有効な Windows Azure Storage アカウントが必要です。詳細は、Azure ドキュメントポータル を参照してください。

38.1. URI 形式

```
azure-queue://accountName/queueName[?options]
```

キューが存在しない場合は作成されます。
URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

たとえば、camelazure ストレージアカウントのキュー messageQueue からメッセージコンテンツを取得し、以下のスニペットを使用します。

```
from("azure-queue:/camelazure/messageQueue").
to("file://queuedirectory");
```

38.2. URI オプション

Azure Storage Queue Service コンポーネントにはオプションがありません。

Azure Storage Queue Service エンドポイントは URI 構文を使用します。

```
azure-queue:containerAndQueueUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

38.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
containerAndQueueUri	**必要な** コンテナーキューのコンパクトなチェック		文字列

38.2.2. クエリーパラメーター（10 パラメーター）：

Name	説明	デフォルト	Type
azureQueueClient (common)	キューサービスクライアント		CloudQueue
認証情報 （共通）	ほとんどの場合は、ストレージ認証情報を設定します。		StorageCredentials
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
messageTimeToLive (producer)	Message Time To Live in seconds		int
messageVisibilityDelay (producer)	Message Visibility Delay in seconds		int
操作 （プロデューサー）	キューサービス操作のヒント（プロデューサーへのキュー）	listQueues	QueueServiceOperations

Name	説明	デフォルト	Type
queuePrefix (producer)	キューの一覧表示に使用できる接頭辞の設定		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

必要な Azure Storage Queue Service コンポーネントのオプション

containerAndQueue URI および認証情報を指定する必要があります。

38.3. 用途

38.3.1. Azure Storage Queue Service プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明

38.3.2. Azure Storage Queue Service プロデューサーで設定されたメッセージヘッダー

ヘッダー	Type	説明

38.3.3. Azure Storage Queue Service プロデューサーコンシューマーによって設定されたメッセージヘッダー

ヘッダー	Type	説明

38.3.4. Azure Queue Service 操作

操作	説明
listQueues	キューを一覧表示します。
createQueue	キューを作成します。
deleteQueue	キューを削除します。
addMessage	キューにメッセージを追加します。
retrieveMessage	キューからメッセージを取得します。
peekMessage	たとえば、キュー内のメッセージを表示して、メッセージが正しいキューに到達するかどうかを判断します。
updateMessage	キューのメッセージを更新します。
deleteMessage	キューのメッセージを削除します。

38.3.5. Azure Queue クライアントの設定

Camel アプリケーションがファイアウォールの内側で実行されている場合や、Azure Queue Client 設定をより詳細に制御する必要がある場合は、独自のインスタンスを作成できます。

```
StorageCredentials credentials = new StorageCredentialsAccountAndKey("camelazure",
"thekey");

CloudQueue client = new CloudQueue("camelazure", credentials);

registry.bind("azureQueueClient", client);
```

そして、Camel azure-queue コンポーネントの設定でこれを参照します。

```
from("azure-queue:/camelazure/messageQueue?azureQueueClient=#client")
.to("mock:result");
```

38.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

-

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-azure</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel（2.19.0 以降）の実際のバージョンに置き換える必要があります。

38.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- Azure コンポーネント

第39章 BARCODE DATAFORMAT

Camel バージョン 2.14 から利用可能

バーコードデータ形式は、zxing ライブラリー に基づいています。このコンポーネントの目的は、String(marshal)からバーコードイメージと、バーコードイメージ（アンマーシャリング）から String を作成することです。zxing が提供するすべての機能を自由に使用できます。

39.1. 依存関係

camel ルートでバーコードデータ形式を使用するには、このデータ形式を実装する camel-barcode の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-barcode</artifactId>
  <version>x.x.x</version>
</dependency>
```

39.2. BARCODE オプション

Barcode データフォーマットは、以下に示す 5 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
width		整数	バーコードの幅
height		整数	バーコードの高さ
imageType		文字列	バーコードのイメージタイプ（png など）
barcodeFormat		文字列	QR-Code などのバーコード形式

Name	デフォルト	Java タイプ	説明
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

39.3. JAVA DSL の使用

最初に、バーコードデータ fomat クラスを初期化する必要があります。デフォルトのコンストラクターまたはパラメーター化のいずれかを使用できます（JavaDoc を参照）。デフォルト値は以下の通りです。

パラメーター	デフォルト値
イメージタイプ (BarcodeImageType)	PNG
幅	100 px
高さ	100 px
encoding	UTF-8
バーコード形式 (BarcodeFormat)	qr-Code

```
// QR-Code default
DataFormat code = new BarcodeDataFormat();
```

zxing ヒントを使用する場合は、BarcodeDataFormat インスタンスの 'addToHintMap' メソッドを使用できます。

```
code.addToHintMap(DecodeHintType.TRY_HARDER, Boolean.true);
```

可能なヒントについては、xzing documentation を参照してください。

39.3.1. マーシャリング

```
from("direct://code")
  .marshal(code)
  .to("file://barcode_out");
```

以下を使用して、テストクラスからルートを呼び出すことができます。

```
template.sendBody("direct://code", "This is a testmessage!");
```

このイメージは、「barcode_out」フォルダーの中に見つかるはずです。

39.3.2. アンマーシャリング

アンチラーは汎用的です。アンマーシャリングには、BarcodeDataFormat インスタンスを使用できます。2 つのインスタンス（生成）QR-Code と PDF417 用インスタンスの 1 つがある場合は、どのインスタンスを使用するかは重要ではありません。

```
from("file://barcode_in?noop=true")
  .unmarshal(code) // for unmarshalling, the instance doesn't matter
  .to("mock:out");
```

上記の QR-Code イメージを 'barcode_in' フォルダーに貼り付ける場合 「は、モック内のテストメッセージ」 です。バーコードデータ形式はヘッダー変数として見つけることができます。

Name	タイプ	説明
BarcodeFormat	文字列	com.google.zxing.BarcodeFormat の値。

第40章 BASE64 DATAFORMAT

Camel バージョン 2.11 で利用可能

Base64 データフォーマットは base64 エンコーディングおよびデコードに使用されます。

40.1. オプション

Base64 dataformat は、以下に示す 4 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
lineLength	**76**	**整数**	エンコードされたデータの最大行長を指定します。デフォルトでは、76 が使用されます。
lineSeparator		**文字列**	使用する区切り文字。デフォルトで新しい行文字(CRLF)を使用します。
urlSafe	**false**	**ブール値**	'' と '/' を出力する代わりに、それぞれ '-' と '_' を出力します。urlSafe は、エンコード操作にのみ適用されます。デコードは両方のモードをシームレスに処理します。デフォルトは false です。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

Spring DSL では、以下のタグを使用してデータ形式を設定します。

```
<camelContext>
  <dataFormats>
    <!-- for a newline character (\n), use the HTML entity notation coupled with the ASCII code. -->
    <base64 lineSeparator="&#10;" id="base64withNewLine" />
    <base64 lineLength="64" id="base64withLineLength64" />
  </dataFormats>
  ...
</camelContext>
```

その後、後で参照情報を使用できます。

```
<route>
    <from uri="direct:startEncode" />
    <marshal ref="base64withLineLength64" />
    <to uri="mock:result" />
</route>
```

多くの場合、デフォルトのオプションを使用する場合、データ形式を宣言する必要はありません。この場合、以下のようにデータフォーマットをインラインで宣言できます。

40.2. マーシャリング

この例では、ファイルの内容を base64 オブジェクトにマーシャルします。

```
from("file://data.bin")
    .marshal().base64()
    .to("jms://myqueue");
```

Spring DSL の場合：

```
<from uri="file://data.bin">
<marshal>
    <base64/>
</marshal>
<to uri="jms://myqueue"/>
```

40.3. アンマーシャリング

この例では、newOrder プロセッサーによって処理される前に、JMS キューから byte[] オブジェクトにペイロードをアンマーシャリングします。

```
from("jms://queue/order")
    .unmarshal().base64()
    .process("newOrder");
```

Spring DSL の場合：

```
<from uri="jms://queue/order">
<marshal>
    <base64/>
```

```
</marshal>
<to uri="bean:newOrder"/>
```

40.4. 依存関係

Camel ルートで Base64 を使用するには、このデータ形式を実装する camel-base64 の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加できます。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-base64</artifactId>
  <version>x.x.x</version>  <!-- use the same version as your Camel core version -->
</dependency>
```

第41章 BEAN コンポーネント

Camel バージョン 1.0 で利用可能

bean: コンポーネントは Bean を Camel メッセージエクスチェンジにバインドします。

41.1. URI 形式

```
bean:beanName[?options]
```

beanID は、レジストリーで Bean を検索するために使用される任意の文字列になります。

41.2. オプション

Bean コンポーネントにはオプションがありません。

Bean エンドポイントは URI 構文を使用します。

```
bean:beanName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

41.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
beanName	**Required**: 呼び出す Bean の名前を設定します。		文字列

41.2.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
メソッド（プロデューサー）	Bean で呼び出すメソッドの名前を設定します。		文字列

Name	説明	デフォルト	Type
キャッシュ （詳細）	有効にすると、Camel は最初のレジストリールックアップの結果をキャッシュします。レジストリーの Bean がシングルトンスコープとして定義されている場合は、キャッシュを有効にすることができます。	false	boolean
multiParameterArray (advanced)	**非推奨**: メッセージボディーから渡されたパラメーターを処理する方法。true の場合は、メッセージボディーはパラメーターの配列である必要があります。注記： このオプションは Camel によって内部に使用され、エンドユーザー向けの目的ではありません。非推奨注記： このオプションは Camel によって内部的に使用され、エンドユーザー向けの予定はありません。	false	boolean
パラメーター （詳細）	Bean の追加プロパティーの設定に使用されます。		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

URI にクエリーオプションを追加するには、?option=value&option=value&...

41.3. 使用

メッセージの消費に使用されるオブジェクトインスタンスは、明示的にレジストリーに登録されている必要があります。たとえば、Spring を使用している場合は、Spring 設定 spring.xml で Bean を定義する必要があります。または Spring を使用しない場合は、JNDI で Bean を登録する必要があります。

Error formatting macro: snippet: java.lang.IndexOutOfBoundsException: Index: 20, Size: 20

エンドポイントが登録されたら、エクスチェンジの処理に使用する Camel ルートを構築できます。

Bean: エンドポイントは、ルートへの入力として定義できません。つまり、消費できません。つまり、一部のインバウンドメッセージエンドポイントから Bean エンドポイントへ、出力としてのみルーティングできます。したがって、direct: または queue: エンドポイントを入力として使用することを検討してください。

ProxyHelper で createProxy（） メソッドを使用して、BeanExchange を生成し、エンドポイントに送信するプロキシーを作成できます。

Spring DSL を使用した同じルートの場合：

```
<route>
  <from uri="direct:hello">
  <to uri="bean:bye"/>
</route>
```

41.4. エンドポイントとしての BEAN

Camel は、エンドポイントとしての Bean の呼び出しもサポートします。ルートは以下のとおりです。

エクスチェンジが myBean Camel にルーティングされると、Bean バインディングが Bean を呼び出すことです。
Beanのソースはプレーンな POJO のみです。

Camel は Bean バインディングを使用し て、Exchange の In ボディーを String 型に変換し、Exchange Out ボディーにメソッドの出力を保存します。

41.5. JAVA DSL BEAN 構文

Java DSL には Bean コンポーネントの構文が含まれています。エンドポイント（例：("bean:beanName")）として Bean を明示的に指定する代わりに、以下の構文を使用できます。

```
// Send message to the bean endpoint
// and invoke method resolved using Bean Binding.
from("direct:start").beanRef("beanName");

// Send message to the bean endpoint
// and invoke given method.
from("direct:start").beanRef("beanName", "methodName");
```

参照名を Bean に渡す代わりに（Camel がレジストリーで検索するように）、Bean 自体を指定できます。

```
// Send message to the given bean instance.
from("direct:start").bean(new ExampleBean());
```

```
// Explicit selection of bean method to be invoked.
from("direct:start").bean(new ExampleBean(), "methodName");

// Camel will create the instance of bean and cache it for you.
from("direct:start").bean(ExampleBean.class);
```

41.6. BEAN バインディング

呼び出される Bean メソッドを選択する方法（ メソッド パラメーターで明示的に指定されていない場合）と、メッセージからパラメーター値が、Camel のさまざまな Bean 統合メカニズムすべてで使用される Bean バインディングメカニズムによって定義される方法。

41.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- Class コンポーネント
- Bean バインディング
- Bean インテグレーション

第42章 BEANIO DATAFORMAT

Camel バージョン 2.10 で利用可能

BeanIO Data Format は BeanIO を使用してフラットなペイロード（XML、CSV、コンマ区切り、固定長形式など）を処理します。

BeanIO は、フラットフォーマットから Objects(POJO)への マッピングを定義するマッピング XML ファイルを使用して設定されます。このマッピングファイルは使用することが必須です。

42.1. オプション

BeanIO データフォーマットは、以下に示す 9 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
mapping		**文字列**	BeanIO マッピングファイル。デフォルトでは、クラスパスからロードされます。file:、http:、または classpath: 接頭辞を付けて、マッピングファイルをロードする場所を示します。
streamName		**文字列**	使用するストリームの名前。
ignoreUnidentifiedRecords	**false**	**ブール値**	識別されていないレコードを無視するかどうか。
ignoreUnexpectedRecords	**false**	**ブール値**	予期しないレコードを無視するかどうか。
ignoreInvalidRecords	**false**	**ブール値**	無効なレコードを無視するかどうか。
encoding		**文字列**	使用する文字セット。デフォルトでは、JVM プラットフォームのデフォルト文字セットです。
beanReaderErrorHandlerType		**文字列**	解析中にカスタムの org.apache.camel.dataformat.beanio.BeanIOErrorHandler をエラーハンドラーとして使用します。エラーハンドラーの完全修飾クラス名を設定します。カスタムエラーハンドラーを使用する場合は ignoreUnidentifiedRecords、ignoreUnexpectedRecords、および ignoreInvalidRecords オプションが使用されていない可能性があることに注意してください。

Name	デフォルト	Java タイプ	説明
unmarshalSingleObject	**false**	**ブール値**	このオプションは、オブジェクトの一覧としてアンマーシャリングするか、1つのオブジェクトとしてのみアンマーシャリングするかどうかを制御します。前者 はデフォルトのモードで、Beanio が Camel メッセージを単一の POJO Bean にマップする特別なユースケースのみを目的としています。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

42.2. 用途

マッピングファイルの例は次のとおりです。

42.2.1. Java DSL の使用

BeanIODataFormat を使用するには、マッピングファイルでデータ形式とストリームの名前を設定する必要があります。
Java DSL では、これは以下のように実行できます。streamName は「employeeFile」です。

その後、2 つのルートがあります。最初のルートは、CSV データを List<Employee> Java オブジェクトに変換することです。分割したので、モックエンドポイント
は各行のメッセージを受信します。

2 番目のルートは、List<Employee> を CSV データのストリームに変換するためのリバース操作用です。

CSV データの例を以下に示します。

42.2.2. XML DSL の使用

XML で BeanIO データフォーマットを使用するには、以下のように <beanio> XML タグを使用して設定する必要があります。ルートは上記の例と同様です。

42.3. 依存関係

Camel ルートで BeanIO を使用するには、このデータ形式を実装する camel-beanio に依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加できます。バージョン番号は最新の最新のリリースに置き換えてください（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-beanio</artifactId>
  <version>2.10.0</version>
</dependency>
```

第43章 BEANSTALK コンポーネント

Camel バージョン 2.15 から利用可能

Camel-beanstalk プロジェクトは、Beantalk ジョブのジョブの取得および後処理を行う Camel コンポーネントを提供します。

Beantalk ジョブのライフサイクルの詳細は、bean talk プロトコル を参照してください。

43.1. 依存関係

Maven ユーザーは以下の依存関係を pom.xmlに追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-beanstalk</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで 、${camel-version} は Camel（2.15.0 以降）の実際のバージョンに置き換える必要があります。

43.2. URI 形式

```
beanstalk://[host[:port]][/tube][?options]
```

ポートまたは ホスト と ポート の両方を省略できます。Beantalk のデフォルト値（「localhost」および 11300）に使用されます。tube を省略した場合、beantalk コンポーネントは「default」という名前に置き換えられます。

リッスンしている場合には、複数のツラブからジョブを監視する必要がある場合があります。単にプラス記号で区切ります。以下に例を示します。

```
beanstalk://localhost:11300/tube1+tube2
```

Tube 名は URL をデコードします。そのため、置き換え名に + や ? などの特殊文字が含まれている場合は、URL でエンコードするか、または RAW 構文を使用する必要があります。詳細 は、こちらを

参照してください。

この方法では、ジョブを Beantalk に書き込むときに複数のツラブを指定することはできません。

43.3. BEANSTALK オプション

Beanstalk コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
connectionSettings Factory (common)	Custom ConnectionSettingsFactory.Beanstalkd への接続を行うために使用する ConnectionSettingsFactory を指定します。特に、Beantalkd デーモンを使用しないユニットテストに便利です（モック接続設定可能）。		ConnectionSettings Factory
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Beanstalk エンドポイントは URI 構文を使用して設定されます。

```
beanstalk:connectionSettings
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

43.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
connectionSettings	接続設定 host:port/tube		文字列

43.3.2. クエリーパラメーター（26 パラメーター）：

Name	説明	デフォルト	Type
コマンド (common)	Pute は、ジョブを Beantalk に配置する手段にします。ジョブボディーは Camel メッセージボディーに指定されます。ジョブ ID は beanstalk.jobId メッセージヘッダーで返されます。delete、release、touch、または bury はメッセージヘッダー beantalk.jobId のジョブ ID が予想されます。オペレーションの結果は beanstalk.result メッセージヘッダーキックで返されるため、ジョブの数がメッセージボディーで開始されることを想定し、メッセージヘッダー beantalk.result で実際に起動されたジョブの数を返します。		BeanstalkCommand
jobDelay （共通）	ジョブの遅延（秒単位）。	0	int
JobPriority (common)	ジョブの優先度。（0 は最も高い Beantalk プロトコルです）	1000	Long
jobTimeToRun (common)	数秒で実行するジョブ時間（0 の場合、beanstalkd デーモンが自動的に 1 に送出します。beantalk プロトコル を参照してください）。	60	int
awaitJob (consumer)	ジョブが完了するまで待ってから Beantalk からのジョブを認証するかどうか	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
onFailure (consumer)	処理の失敗時に使用するコマンド。		BeanstalkCommand
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
useBlockIO (consumer)	blockIO を使用するかどうか。	true	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean

Name	説明	デフォルト	Type
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

プロデューサーの動作は、ジョブの処理方法を示す command パラメーターの影響を受けます。

コンシューマーは、予約後すぐにジョブを削除するか、Camel ルートがジョブを処理するまで待機します。最初のシナリオは「メッセージキュー」のようになりますが、2 つ目は「ジョブキュー」に似ています。この動作は、デフォルトで true と等しい consumer.awaitJob パラメーターにより制御されます（Beanstalkd の性質に従う）。

同期すると、コンシューマーは正常なジョブの完了時に 削除 され、失敗時に bury を呼び出します。URI に consumer.onFailure パラメーターを指定すると、どのコマンドが失敗した場合に実行する

コマンドを選択できます。bury、 delete 、または release の値を取ることができます。

JavaBeanstalkClient ライブラリーの同じパラメーターに対応するブール値パラメーター consumer.useBlockIO があります。デフォルトでは true です。

リリース を指定する場合は注意してください。失敗したジョブが同じ場所ですぐに利用可能になり、コンシューマーはこれを再度取得しようとするので注意が必要です。ただし、 jobDelay を リリース し、指定できます。

beanstalk コンシューマーは Scheduled Polling Consumer であり、コンシューマーがポーリングする頻度など、設定可能なオプションがあります。詳細は「コンシューマーのポーリング」を参照してください。

43.4. コンシューマーヘッダー

コンシューマーは多くのジョブヘッダーをエクスチェンジメッセージに保存します。

プロパティー	型	説明
beanstalk.jobId	Long	ジョブ ID:
beanstalk.tube	string	このジョブが含まれる tube の名前
beanstalk.state	string	"ready" または "delayed" または "reserved" または "buried"（"reserved" でなければなりません）
beanstalk.priority	Long	優先度の値セット
beanstalk.age	int	このジョブを作成した put コマンドからの経過時間（秒単位）
beanstalk.time-left	int	サーバーがこのジョブを準備状態にあるキューに入れるまでの残り秒数

プロパティー	型	説明
beanstalk.timeouts	int	予約中にこのジョブがタイムアウトした回数
beanstalk.releases	int	クライアントが予約からこのジョブをリリースした回数
beanstalk.buries	int	このジョブが使用された回数
beanstalk.kicks	int	このジョブの開始回数

43.5. 例

この Camel コンポーネントを使用すると、ジョブをリクエストして Beantalkd デーモンに提供することができます。シンプルなデモルートは次のようになります。

```
from("beanstalk:testTube").
  log("Processing job #${property.beanstalk.jobId} with body ${in.body}").
  process(new Processor() {
    @Override
    public void process(Exchange exchange) {
      // try to make integer value out of body
      exchange.getIn().setBody( Integer.valueOf(exchange.getIn().getBody(classOf[String])) );
    }
  }).
  log("Parsed job #${property.beanstalk.jobId} to body ${in.body}");
```

```
from("timer:dig?period=30seconds").
  setBody(constant(10)).log("Kick ${in.body} buried/delayed tasks").
  to("beanstalk:testTube?command=kick");
```

最初のルートでは、「testTube」で新規ジョブをリッスンしています。これらが到着すると、メッセージボディーから整数値を解析しようとしています。正常に実行された場合、これをログに記録します。これにより、エクスチェンジの完了時に Camel コンポーネントにより、このジョブが自動的に Beantalk から 削除 されます。ただし、ジョブデータを解析できない場合、エクスチェンジが失敗し、Camel コンポーネントがデフォルトで検出され、後で処理するか、失敗したジョブを手動で検査することができます。

2 つ目のルートは、定期的に Beantalk を要求 して 10 個のジョブを開始し、または遅延状態を通常のキューに入れます。

43.6. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第44章 BEAN VALIDATOR コンポーネント

Camel バージョン 2.3 の時点で利用可能

Validator コンポーネントは、Java Bean Validation API(JSR 303)を使用してメッセージボディーの Bean 検証を実行します。Camel は Hibernate Validator のリファレンス実装を使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-bean-validator</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

44.1. URI 形式

```
bean-validator:label[?options]
```

または

```
bean-validator://label[?options]
```

ここで、label はエンドポイントを記述する任意のテキスト値になります。
URI にクエリーオプションを追加するには、?option=value&option=value&... を使用します。

44.2. URI オプション

Bean Validator コンポーネントにはオプションがありません。

Bean Validator エンドポイントは URI 構文を使用します。

```
bean-validator:label
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

44.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
ラベル	**必要な** 場所（label がエンドポイントを記述する任意のテキスト値）		文字列

44.2.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
constraintValidatorFactory (producer)	カスタムの ConstraintValidatorFactory を使用する場合		ConstraintValidator Factory
グループ （プロデューサー）	カスタム検証グループを使用するには、以下を実行します。	javax.validation.groups.Default	文字列
messageInterpolator (producer)	カスタム MessageInterpolator の使用		MessageInterpolator
traversableResolver (producer)	カスタムの TraversableResolver を使用する場合		TraversableResolver
validationProviderResolver (producer)	カスタムの ValidationProviderResolver を使用するには、以下を行います。		ValidationProvider Resolver
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

44.3. OSGI デプロイメント

OSGi 環境で Hibernate Validator を使用するには、org.apache.camel.component.bean.validator.Hibernate ValidationProviderResolver と同様に専用の ValidationProviderResolver 実装を使用します。以下のスニペットは、このアプローチを示しています。Camel 2.13.0 以降の HibernateValidationProviderResolver を使用することができることに注意してください。

Using HibernateValidationProviderResolver

```
from("direct:test").
  to("bean-validator://ValidationProviderResolverTest?
validationProviderResolver=#myValidationProviderResolver");

...

<bean id="myValidationProviderResolver"
class="org.apache.camel.component.bean.validator.HibernateValidationProviderResolver"/>
```

カスタム ValidationProviderResolver が定義されておらず、バリデーターコンポーネントが OSGi 環境にデプロイされている場合、 HibernateValidationProviderResolver は自動的に使用されます。

44.4. 例

以下のアノテーションを持つ java Bean があると仮定します。

Car.java

```
public class Car {

    @NotNull
    private String manufacturer;

    @NotNull
    @Size(min = 5, max = 14, groups = OptionalChecks.class)
    private String licensePlate;

    // getter and setter
}
```

カスタム検証グループのインターフェース定義

OptionalChecks.java

```
public interface OptionalChecks {
}
```

以下の Camel ルートでは、属性製造元および licensePlate 属性の @NotNull 制約のみが検証されます（Camel はデフォルトのグループ javax.validation.groups.Defaultを使用します）。

```
from("direct:start")
.to("bean-validator://x")
.to("mock:end")
```

OptionalChecks グループから制約を確認する場合は、以下のようなルートを定義する必要があります。

```
from("direct:start")
.to("bean-validator://x?group=OptionalChecks")
.to("mock:end")
```

両方のグループから制約を確認する場合は、最初に新しいインターフェースを定義する必要があります。

AllChecks.java

```
@GroupSequence({Default.class, OptionalChecks.class})
public interface AllChecks {
}
```

ルート定義は以下のようになります。

```
from("direct:start")
.to("bean-validator://x?group=AllChecks")
.to("mock:end")
```

また、独自のメッセージインターミネーター、トラバーサルなリゾルバー、および制約バリデーターファクトリーを提供する必要がある場合は、以下のようなルートを作成する必要があります。

```
<bean id="myMessageInterpolator" class="my.ConstraintValidatorFactory" />
<bean id="myTraversableResolver" class="my.TraversableResolver" />
<bean id="myConstraintValidatorFactory" class="my.ConstraintValidatorFactory" />

from("direct:start")
.to("bean-validator://x?group=AllChecks&messageInterpolator=#myMessageInterpolator
&traversableResolver=#myTraversableResolver&constraintValidatorFactory=#myConstraintVa
lidatorFactory")
.to("mock:end")
```

また、制約を XML として記述し、Java アノテーションとして記述することもできます。この場合は、以下のような META-INF/validation.xml ファイルを指定する必要があります。

validation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<validation-config
    xmlns="http://jboss.org/xml/ns/javax/validation/configuration"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/configuration">
    <default-provider>org.hibernate.validator.HibernateValidator</default-provider>
    <message-
interpolator>org.hibernate.validator.engine.ResourceBundleMessageInterpolator</message-
interpolator>
    <traversable-
resolver>org.hibernate.validator.engine.resolver.DefaultTraversableResolver</traversable-
resolver>
    <constraint-validator-
factory>org.hibernate.validator.engine.ConstraintValidatorFactoryImpl</constraint-validator-
factory>

    <constraint-mapping>/constraints-car.xml</constraint-mapping>
</validation-config>
```

constraints-car.xml ファイル

constraints-car.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<constraint-mappings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/mapping validation-mapping-
1.0.xsd"
    xmlns="http://jboss.org/xml/ns/javax/validation/mapping">
    <default-package>org.apache.camel.component.bean.validator</default-package>

    <bean class="CarWithoutAnnotations" ignore-annotations="true">
        <field name="manufacturer">
            <constraint annotation="javax.validation.constraints.NotNull" />
        </field>

        <field name="licensePlate">
            <constraint annotation="javax.validation.constraints.NotNull" />

            <constraint annotation="javax.validation.constraints.Size">
                <groups>
                    <value>org.apache.camel.component.bean.validator.OptionalChecks</value>
                </groups>
                <element name="min">5</element>
                <element name="max">14</element>
            </constraint>
```

```
        </field>
    </bean>
</constraint-mappings>
```

以下は、OrderedChecks を https://github.com/apache/camel/blob/master/components/camel-bean-validator/src/test/java/org/apache/camel/component/bean/validator/OrderedChecks.java できるルート定義の XML 構文です。

ボディーには検証するクラスのインスタンスが含まれる必要があることに注意してください。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="direct:start"/>
            <to uri="bean-validator://x?
group=org.apache.camel.component.bean.validator.OrderedChecks"/>
        </route>
    </camelContext>
</beans>
```

44.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第45章 バインディングコンポーネント（非推奨）

Camel バージョン 2.11 で利用可能

Camel では、バインディング は、Data Format、Content Enricher、または検証ステップなどのコントラクトでエンドポイントをラップする方法です。バインディングは完全に任意で、任意の Camel エンドポイントで使用することができます。

バインディングは、SwitchYard プロジェクト の作業により、Camel などのさまざまなテクノロジーにサービス契約を追加します。しかし、Camel を SCA でラップする SwitchYard アプローチではなく、Camel バインディング は Camel フレームワーク自体内のコントラクトでラップする方法を提供します。そのため、Camel ルート内で簡単に使用できます。

45.1. オプション

Binding コンポーネントにはオプションがありません。

Binding エンドポイントは URI 構文を使用して設定します。

```
binding:bindingName:delegateUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

45.1.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
bindingName	Camel レジストリーで検索するバインディングの名前。		文字列
delegateUri	delegate エンドポイントに **必要な** Uri。		文字列

45.1.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

45.2. バインディングの使用

バインディングは現在、コントラクトを定義する Bean です（Camel DSL にバインディングを追加することを期待しています）。

バインドされたエンドポイントを定義する方法はいくつかあります（バインディングでバインドされたエンドポイントなど）。

45.3. バインディング URI の使用

*binding:nameOfBinding: を使用してエンドポイント URI のプレフィックスを付けることができます。**nameOfBinding** はレジストリーの Binding Bean の名前です。*

```
from("binding:jaxb:activemq:myQueue").to("binding:jaxb:activemq:anotherQueue")
```

ここでは、「jaxb」バインディングを使用します。たとえば、JAXB Data Format を使用してメッセージをマーシャリングおよびアンマーシャリングします。

45.4. BINDINGCOMPONENT の使用

ディペンデンシーインジェクション（依存性の注入）によってレジストリーに設定できるコンポーネントがあり、すでに一部のバインディングにバインドされているエンドポイントを作成できます。

たとえば、以下のようなコードを使用して、レジストリーに「jsonmq」という新しいコンポーネントを登録しているとします。

```
JacksonDataFormat format = new JacksonDataFormat(MyBean.class);
context.bind("jsonmq", new BindingComponent(new DataFormatBinding(format),
"activemq:foo."));
```

エンドポイントは他のエンドポイントであるかのように使用できます。

```
from("jsonmq:myQueue").to("jsonmq:anotherQueue")
```

これは、キューの「foo.myQueue」と「foo.anotherQueue」を使用し、指定の Jackson Data Format を使用してキューのオンとオフを使用します。

45.5. バインディングを使用するタイミング

1 つのルートでエンドポイントを 1 度だけ使用すると、バインディングは 'raw' エンドポイントを直接使用して、通常通りに camel ルートで明示的なマーシャリングおよび検証を使用するよりも複雑で、複数の作業になることがあります。

ただし、バインディングは、多くのルートを 1 つにまとめる場合や、入出力エンドポイントを設定する「テンプレート」として単一のルートを使用する場合に役に立ちます。その後、バインディングはコントラクトとエンドポイントをラッピングする優れた方法を提供します。

バインディングには、同じバインディングを使用するエンドポイントが多数使用する場合が挙げられます。ただし、常に特定のデータ形式または検証ルールを示すのではなく、BindingComponent を使用してエンドポイントを任意のバインディングでラップすることができます。

そのため、バインディングは実際には構成ツールであり、意味のある場合にのみ使用します。ルートやエンドポイントが多数ない限り、さらに複雑ではない可能性があります。

第46章 BINDY DATAFORMAT

Camel バージョン 2.0 で利用可能

このコンポーネントの目的は、非構造化データ（またはより正確な非 XML データである）アノテーションで定義されたバインディングマッピングを持つ Java Bean の解析/バインディングを許可することです。Bindy を使用すると、以下のようなソースからデータをバインドできます。

- *CSV レコード*
- *固定長レコード*
- *Reseller メッセージ*
- *またはほぼすべての構造化されていないデータ*

1 つまたは多数のPlain Old Java Object(POJO)。バインドは、java プロパティーのタイプに従ってデータを変換します。POJO は、場合によっては利用可能な 1 対多の関係とリンクすることができます。さらに、Date、Double、Float、Integer、Short、Long、および BigDecimal などのデータ型の場合、プロパティーのフォーマット時に適用するパターンを指定できます。

BigDecimal 番号では、精度と小数点またはグループ化セパレーターを定義することもできます。

Type	形式タイプ	パターンの例	リンク
Date	DateFormat	**dd-MM-yyyy**	http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html
10 進数*	Decimalformat	**..##**	http://java.sun.com/j2se/1.5.0/docs/api/java/text/DecimalFormat.html

10 進数 = Double、Integer、Float、Short、Long*

Format supported

この最初のリリースでは、コンマ区切りの値フィールドとキーと値のペアのフィールドのみをサポートしています（例： FIX messages）。

camel-bindy と連携するには、まずパッケージにモデルを定義する必要があります（例： com.acme.model）。また、各モデルクラス（order、Client、Instrument、... など）で、必要なアノテーション（ここで説明）を Class またはフィールドに追加します。

Multiple models

複数のモデルを使用する場合は、予測不可能な結果を防ぐために、各モデルを独自のパッケージに配置する必要があります。

Camel 2.16 以降では、パッケージ名ではなくクラス名を使用してバインドを設定するため、同じパッケージに複数のモデルを安全に設定できるため、これは当てはまりなくなりました。

46.1. オプション

Bindy データフォーマットは、以下に示す 5 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
type		**Bindy Type**	csv、fixed、または key value ペアモードを使用するかどうか。デフォルト値は、選択した dataformat に応じて Csv または KeyValue です。
classType		**文字列**	使用するモデルクラスの名前。
locale		**文字列**	使用するデフォルトのロケールを設定します（例： ユニットの状態）。JVM プラットフォームのデフォルトロケールを使用するには、default という名前を使用します。
unwrapSingleInstance	**true**	**ブール値**	アンマーシャリングは、単一のインスタンスをアンラップし、java.util.List でラップせずに返します。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

46.2. アノテーション

作成されたアノテーションにより、モデルの異なる概念を POJO に以下のようにマッピングできます。

- レコードのタイプ（csv、key value pair（例： FIX message）、固定長...）
- リンク（別のオブジェクトのリンクオブジェクト）
- DataField とそのプロパティー（int、type、...）
- KeyValuePairField(key = value format like we have in FIX personal messages)）
- セクション（ヘッダー、ボディー、およびフッターのセクションを特定するため）
- OneToMany,
- BindyConverter (since 2.18.0),
- FormatFactories（2.18.0 以降）

本セクションでは、これらを説明します。

46.3. 1.CSVRECORD

CsvRecord アノテーションは、モデルのルートクラスを特定するために使用されます。これは CSV ファイルのレコード = 行を表し、複数の子モデルクラスにリンクできます。

アノテーション名	レコードタイプ	レベル
CsvRecord	csv	クラス

パラメーター名	type	Info
separator	string	必須 - ',' または ';' または 'anything' を指定できます。この値は正規表現として解釈されます。'|' のように、正規表現で特別な意味を持つ記号を使用したい場合（例： '|' 記号）
skipFirstLine	boolean	オプション： デフォルト値 = false - CSV ファイルの最初の行をスキップできるようにします。
crlf	string	オプション - 使用できる値 = WINDOWS、UNIX、MAC、または custom（デフォルト値）WINDOWS - 使用するキャリッジリターン文字を定義できます。前述した 3 つの値以外の値を指定すると、入力する値が CRLF 文字として使用されます。
generateHeaderColumns	boolean	オプション： デフォルト値 = false - CSV 生成のヘッダー列を生成するのに使用します。
autospanLine	boolean	**Camel 2.13/2.12.2:** optional - デフォルト値 = false - 有効にすると、最後の列が行末に自動スパンされます。たとえば、コメントの場合など、行にすべての文字を含めることができ、区切り文字文字も含まれるようになります。
isOrdered	boolean	オプション： デフォルト値 = false - CSV の生成時にフィールドの順番を変更できるようにします。
引用符	文字列	**camel 2.8.3/2.9: オプション：** CSV の生成時にフィールドの引用符文字を指定できます。このアノテーションはモデルの root クラスに関連付けられ、一度宣言する必要があります。
引用	boolean	*Camel 2.11:* 任意 - デフォルト値 = false - CSV の生成時にマーシャリングする際に値（およびヘッダー）を引用符で囲む必要があるかどうかを示します。
endWithLineBreak	boolean	**Camel 2.21:** optional - default value = true - CSV 生成されたファイルが改行で終わる必要があるかどうかを示します。

case 1 : separator = ','

CSV レコードのフィールドを分離するために使用される区切り文字は ',' です。

```
10, J, Pauline, M, XD12345678, Fortis Dynamic 15/15, 2500,
USD,08-01-2009
```

```
@CsvRecord( separator = "," )
public Class Order {

}
```

case 2 : separator = ';'

前のケースと比較すると、ここでのセパレーターは ',' ではなく ';' になります。

10; J; Pauline; M; XD12345678; Fortis Dynamic 15/15; 2500; USD; 08-01-2009

```
@CsvRecord( separator = ";" )
public Class Order {

}
```

case 3 : separator = '|'

前のケースと比較すると、ここでの区切り文字は ';' ではなく '|' です。

```
10| J| Pauline| M| XD12345678| Fortis Dynamic 15/15| 2500| USD|
08-01-2009
```

```
@CsvRecord( separator = "\\|" )
public Class Order {

}
```

case 4 : separator = '\",\"'

Camel 2.8.2 以前に適用されます。

CSV レコードの解析されるフィールドに ',' または ';' が含まれ、これはセパレーターとしても使用されます。
は、このケースの処理方法を camel バインドy に伝えるために別のストラテジーを見つけました。コンマで区切られたデータを含むフィールドを定義するには、区切り文字として simple または double quotes
を使用します（例： '10'、'Street 10、NY'、'USA'、"Street 10、NY", "USA"）。
Remark：この場合、単純または二重引用符である行の最初の文字と最後の文字がバインドによって削除されます。

```
"10","J","Pauline"," M","XD12345678","Fortis Dynamic 15,15"
2500","USD","08-01-2009"
```

```
@CsvRecord( separator = "\",\"" )
public Class Order {

}
```

Camel 2.8.3/2.9 以降、またはバインドy は、レコードが一 重引用符または二重引用符で囲まれているかどうかを自動的に検出し、CSV からオブジェクトへのアンマーシャリング時にこれらの引用符を自動的に削除します。したがって、区切り文字には引用符を含め ない でくださいが、以下のように簡単に実行できます。

```
"10","J","Pauline"," M","XD12345678","Fortis Dynamic 15,15"
2500","USD","08-01-2009"
```

```
@CsvRecord( separator = "," )
public Class Order {

}
```

オブジェクトから CSV にマーシャリングし、引用符を使用する場合は、以下のように @CsvRecord の引用符属性を使用して、使用する 引用 符を指定する必要があります。

```
@CsvRecord( separator = ",", quote = "\"" )
public Class Order {

}
```

ケース5: セパレーターおよびスキップライン

この機能は、クライアントがデータフィールドの名前であるファイルの最初の行に合わせる場合に適しています。

order id、クライアント ID、名、姓、コードのインストルメント化名、数値、通貨、日付

解析プロセス中にこの最初の行をスキップする必要があることをバインドするには、属性を使用します。

```
@CsvRecord(separator = ",", skipFirstLine = true)
public Class Order {

}
```

case 6 : generateHeaderColumns

生成された CSV の最初の行で追加するには、以下のようにアノテーションで generateHeaderColumns 属性を true に設定する必要があります。

```
@CsvRecord( generateHeaderColumns = true )
public Class Order {

}
```

その結果、アンマーシャリングプロセス中にバインドすると、以下のような CSV が生成されます。

order id、クライアント ID、名、姓、コードのインストルメント化名、数値、通貨、日付

```
10, J, Pauline, M, XD12345678, Fortis Dynamic 15/15, 2500, USD,08-01-2009
```

case 7 : carriage return

camel-bindy が実行されるプラットフォームが Windows ではなく、Macintosh または Unix の場合は、以下のように crlf プロパティーを変更できます。使用できる値は、WINDOWS、UNIX、または MAC です。

```
@CsvRecord(separator = ",", crlf="MAC")
public Class Order {

}
```

さらに、何らかの理由で別の行末文字を追加する必要がある場合は、crlf パラメーターを使用してそれを指定することもできます。以下の例では、行末には改行文字を続きます。

```
@CsvRecord(separator = ",", crlf=",\n")
public Class Order {

}
```

case 8 : isOrdered

モデルからの CSV レコードの作成時に発生する順序は、解析中に使用される順序と異なる場合があります。次に、この属性 isOrdered = true を使用して、DataField アノテーションの属性 'position' と組み合わせて使用できます。

```
@CsvRecord(isOrdered = true)
public Class Order {

  @DataField(pos = 1, position = 11)
  private int orderNr;

  @DataField(pos = 2, position = 10)
  private String clientNr;

}
```

Remark : ファイルの解析に使用され、CSV の生成に位置が使用されます。

46.4. 2.リンク

リンクアノテーションを使用すると、オブジェクトをリンクできます。

アノテーション名	レコードタイプ	レベル
リンク	all	クラスおよびプロパティー

パラメーター名	type	Info
linkType	LinkType	オプション： デフォルトでは、値は LinkType.oneToOne です。そのため、説明するものではありません。

1 対 1 の関係のみが許可されます。

例： モデル Class Client が Order クラスにリンクされている場合は、以下のような Order クラスのアノテーション Link を使用します。

プロパティーリンク

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @Link
    private Client client;
}
```

Client クラスの AND:

クラスリンク

```
@Link
public class Client {

}
```

46.5. 3.DATAFIELD

DataField アノテーションは、フィールドのプロパティーを定義します。各データフィールドは、レコードの位置、タイプ（文字列、整数、日付、および任意でパターン）で識別されます。

アノテーション名	レコードタイプ	レベル
DataField	all	プロパティー

パラメーター名	type	Info
pos	int	必須： フィールドの **入力** 位置。1 から ... から始まる数字の数字： position パラメーターを参照してください。
pattern	string	オプション： デフォルト値 = "" - は、10 進数、日付、形式に使用されます。
長さ	int	オプション： 固定長形式のフィールドの長さを表します。
精度	int	オプション： 16 進数がフォーマット/解析されるときに使用される精度を表します。
pattern	string	任意： デフォルト値 = "" - は、データをフォーマット/検証するために Java フォーマッター（例： SimpleDateFormat）で使用されます。パターンを使用する場合には、バインドデータフォーマットにロケールを設定することが推奨されます。「us」などの既知のロケールに設定するか、プラットフォームのデフォルトロケールを使用するには「default」を使用します。「default」には Camel 2.14/2.13.3/2.12.5 が必要であることに注意してください。
position	int	オプション： CSV 生成された（出力メッセージ）のフィールドの位置が入力位置(pos)とは異なる比較が必要な場合に使用する必要があります。pos パラメーターを参照してください。
required	boolean	optional - default value = "false"
trim	boolean	optional - default value = "false"
defaultValue	string	**camel 2.10:** optional - default value = "": それぞれの CSV フィールドが空でない/利用不可の場合にフィールドのデフォルト値を定義します。
impliedDecimalSeparator	boolean	**Camel 2.11:** optional - デフォルト値 = "false" - 指定の場所の 10 進数の暗黙がある場合に示唆します。
lengthPos	int	**Camel 2.11**: optional - このフィールドの固定長を定義する固定長レコードのデータフィールドを特定するために使用できます。

パラメーター名	type	Info
揃える	string	オプション： デフォルト値 = "R" - 固定長フィールドの右側または左側にテキストを合わせます。'R' または 'L' の値を使用します。
delimiter	string	**Camel 2.11:** optional - 固定長レコード内の variable-length フィールドの末尾を区別するために使用できます。

ケース 1: 指定可能

このパラメーター/属性は csv レコードのフィールドの位置を表します。

Position

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 5)
    private String isinCode;

}
```

この例で分かるように、位置は '1' から始まりますが、クラスの順序にある '5' に続きます。'2' から '4' までの数字は、クラスクライアントで定義されます（こちらを参照）。

別のモデルクラスで位置が継続される

```
public class Client {

    @DataField(pos = 2)
    private String clientNr;

    @DataField(pos = 3)
    private String firstName;

    @DataField(pos = 4)
    private String lastName;
}
```

- *case 2 : pattern*

  パターンにより、データの形式を補完したり、検証したりすることができます。

  パターン

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 5)
    private String isinCode;

    @DataField(name = "Name", pos = 6)
    private String instrumentName;

    @DataField(pos = 7, precision = 2)
    private BigDecimal amount;

    @DataField(pos = 8)
    private String currency;

    // pattern used during parsing or when the date is created
    @DataField(pos = 9, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

  ケース 3: 精度

  数値の 10 進数部分を定義する場合に精度が便利です。

  精度

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @Link
    private Client client;
```

```
    @DataField(pos = 5)
    private String isinCode;

    @DataField(name = "Name", pos = 6)
    private String instrumentName;

    @DataField(pos = 7, precision = 2)
    private BigDecimal amount;

    @DataField(pos = 8)
    private String currency;

    @DataField(pos = 9, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース4: 出力での位置が異なります。

position 属性は、フィールドの配置方法を CSV レコードに通知します。デフォルトでは、使用される位置は属性「pos」で定義された位置に対応します。位置が異なる場合（つまり、アンマーシャリングからマーキングからマーシャリングしていることを意味します）、'position' を使用してそれを示すことができます。

以下に例を示します。

出力の内容は異なります。

```
@CsvRecord(separator = ",", isOrdered = true)
public class Order {

    // Positions of the fields start from 1 and not from 0

    @DataField(pos = 1, position = 11)
    private int orderNr;

    @DataField(pos = 2, position = 10)
    private String clientNr;

    @DataField(pos = 3, position = 9)
    private String firstName;

    @DataField(pos = 4, position = 8)
    private String lastName;

    @DataField(pos = 5, position = 7)
    private String instrumentCode;
```

```
    @DataField(pos = 6, position = 6)
    private String instrumentNumber;
}
```

@DataField アノテーションのこの属性は、@CsvRecord のアノテーションの isOrdered = true 属性と併用する必要があります。

ケース 5: 必須

フィールドが必須の場合は、単に true に設定された属性「required」を使用します。

必須

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 2, required = true)
    private String clientNr;

    @DataField(pos = 3, required = true)
    private String firstName;

    @DataField(pos = 4, required = true)
    private String lastName;
}
```

このフィールドがレコードに存在しない場合は、パーサーにより以下の情報のエラーが発生します。

一部のフィールドが欠落（オプションまたは必須）の行になります。

case 6 : trim

フィールドに、処理される前に削除する必要のある先頭および/または末尾の空白がある場合は、true に設定した属性「trim」を使用します。

Trim

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1, trim = true)
    private int orderNr;

    @DataField(pos = 2, trim = true)
    private Integer clientNr;

    @DataField(pos = 3, required = true)
    private String firstName;

    @DataField(pos = 4)
    private String lastName;
}
```

case 7 : defaultValue

フィールドが定義されていない場合、defaultValue 属性で指定された値を使用します。

デフォルト値

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 2)
    private Integer clientNr;

    @DataField(pos = 3, required = true)
    private String firstName;

    @DataField(pos = 4, defaultValue = "Barin")
    private String lastName;
}
```

この属性は、オプションのフィールドにのみ適用されます。

46.6. 4.FIXEDLENGTHRECORD

***FixedLengthRecord** アノテーションは、モデルのルートクラスを特定するために使用されます。これは、フォーマットされたデータが含まれるファイル/メッセージの記録 = 行を表し、複数の子モデルクラスにリンクできます。この形式は、フィールドのビット固有データで、右側または左側に合わせることができます。*
データのサイズがフィールドの長さが完全に一杯でない場合は、「padd」文字を追加することができます。

アノテーション名	レコードタイプ	レベル
FixedLengthRecord	固定:	クラス

パラメーター名	type	Info
crlf	string	オプション - 使用できる値 = WINDOWS、UNIX、MAC、または custom（デフォルト値）WINDOWS - 使用するキャリッジリターン文字を定義できます。前述した 3 つの値以外の値を指定すると、入力する値が CRLF 文字として使用されます。このオプションはマーシャリング時にのみ使用されますが、eol がカスタマイズされない限り、アンマーシャリングはシステムのデフォルト JDK が提供する行区切り文字を使用します。
EOL	string	オプション： default=""（空の文字列）アンマーシャリング中に各レコードの後に行末を処理するのに使用する文字（任意： default = ""）。これは、他の行区切り文字を指定しない限り、デフォルトの JDK が提供する行区切り文字の使用に役立ちます。このオプションは、アンマーシャリング時にのみ使用されます。マーシャリングは、その他の値を提供しない限り、システムのデフォルトの区切り文字を「WINDOWS」として使用します。
paddingChar	char	mandatory - デフォルト値 = ' '
長さ	int	必須 = 固定長レコードのサイズ
hasHeader	boolean	**Camel 2.11**: オプション： このタイプのレコードの前に、ファイル/ストリームの先頭にある単一のヘッダーレコードが追加されることを示唆します。
hasFooter	boolean	**Camel 2.11**: 任意： このタイプのレコードの後に、ファイル/ストリームの最後に 1 つのフッターレコードがあることを示します。
skipHeader	boolean	**Camel 2.11** - オプション： ヘッダーレコードのマーシャリング/アンマーシャリングを省略するようにデータフォーマットを設定します。プライマリーレコード（ヘッダーやフッターなど）にこのパラメーターを設定します。

パラメーター名	type	Info
skipFooter	boolean	**Camel 2.11** - オプション： フッターレコードのマーシャリング/アンマーシャリングをスキップするようにデータフォーマットを設定します（ヘッダーやフッターではなく）。
isHeader	boolean	**Camel 2.11** - オプション： この FixedLengthRecord をヘッダーレコードとして特定
isFooter	boolean	**Camel 2.11** - オプション： この FixedLengthRecords をフッターレコードとして特定
ignoreTrailingChars	boolean	**Camel 2.11.1** - オプション： アンマーシャリング/解析時に、最後にマップされた filed 以外の文字を無視できることを示します。 このアノテーションはモデルの root クラスに関連付けられ、一度宣言する必要があります。

hasHeader/hasFooter パラメーターは isHeader/isFooter と相互に排他的です。レコードはヘッダー/フッターとプライマリー固定長レコードの両方にない場合があります。

ケース1: 簡単な固定長レコード

この簡単な例は、固定メッセージを解析/フォーマットするモデルを設計する方法を示しています。

```
10A9PaulineMISINXD12345678BUYShare2500.45USD01-08-2009
```

fixed-simple

```
@FixedLengthRecord(length=54, paddingChar=' ')
public static class Order {

    @DataField(pos = 1, length=2)
    private int orderNr;

    @DataField(pos = 3, length=2)
    private String clientNr;

    @DataField(pos = 5, length=7)
    private String firstName;

    @DataField(pos = 12, length=1, align="L")
    private String lastName;
```

```
    @DataField(pos = 13, length=4)
    private String instrumentCode;

    @DataField(pos = 17, length=10)
    private String instrumentNumber;

    @DataField(pos = 27, length=3)
    private String orderType;

    @DataField(pos = 30, length=5)
    private String instrumentType;

    @DataField(pos = 35, precision = 2, length=7)
    private BigDecimal amount;

    @DataField(pos = 42, length=3)
    private String currency;

    @DataField(pos = 45, length=10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース2 - 調整とパディングを使用した長さレコードの修正

この例は、フィールドの調整を定義する方法と、パディング文字（「ここ」）を定義する方法を示しています。

```
10A9 PaulineM ISINXD12345678BUYShare2500.45USD01-08-2009
```

fixed-padding-align

```
@FixedLengthRecord(length=60, paddingChar=' ')
public static class Order {

    @DataField(pos = 1, length=2)
    private int orderNr;

    @DataField(pos = 3, length=2)
    private String clientNr;

    @DataField(pos = 5, length=9)
    private String firstName;

    @DataField(pos = 14, length=5, align="L")   // align text to the LEFT zone of the block
    private String lastName;

    @DataField(pos = 19, length=4)
    private String instrumentCode;
```

```
    @DataField(pos = 23, length=10)
    private String instrumentNumber;

    @DataField(pos = 33, length=3)
    private String orderType;

    @DataField(pos = 36, length=5)
    private String instrumentType;

    @DataField(pos = 41, precision = 2, length=7)
    private BigDecimal amount;

    @DataField(pos = 48, length=3)
    private String currency;

    @DataField(pos = 51, length=10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース 3: フィールドパディング

レコードに定義されたデフォルトのパディングはフィールドに適用することができない場合があります。' の代わりに '0' でパディングする場合は、数字の形式があるためです。この場合は、モデルで paddingField 属性を使用してこの値を設定できます。

```
10A9 PaulineM ISINXD12345678BUYShare000002500.45USD01-08-2009
```

Fixed-padding-field

```
@FixedLengthRecord(length = 65, paddingChar = ' ')
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 3, length = 2)
    private String clientNr;

    @DataField(pos = 5, length = 9)
    private String firstName;

    @DataField(pos = 14, length = 5, align = "L")
    private String lastName;

    @DataField(pos = 19, length = 4)
    private String instrumentCode;

    @DataField(pos = 23, length = 10)
```

```
    private String instrumentNumber;

    @DataField(pos = 33, length = 3)
    private String orderType;

    @DataField(pos = 36, length = 5)
    private String instrumentType;

    @DataField(pos = 41, precision = 2, length = 12, paddingChar = '0')
    private BigDecimal amount;

    @DataField(pos = 53, length = 3)
    private String currency;

    @DataField(pos = 56, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース 4: 区切り文字を使用した長さのレコードの修正

固定長レコードでは、レコード内の内容が区切られることがあります。firstName と lastName フィールドは、以下の例のように '^' 文字で区切られています。

```
10A9Pauline^M^ISINXD12345678BUYShare000002500.45USD01-08-2009
```

fixed-delimited

```
@FixedLengthRecord()
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 2, length = 2)
    private String clientNr;

    @DataField(pos = 3, delimiter = "^")
    private String firstName;

    @DataField(pos = 4, delimiter = "^")
    private String lastName;

    @DataField(pos = 5, length = 4)
    private String instrumentCode;

    @DataField(pos = 6, length = 10)
    private String instrumentNumber;

    @DataField(pos = 7, length = 3)
```

```
    private String orderType;

    @DataField(pos = 8, length = 5)
    private String instrumentType;

    @DataField(pos = 9, precision = 2, length = 12, paddingChar = '0')
    private BigDecimal amount;

    @DataField(pos = 10, length = 3)
    private String currency;

    @DataField(pos = 11, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

Camel 2.11 の時点では、固定長レコードの「pos」値は、正確な列番号ではなく、ordinal の連続値を使用して定義することができます。

ケース5: レコード定義フィールド長のレコードレコードの修正

固定長レコードには、同じレコード内の別のフィールドの想定される長さを定義するフィールドが含まれる場合があります。以下の例では、instrumentNumber フィールド値の長さはレコードの instrumentNumberLen フィールドの値によって定義されます。

```
10A9Pauline^M^ISIN10XD12345678BUYShare000002500.45USD01-08-2009
```

fixed-delimited

```
@FixedLengthRecord()
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 2, length = 2)
    private String clientNr;

    @DataField(pos = 3, delimiter = "^")
    private String firstName;

    @DataField(pos = 4, delimiter = "^")
    private String lastName;

    @DataField(pos = 5, length = 4)
    private String instrumentCode;

    @DataField(pos = 6, length = 2, align = "R", paddingChar = '0')
```

```
    private int instrumentNumberLen;

    @DataField(pos = 7, lengthPos=6)
    private String instrumentNumber;

    @DataField(pos = 8, length = 3)
    private String orderType;

    @DataField(pos = 9, length = 5)
    private String instrumentType;

    @DataField(pos = 10, precision = 2, length = 12, paddingChar = '0')
    private BigDecimal amount;

    @DataField(pos = 11, length = 3)
    private String currency;

    @DataField(pos = 12, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

ケース6 - ヘッダーおよびフッターのある長さのレコードの修正

Bindy は、モデルの一部として設定された固定長ヘッダーとフッターレコードを検出します。アノテーションされたクラスがプライマリー @FixedLengthRecord クラスと同じパッケージ内または設定されたスキャンパッケージのいずれかに存在する場合です。以下のテキストは、ヘッダーレコードとフッターレコードで 2 つの固定長のレコードを示しています。

```
101-08-2009
10A9 PaulineM ISINXD12345678BUYShare000002500.45USD01-08-2009
10A9 RichN ISINXD12345678BUYShare000002700.45USD01-08-2009
9000000002
```

Fixed-header-and-footer-main-class

```
@FixedLengthRecord(hasHeader = true, hasFooter = true)
public class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 2, length = 2)
    private String clientNr;

    @DataField(pos = 3, length = 9)
    private String firstName;

    @DataField(pos = 4, length = 5, align = "L")
    private String lastName;
```

```
    @DataField(pos = 5, length = 4)
    private String instrumentCode;

    @DataField(pos = 6, length = 10)
    private String instrumentNumber;

    @DataField(pos = 7, length = 3)
    private String orderType;

    @DataField(pos = 8, length = 5)
    private String instrumentType;

    @DataField(pos = 9, precision = 2, length = 12, paddingChar = '0')
    private BigDecimal amount;

    @DataField(pos = 10, length = 3)
    private String currency;

    @DataField(pos = 11, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}

@FixedLengthRecord(isHeader = true)
public  class OrderHeader {
    @DataField(pos = 1, length = 1)
    private int recordType = 1;

    @DataField(pos = 2, length = 10, pattern = "dd-MM-yyyy")
    private Date recordDate;
}

@FixedLengthRecord(isFooter = true)
public class OrderFooter {

    @DataField(pos = 1, length = 1)
    private int recordType = 9;

    @DataField(pos = 2, length = 9, align = "R", paddingChar = '0')
    private int numberOfRecordsInTheFile;
}
```

Case 7: 固定長レコードを解析するときにコンテンツをスキップします。(Camel 2.11.1)

ターゲットのユースケースに必要な以上の情報が含まれる固定長レコードを提供するシステムを統合することが一般的です。この状況は、必要のないこれらのフィールドの宣言をスキップして解析する場合に役立ちます。これを許容するために、次の宣言されたフィールドの「pos」値が最後に解析されたフィールドのカーソル位置を超えても、Bindy はレコード内の次のマップされたフィールドへ進むようスキップします。（ordinal 値の代わりに）対象のフィールドに「pos」ロケーションを使用すると、Bindy は 2 つのフィールド間でコンテンツをスキップします。

同様に、フィールド以外のコンテンツは重要ではありません。この場合、@FixedLengthRecord 宣言に ignoreTrailingChars プロパティーを設定することで、Bindy に最後にマップされたフィールド以外のすべての解析をスキップするように指示できます。

```
@FixedLengthRecord(ignoreTrailingChars = true)
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 3, length = 2)
    private String clientNr;

    // any characters that appear beyond the last mapped field will be ignored

}
```

46.7. 5.メッセージ

Message アノテーションは、キーと値のペアフィールドが含まれるモデルのクラスを特定するために使用されます。このようなフォーマットは、主にサーバー交換プロトコルメッセージ(FIX)で使用されます。ただし、このアノテーションは、データがキーで識別される他の形式に使用できます。キーペアの値は、タブデリミット（ユニコード表現：\u0009）や見出しの開始（unicode 表現：\u0001）などの特殊文字で区切られます。

```
*"FIX information"*
```

FIX の詳細は、この Web サイト http://www.fixprotocol.org/ を参照してください。FIX メッセージを使用するには、モデルに Order クラスであるルートメッセージクラスにリンクされた Header および Trailer クラスが含まれている必要があります。これは必須ではありませんが、fastFix プロジェクト http://www.quickfixj.org/ に基づく Fix ゲートウェイである camel-fix と組み合わせて camel-bindy を使用する場合は非常に便利です。

アノテーション名	レコードタイプ	レベル
メッセージ	キーと値のペア	クラス

パラメーター名	type	Info
pairSeparator	string	必須 - '= または ';' または 'anything' にすることができます
keyValuePairSeparair	string	必須 - '\u0001'、'\u0009'、'#'、または 'anything' にすることができます。
crlf	string	オプション - 使用できる値 = WINDOWS、UNIX、MAC、または custom。デフォルト値 = WINDOWS - 使用するキャリッジリターン文字を定義できます。前述した 3 つの値以外の値を指定すると、入力する値が CRLF 文字として使用されます。
type	string	オプション： メッセージのタイプを定義します（例： FIX、EMX、...）
version	string	オプション： メッセージのバージョン（例： 4.1）
isOrdered	boolean	optional - デフォルト値 = false - FIX メッセージの生成時にフィールドの順番を変更できるようにします。このアノテーションはモデルのメッセージクラスに関連付けられ、一度宣言する必要があります。

case 1 : separator = 'u0001'

FIX メッセージ内のキーと値のペアフィールドを分離するために使用される区切り文字は ASCII '01' 文字または unicode 形式の '\u0001' です。この文字は、java ランタイムエラーを回避するために 2 回エスケープする必要があります。以下に例を示します。

```
8=FIX.4.1 9=20 34=1 35=0 49=INVMGR 56=BRKR 1=BE.CHM.001 11=CHM0001-01
22=4 ...
```

アノテーションの使用方法

FIX - message

```
@Message(keyValuePairSeparator = "=", pairSeparator = "\u0001", type="FIX", version="4.1")
public class Order {

}
```

> ***Look at test cases***

タブなどの ASCII 文字は WIKI ページには表示されません。そのため、camel-bindy のテストケースを確認し、FIX メッセージがどのように見えるか(src\test\data\fix.txt)と Order、Trailer、Header クラス(src\test\java\org\apache\camel\dataformat\bindy\model\fix\simple\Order.java)がどのように見えるかを確認します。

46.8. 6.KEYVALUEPAIRFIELD

KeyValuePairField アノテーションは、キーと値のペアフィールドのプロパティーを定義します。各 KeyValuePairField はタグ（= キー）とその関連付けられた値（タイプ（string、int、date、...）、任意パターン、およびフィールドが必要な場合で識別されます。

アノテーション名	レコードタイプ	レベル
KeyValuePairField	キー値のペア - FIX	プロパティー

パラメーター名	type	Info
tag	int	mandatory: メッセージのフィールドを識別する数字 - 一意である必要があります。
pattern	string	オプション： デフォルト値 = "" - will be used to format Decimal, Date, ...
精度	int	オプション： 数字の数字 - デクリー番号がフォーマット/解析されるときに使用される精度を表します。
position	int	オプション： FIX メッセージのキー/タグの位置が異なる場合は、これを使用する必要があります。
required	boolean	optional - default value = "false"
impliedDecimalSeparator	boolean	**Camel 2.11:** optional - デフォルト値 = "false" - 指定の場所の 10 進数の暗黙がある場合に示唆します。

case 1 : tag

このパラメーターは、メッセージのフィールドのキーを表します。

FIX message: Tag

```
@Message(keyValuePairSeparator = "=", pairSeparator = "\u0001", type="FIX", version="4.1")
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1) // Client reference
    private String Account;

    @KeyValuePairField(tag = 11) // Order reference
    private String ClOrdId;

    @KeyValuePairField(tag = 22) // Fund ID type (Sedol, ISIN, ...)
    private String IDSource;

    @KeyValuePairField(tag = 48) // Fund code
    private String SecurityId;

    @KeyValuePairField(tag = 54) // Movement type ( 1 = Buy, 2 = sell)
    private String Side;

    @KeyValuePairField(tag = 58) // Free text
    private String Text;
}
```

ケース 2: 出力の異なる位置

FIX メッセージに配置されるタグ/キーが事前に定義順に並べ替える必要がある場合、@KeyValuePairField のアノテーション 'position' 属性を使用します。

FIX message: Tag - sort

```
@Message(keyValuePairSeparator = "=", pairSeparator = "\\u0001", type = "FIX", version =
"4.1", isOrdered = true)
public class Order {

    @Link Header header;
```

```
    @Link Trailer trailer;

    @KeyValuePairField(tag = 1, position = 1) // Client reference
    private String account;

    @KeyValuePairField(tag = 11, position = 3) // Order reference
    private String clOrdId;
}
```

46.9. 7.セクション

固定長レコードの FIX メッセージでは、一般的には：ヘッダー、ボディー、およびセクションの情報を表すセクションが異なります。@Section アノテーションの目的は、モデルのどのクラスがヘッダー（= セクション 1）、本文（= セクション 2）、フッター（= セクション 3）を表すことにあります。

このアノテーションには 1 つの属性/パラメーターのみが存在します。

アノテーション名	レコードタイプ	レベル
セクション	FIX	クラス

パラメーター名	type	Info
number	int	セクションの位置を特定する数字

ケース 1: セクション

ヘッダーセクションの定義

FIX message - Section - Header

```
@Section(number = 1)
public class Header {
```

```
    @KeyValuePairField(tag = 8, position = 1) // Message Header
    private String beginString;

    @KeyValuePairField(tag = 9, position = 2) // Checksum
    private int bodyLength;
}
```

body セクションの定義

FIX message - Section - Body

```
@Section(number = 2)
@Message(keyValuePairSeparator = "=", pairSeparator = "\\u0001", type = "FIX", version =
"4.1", isOrdered = true)
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1, position = 1) // Client reference
    private String account;

    @KeyValuePairField(tag = 11, position = 3) // Order reference
    private String clOrdId;
```

footer セクションの定義

FIX message - Section - Footer

```
@Section(number = 3)
public class Trailer {

    @KeyValuePairField(tag = 10, position = 1)
    // CheckSum
    private int checkSum;

    public int getCheckSum() {
        return checkSum;
    }
```

46.10. 8.ONETOMANY

@OneToMany アノテーションの目的は、List<?> フィールドが POJO クラスを定義するか、反復グループが含まれるレコードから作業できるようにすることです。

> ***Restrictions OneToMany***

多くのバインドでは、階層の複数のレベルで定義されている繰り返しを処理することができないことに注意してください。

以下の場合に 1ToMany ONLY WORKS の関係：

- *繰り返しグループを含む FIX メッセージの読み取り（tags/keys のグループ）*
- *反復データでの CSV の生成*

アノテーション名	レコードタイプ	レベル
OneToMany	all	プロパティー

パラメーター名	type	Info
mappedTo	string	オプション - string - クラスの List<Type のタイプに関連付けられたクラス名>

ケース 1: 反復データを使用した CSV の生成

以下は、必要な CSV 出力です。

```
Claus,Ibsen,Camel in Action 1,2010,35
Claus,Ibsen,Camel in Action 2,2012,35
Claus,Ibsen,Camel in Action 3,2013,35
Claus,Ibsen,Camel in Action 4,2014,35
```

***Remark :** 反復データで、書籍のタイトルおよびその公開日に、姓と年齢が一般的です。*

そして、モデル化に使用されるクラス。Author クラスには、Book のリストが含まれています。

反復データを含む CSV の生成

```
@CsvRecord(separator=",")
public class Author {

    @DataField(pos = 1)
    private String firstName;

    @DataField(pos = 2)
    private String lastName;

    @OneToMany
    private List<Book> books;

    @DataField(pos = 5)
    private String Age;
}

public class Book {

    @DataField(pos = 3)
    private String title;

    @DataField(pos = 4)
    private String year;
}
```

非常にシンプルではありません！！

ケース 2: tags/keys グループが含まれる FIX メッセージの読み取り

以下は、モデルで処理したいメッセージです。

```
8=FIX 4.19=2034=135=049=INVMGR56=BRKR
1=BE.CHM.00111=CHM0001-0158=this is a camel - bindy test
22=448=BE000124567854=1
22=548=BE000987654354=2
22=648=BE000999999954=3
10=220
```

タグ 22、48、および 54 が繰り返し使用されます。

そしてコード

タグ/キーのグループが含まれる FIX メッセージの読み取り

```
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1) // Client reference
    private String account;

    @KeyValuePairField(tag = 11) // Order reference
    private String clOrdId;

    @KeyValuePairField(tag = 58) // Free text
    private String text;

    @OneToMany(mappedTo =
"org.apache.camel.dataformat.bindy.model.fix.complex.onetomany.Security")
    List<Security> securities;
}

public class Security {

    @KeyValuePairField(tag = 22) // Fund ID type (Sedol, ISIN, ...)
    private String idSource;

    @KeyValuePairField(tag = 48) // Fund code
    private String securityCode;

    @KeyValuePairField(tag = 54) // Movement type ( 1 = Buy, 2 = sell)
    private String side;
}
```

46.11. 9.BINDYCONVERTER

@BindyConverter アノテーションの目的は、フィールドレベルで使用されるコンバーターを定義します。提供されたクラスは Format インターフェースを実装する必要があります。

```
@FixedLengthRecord(length = 10, paddingChar = ' ')
public static class DataModel {
    @DataField(pos =  1, length = 10, trim = true)
    @BindyConverter(CustomConverter.class)
    public String field1;
}

public static class CustomConverter implements Format<String> {
```

```
    @Override
    public String format(String object) throws Exception {
        return (new StringBuilder(object)).reverse().toString();
    }

    @Override
    public String parse(String string) throws Exception {
        return (new StringBuilder(string)).reverse().toString();
    }
}
```

46.12. 10.FORMATFACTORIES

@FormatFactories アノテーションの目的は、レコードレベルでコンバーターのセットを定義することです。指定されたクラスは FormatFactoryInterface インターフェースを実装する必要があります。

```
@CsvRecord(separator = ",")
@FormatFactories({OrderNumberFormatFactory.class})
public static class Order {

    @DataField(pos = 1)
    private OrderNumber orderNr;

    @DataField(pos = 2)
    private String firstName;
}

public static class OrderNumber {
    private int orderNr;

    public static OrderNumber ofString(String orderNumber) {
        OrderNumber result = new OrderNumber();
        result.orderNr = Integer.valueOf(orderNumber);
        return result;
    }
}

public static class OrderNumberFormatFactory extends AbstractFormatFactory {

    {
        supportedClasses.add(OrderNumber.class);
    }

    @Override
    public Format<?> build(FormattingOptions formattingOptions) {
        return new Format<OrderNumber>() {
            @Override
            public String format(OrderNumber object) throws Exception {
                return String.valueOf(object.orderNr);
            }

            @Override
            public OrderNumber parse(String string) throws Exception {
```

```
            return OrderNumber.ofString(string);
        }
    };
  }
}
```

46.13. サポートされるデータタイプ

DefaultFormatFactory は、提供される FormattingOptions に基づいてインターフェース FormatFactoryInterface のインスタンスを返すことで、以下のデータタイプのフォーマットを提供します。

- BigDecimal
- BigInteger
- ブール値
- Byte
- Character
- Date
- double
- Enum
- Float
- 整数

- LocalDate（java 8 以降の 2.18.0）
- LocalDateTime (java 8, since 2.18.0)
- LocalTime (java 8, since 2.18.0)
- Long
- Short
- 文字列

DefaultFormatFactory は、使用中のレジストリーに FactoryRegistry のインスタンス（例： spring または JNDI）を指定して上書きできます。

46.14. JAVA DSL の使用

次の手順では、このレコードタイプに関連する DataFormat バインドクラスをインスタンス化し、Java パッケージ名をパラメーターとして提供します。

たとえば、以下は、com.acme.model パッケージ名で設定された BindyCsvDataFormat クラス（CSV レコードタイプに関連付けられたクラスに対応します）クラスを使用して、このパッケージで設定したモデルオブジェクトを初期化します。

```
// Camel 2.15 or older (configure by package name)
DataFormat bindy = new BindyCsvDataFormat("com.acme.model");


// Camel 2.16 onwards (configure by class name)
DataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);
```

46.14.1. ロケールの設定

bindy は、以下のようにデータフォーマットでロケールの設定をサポートします。

```
// Camel 2.15 or older (configure by package name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat("com.acme.model");
// Camel 2.16 onwards (configure by class name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);

bindy.setLocale("us");
```

プラットフォームのデフォルトロケールを使用するには、ロケール名として「default」を使用します。これには Camel 2.14/2.13.3/2.12.5 が必要です。

```
// Camel 2.15 or older (configure by package name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat("com.acme.model");
// Camel 2.16 onwards (configure by class name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);

bindy.setLocale("default");
```

以前のリリースでは、以下のように Java コードを使用して設定できます。

```
// Camel 2.15 or older (configure by package name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat("com.acme.model");
// Camel 2.16 onwards (configure by class name)
BindyCsvDataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);

bindy.setLocale(Locale.getDefault().getISO3Country());
```

46.14.2. アンマーシャリング

```
from("file://inbox")
  .unmarshal(bindy)
  .to("direct:handleOrders");
```

または、Spring XML ファイルなどのレジストリーで定義できるデータ形式への名前付き参照を使用することもできます。

```
from("file://inbox")
  .unmarshal("myBindyDataFormat")
  .to("direct:handleOrders");
```

Camel ルートは inbox ディレクトリーのファイルを取得して、CSV レコードをモデルオブジェクトのコレクションにアンマーシャリングし、コレクションを 'handleOrders' で参照されるルートに送信します。

返されるコレクションは Map オブジェクトの一覧 です。一覧内の各マップには、CSV の各行からマーシャリングされたモデルオブジェクトが含まれます。これは、各行が複数のオブジェクトに対応する可能性 がある理由です。これは、1 行ごとに 1 つのオブジェクトが返されることが予想される場合に混乱を生じさせる可能性があります。

各オブジェクトは、クラス名を使用して取得できます。

```
List<Map<String, Object>> unmarshaledModels = (List<Map<String, Object>>)
exchange.getIn().getBody();

int modelCount = 0;
for (Map<String, Object> model : unmarshaledModels) {
  for (String className : model.keySet()) {
     Object obj = model.get(className);
     LOG.info("Count : " + modelCount + ", " + obj.toString());
  }
 modelCount++;
}

LOG.info("Total CSV records received by the csv bean : " + modelCount);
```

ルートの処理のためにこのマップから単一の Order オブジェクトを抽出する場合、以下のように Splitter と Processor の組み合わせを使用できます。

```
from("file://inbox")
    .unmarshal(bindy)
    .split(body())
        .process(new Processor() {
            public void process(Exchange exchange) throws Exception {
                Message in = exchange.getIn();
                Map<String, Object> modelMap = (Map<String, Object>) in.getBody();
                in.setBody(modelMap.get(Order.class.getCanonicalName()));
            }
        })
        .to("direct:handleSingleOrder")
    .end();
```

Bindy は、Exchange インターフェースで CHARSET_NAME プロパティーまたは CHARSET_NAME ヘッダーを使用して、アンマーシャリング用に受信された入力ストリームの文字セット変換を行うことに注意してください。プロデューサーによっては（例： file-endpoint）、文字セットを定義できます。文字セット変換は、このプロデューサーによりすでに実行されています。エクスチェンジからこのプロパティーまたはヘッダーを削除してから、アンマーシャリングに送信する必要があります。これを削除しないと、変換が 2 回実行される可能性があり、不要な結果が発生する可能性があります。

```
from("file://inbox?charset=Cp922")
  .removeProperty(Exchange.CHARSET_NAME)
  .unmarshal("myBindyDataFormat")
```

```
    .to("direct:handleOrders");
```

46.14.3. マーシャリング

モデルオブジェクトのコレクションから CSV レコードを生成するには、以下のルートを作成します。

```
from("direct:handleOrders")
  .marshal(bindy)
  .to("file://outbox")
```

46.15. SPRING XML の使用

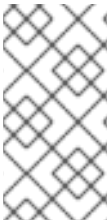
これは、camel-bindy に使用されるルートを宣言するためにお気に入りの DSL 言語として Spring を使用するのが非常に簡単です。以下の例は、最初のレコードをファイルから取得し、コンテンツをアンマーシャリングしてモデルにバインドする 2 つのルートを示しています。結果は pojo（特別なものを実行）に送信され、キューに配置されます。

2 つ目のルートはキューから pojos を抽出し、コンテンツをマーシャリングして csv レコードを含むファイルを生成します。上記の例は Camel 2.16 以降を使用している場合です。

spring dsl

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/spring
      http://camel.apache.org/schema/spring/camel-spring.xsd">

   <!-- Queuing engine - ActiveMq - work locally in mode virtual memory -->
   <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
      <property name="brokerURL" value="vm://localhost:61616"/>
   </bean>

   <camelContext xmlns="http://camel.apache.org/schema/spring">
      <dataFormats>
        <bindy id="bindyDataformat" type="Csv" classType="org.apache.camel.bindy.model.Order"/>
      </dataFormats>

      <route>
         <from uri="file://src/data/csv/?noop=true" />
         <unmarshal ref="bindyDataformat" />
```

```
        <to uri="bean:csv" />
        <to uri="activemq:queue:in" />
      </route>

      <route>
        <from uri="activemq:queue:in" />
        <marshal ref="bindyDataformat" />
        <to uri="file://src/data/csv/out/" />
      </route>
    </camelContext>
</beans>
```



注記

モデルクラスがシリアライズ可能であることを確認すると、キューマネージャーがエラーが発生します。

46.16. 依存関係

camel ルートで Bindy を使用するには、このデータ形式を実装する camel-bindy の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-bindy</artifactId>
  <version>x.x.x</version>
</dependency>
```

第47章 CAMEL での OSGI BLUEPRINT の使用

Blueprint のカスタム XML 名前空間が作成され、優れた XML 方言を活用できるようになります。Blueprint カスタム名前空間はまだ標準化されていません。この名前空間は、Apache Karaf によって使用される Blueprint 実装の Apache Aries Blueprint 実装でのみ使用できます。

47.1. 概要

XML スキーマは Spring のスキーマとほとんど同じであるため、Spring XML を参照するドキュメント全体のすべての xml スニペットも Blueprint ルートに適用されます。

以下は、Blueprint を使用した非常に単純なルート定義です。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <camelContext xmlns="http://camel.apache.org/schema/blueprint">
    <route>
      <from uri="timer:test" />
      <to uri="log:test" />
    </route>
  </camelContext>

</blueprint>
```

この時点では、サポートされる xml 要素（Spring xml 構文と比較）にはいくつかの制限があります。

- beanPostProcessor は Spring 固有のもので、許可されない

ただし、OSGi 環境内にアプリケーションをデプロイする場合、Blueprint を使用する利点は複数あります。

- 新しい Camel バージョンにアップグレードする場合は、バンドルによってインポートされる camel パッケージに基づいて正しいバージョンが選択されるため、namespace を変更する必要はありません。
- カスタム namespace およびバンドルに関して起動順序の問題はありません。

- Blueprint プロパティープレースホルダーを使用できます。

47.2. CAMEL-BLUEPRINT の使用

camel-blueprint を OSGi で活用するには、camel-core とその依存関係に加えて Aries Blueprint バンドルと camel-blueprint バンドルのみが必要になります。

Karaf を使用する場合は、必要なバンドルをすべてインストールする camel-blueprint という名前の機能を使用できます。

第48章 BONITA コンポーネント

Camel バージョン 2.19 から利用可能

リモート Bonita BPM プロセスエンジンとの通信に使用されます。

48.1. URI 形式

```
bonita://[operation]?[options]
```

ここでの operation は、Bonita で実行する固有のアクションになります。

48.2. 一般的なオプション

Bonita コンポーネントにはオプションがありません。

Bonita エンドポイントは URI 構文を使用して設定します。

```
bonita:operation
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

48.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
operation	使用する **必須** 操作		BonitaOperation

48.2.2. クエリーパラメーター（9 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
ホスト名 （コンシューマー）	Bonita エンジンが実行されるホスト名	localhost	文字列
ポート （コンシューマー）	Bonita エンジンをホストするサーバーのポート	8080	文字列
processName (consumer)	操作に関連するプロセスの名前		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
パスワード （セキュリティー）	Bonita エンジンに対して認証を行うためのパスワード。		文字列
ユーザー名 （セキュリティー）	Bonita エンジンに対して認証するユーザー名。		文字列

48.3. ボディーのコンテンツ

startCase 操作の場合、入力変数はボディーメッセージから取得されます。これには Map<String,336> を含める必要があります。

48.4. 例

以下の例は、Bonita で新しいケースを開始します。

```
from("direct:start").to("bonita:startCase?
hostname=localhost&amp;port=8080&amp;processName=TestProcess&amp;username=install&amp;password=install")
```

48.5. 依存関係

Camel ルートで Bonita を使用するには、コンポーネントを実装する camel-bonita の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加できます。バージョン番号は最新の最新のリリースに置き換えてください（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-bonita</artifactId>
  <version>x.x.x</version>
</dependency>
```

第49章 BOON DATAFORMAT

Camel バージョン 2.16 から利用可能

boon は、Boon JSON マーシャリングライブラリーを使用して JSON ペイロードを Java オブジェクトにアンマーシャリングしたり、Java オブジェクトを JSON ペイロードにマーシャリングしたりするデータ形式です。boon は、現在使用されている他の共通 パーサーよりも単純かつ高速パーサー となることを目的としています。

49.1. オプション

Boon データフォーマットは、以下に示す 3 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
unmarshalTypeName		文字列	無効化解除時に使用する java タイプのクラス名
useList	false	ブール値	マップのリストまたは Pojo のリストへ無視する場合は、以下を行います。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

49.2. JAVA DSL の使用

```
DataFormat boonDataFormat = new BoonDataFormat("com.acme.model.Person");

from("activemq:My.Queue")
  .unmarshal(boonDataFormat)
  .to("mqseries:Another.Queue");
```

49.3. BLUEPRINT XML の使用

```
<bean id="boonDataFormat" class="org.apache.camel.component.boon.BoonDataFormat">
  <argument value="com.acme.model.Person"/>
</bean>

<camelContext id="camel" xmlns="http://camel.apache.org/schema/blueprint">
  <route>
```

```
    <from uri="activemq:My.Queue"/>
    <unmarshal ref="boonDataFormat"/>
    <to uri="mqseries:Another.Queue"/>
  </route>
</camelContext>
```

49.4. 依存関係

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-boon</artifactId>
  <version>x.x.x</version>
</dependency>
```

第50章 ボックスコンポーネント

Camel バージョン 2.14 から利用可能

Box コンポーネントは、https://github.com/box/box-java-sdk でアクセス可能なすべての Box.com API へのアクセスを提供します。これにより、メッセージの生成により、ファイルのアップロードやダウンロード、フォルダーの作成、編集、および管理が可能になります。また、ユーザーアカウントの更新やエンタープライズアカウントの変更などをポーリングできる API もサポートしています。

box.com では、すべてのクライアントアプリケーション認証で OAuth2.0 を使用する必要があります。アカウントで camel-box を使用するには、https://developer.box.com の Box.com 内に新しいアプリケーションを作成する必要があります。Box アプリケーションのクライアント ID およびシークレットは、現行ユーザーを必要とする Box API へのアクセスを許可します。ユーザーのアクセストークンは、エンドユーザーの API によって生成および管理されます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-box</artifactId>
  <version>${camel-version}</version>
</dependency>
```

50.1. 接続認証タイプ

Box コンポーネントは 3 種類の認証接続をサポートします。

50.1.1. 標準認証

標準の認証 では、OAuth 2.0 の 3legged 認証プロセス を使用して Box.com で接続を認証します。このタイプの認証により、Box 管理のユーザー と 外部ユーザー は Box コンポーネント経由で Box コンテンツにアクセスし、編集し、保存できます。

50.1.2. アプリケーションのエンタープライズ認証

アプリケーションエンタープライズ認証 は、JSON Web Tokens(JWT)と OAuth 2.0 を使用して、Box アプリケーションの サービスアカウントと して接続を認証します。このタイプの認証により、サービスアカウントは Box コンポーネントを介して Box アプリケーション の Box コンテンツにアクセスし、編集し、保存することができます。

50.1.3. アプリケーションユーザー認証

アプリケーション ユーザー認証 は、JSON Web Tokens(JWT)と OAuth 2.0 を使用して、Box アプリケーションの App User として接続を認証します。このタイプの認証により、アプリケーション ユーザーは Box コンポーネントを介して Box コンテンツにアクセスし、編集し、保存することができます。

50.2. ボックスオプション

Box コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有設定の使用		BoxConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Box エンドポイントは URI 構文を使用して設定します。

```
box:apiName/methodName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

50.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
apiName	**必要な** 操作の種類		BoxApiName
methodName	選択した操作に使用するサブ操作が **必要**		文字列

50.2.2. クエリーパラメーター（20 パラメーター）：

Name	説明	デフォルト	Type
clientId (common)	Box アプリケーションクライアント ID		文字列
enterpriseId (common)	App Enterprise に使用するエンタープライズ ID。		文字列
inBody (common)	エクスチェンジ In Body で渡されるパラメーターの名前を設定します。		文字列
userId (common)	アプリケーションユーザーに使用するユーザー ID。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
httpParams (advanced)	プロキシーホストなどの設定のカスタム HTTP パラメーター		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
accessTokenCache (security)	アクセストークンの保存および取得のためのカスタムアクセストークンキャッシュ。		IAccessTokenCache
clientSecret (security)	Box アプリケーションクライアントシークレット		文字列
encryptionAlgorithm (security)	JWT の暗号化アルゴリズムのタイプ。サポートされるアルゴリズム： RSA_SHA_256 RSA_SHA_384 RSA_SHA_512	RSA_SHA_256	EncryptionAlgorithm

Name	説明	デフォルト	Type
maxCacheEntries (security)	キャッシュのアクセストークンの最大数。	100	int
authenticationType (authentication)	接続の認証のタイプ。認証の種類：STANDARD_AUTHENTICATION - OAuth 2.0(3-legged)SERVER_AUTHENTICATION - OAuth 2.0 with JSON Web Tokens	APP_USER_AUTHENTICATION	文字列
privateKeyFile (security)	JWT 署名を生成するための秘密鍵。		文字列
privateKeyPassword (security)	秘密鍵のパスワード。		文字列
publicKeyId (security)	JWT 署名を検証する公開鍵の ID。		文字列
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定します。		SSLContextParameters
userName (security)	ボックスユーザー名を指定する必要があります		文字列
userPassword (security)	ボックスユーザーパスワード。authSecureStorage が設定されていない場合、または最初の呼び出しで null を返す必要があります。		文字列

50.3. URI 形式

```
box:apiName/methodName
```

apiName は以下のいずれかになります。

- *コラボレーション*
- *コメント*

- *event-logs*
- *ファイル*
- *folders*
- *groups*
- *events*
- *search*
- *tasks*
- *users*

50.4. プロデューサーエンドポイント：

プロデューサーエンドポイントは、エンドポイント接頭辞の後にエンドポイント名および関連するオプションを使用できます。省略形のエイリアスは、一部のエンドポイントに使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションは [] で表されます。エンドポイントに必須オプションがない場合は、[] オプションのセットの 1 つを指定する必要があります。プロデューサーエンドポイントは、特別なオプション inBody を使用することもできます。そのオプションには、値が Camel Exchange In メッセージに含まれる endpoint オプションの名前が含まれる必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は CamelBox.<option> 形式である必要があります。inBody オプションはメッセージヘッダー（例： body = option）が CamelBox.option ヘッダーを上書きすることに注意してください。

エンドポイント URI またはメッセージヘッダーのいずれかで、オプション defaultRequest に対して

値が指定されていない場合、 null と見なされます。null 値は、他のオプションが一致するエンドポイントを満たさない場合にのみ使用されることに注意してください。

Box API エラーがある場合、エンドポイントは com.box.sdk.BoxAPIException の派生例外原因で RuntimeCamelException をスローします。

50.4.1. エンドポイント接頭辞 のコラボレーション

Box のコラボレーションに関する詳細は、https://developer.box.com/reference#collaboration-object を参照してください。 以下のエンドポイントは、以下のように接頭辞 のコラボレーション で呼び出すことができます。

```
box:collaborations/endpoint?[options]
```

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
addFolderCollaboration	add	folderId、コラボレーター、ロール	com.box.sdk.BoxCollaboration
addFolderCollaborationByEmail	addByEmail	folderId、email、role	com.box.sdk.BoxCollaboration
deleteCollaboration	削除	collaborationId	
getFolderCollaborations	コラボレーション	folderId	java.util.Collection
getPendingCollaborations	pendingCollaborations		java.util.Collection

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
getCollaborationInfo	info	collaborationId	com.box.sdk.BoxCollaboration.Info
updateCollaborationInfo	updateInfo	collaborationId, info	com.box.sdk.BoxCollaboration

コラボレーション用の URI オプション

Name	Type
collaborationId	文字列
コラボレーター	com.box.sdk.BoxCollaborator
role	com.box.sdk.BoxCollaboration.Role
folderId	文字列
email	文字列
info	com.box.sdk.BoxCollaboration.Info

50.4.2. エンドポイント接頭辞の コメント

Box のコメントに関する詳細は、https://developer.box.com/reference#comment-object を参照してください。 以下のエンドポイントは、以下のように接頭辞の コメント を使用して呼び出すことができます。

```
box:comments/endpoint?[options]
```

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
addFileComment	add	fileId, message	com.box.sdk.BoxFile
changeCommentMessage	updateMessage	commentId、message	com.box.sdk.BoxComment
deleteComment	削除	commentId	
getCommentInfo	info	commentId	com.box.sdk.BoxComment.Info
getFileComments	コメント	fileId	java.util.List
replyToComment	応答	commentId、message	com.box.sdk.BoxComment

コラボレーション用の URI オプション

Name	Type
commentId	文字列
fileId	文字列
message	文字列

50.4.3. エンドポイント接頭辞 events-logs

Box イベントログの詳細は、https://developer.box.com/reference#events を参照してください。以下のエンドポイントは、以下のようにプレフィックス イベント で呼び出すことができます。

box:event-logs/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
getEnterpriseEvents	events	position、after、before、[types]	java.util.List

event-logsの *URI* オプション

Name	Type
position	文字列
after	Date
before	Date
タイプ	com.box.sdk.BoxEvent.Types[]

50.4.4. エンドポイント接頭辞 ファイル

***Box* ファイルの詳細は、*https://developer.box.com/reference#file-object* を参照してください。以下のエンドポイントは、以下のように接頭辞 *ファイル* を使用して呼び出すことができます。**

box:files/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
uploadFile	Upload	parentFolderId, content, fileName, [created], [modified], [size], [listener]	com.box.sdk.BoxFile
downloadFile	download	fileId, output, [rangeStart], [rangeEnd], [listener]	java.io.OutputStream
copyFile	コピー	fileId, destinationFolderId, [newName]	com.box.sdk.BoxFile
moveFile	Move	fileId, destinationFolderId, [newName]	com.box.sdk.BoxFile
renameFile	rename	fileId, newFileName	com.box.sdk.BoxFile
createFileSharedLink	リンク	fileId, access, [unshareDate], [permissions]	com.box.sdk.BoxSharedLink

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
deleteFile	削除	fileId	
uploadNewFileVersion	uploadVersion	fileId, fileContent, [modified], [fileSize], [listener]	com.box.boxsdk.BoxFile
promoteFileVersion	promoteVersion	fileId、version	com.box.sdk.BoxFileVersion
getFileVersions	versions	fileId	java.util.Collection
downloadPreviousFileVersions	downloadVersion	fileId, version, output, [listener]	java.io.OutputStream
deleteFileVersion	deleteVersion	fileId、version	
getFileInfo	info	fileId、フィールド	com.box.sdk.BoxFile.Info
updateFileInfo	updateInfo	fileId, info	com.box.sdk.BoxFile
createFileMetadata	createMetadata	fileId, metadata, [typeName]	com.box.sdk.Metadata
getFileMetadata	metadata	fileId, [typeName]	com.box.sdk.Metadata

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
updateFileMetadata	updateMetadata	fileId, metadata	com.box.sdk.Metadata
deleteFileMetadata	deleteMetadata	fileId	
getDownloadUrl	url	fileId	java.net.URL
getPreviewLink	プレビュー	fileId	java.net.URL
getFileThumbnail	thumbnail	fileId、fileType、minWidth、minHeight、maxWidth、maxHeight	byte[]

ファイルの URI オプション

Name	Type
parentFolderId	文字列
content	java.io.InputStream
fileName	文字列
created	Date

Name	Type
modified	Date
size	Long
listener	com.box.sdk.ProgressListener
output	java.io.OutputStream
rangeStart	Long
rangeEnd	Long
outputStreams	java.io.OutputStream[]
destinationFolderId	文字列
newName	文字列
fields	String[]
info	com.box.sdk.BoxFile.Info
fileSize	Long
version	整数
access	com.box.sdk.BoxSharedLink.Access
unshareDate	Date
permissions	com.box.sdk.BoxSharedLink.Permissions
fileType	com.box.sdk.BoxFile.ThumbnailFileType

Name	Type
minWidth	整数
minHeight	整数
maxWidth	整数
maxHeight	整数
metadata	com.box.sdk.Metadata
typeName	文字列

50.4.5. エンドポイント接頭辞 フォルダー

Box フォルダーの詳細は、https://developer.box.com/reference#folder-object を参照してください。以下のエンドポイントは、以下のようにプレフィックス フォルダー を使用して呼び出すことができます。

```
box:folders/endpoint?[options]
```

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
getRootFolder	root		com.box.sdk.BoxFolder
createFolder	create	parentFolderId, folderName	com.box.sdk.BoxFolder
createFolder	create	parentFolderId, path	com.box.sdk.BoxFolder

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
copyFolder	コピー	folderId, destinationfolderId, [newName]	com.box.sdk.BoxFolder
moveFolder	Move	folderId, destinationFolderId, newName	com.box.sdk.BoxFolder
renameFolder	rename	folderId, newFolderName	com.box.sdk.BoxFolder
createFolderSharedLink	リンク	folderId, access, [unsharedDate], [permissions]	java.util.List
deleteFolder	削除	folderId	
getFolder	folder	path	com.box.sdk.BoxFolder
getFolderInfo	info	folderId, fields	com.box.sdk.BoxFolder.Info
getFolderItems	items	folderId、offsets、limit、field	com.box.sdk.BoxFolder

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
updateFolderInfo	updateInfo	folderId、info	com.box.sdk.BoxFolder

フォルダーの URI オプション

Name	Type
path	String[]
folderId	文字列
offset	Long
limit	Long
fields	String[]
parentFolderId	文字列
folderName	文字列
destinationFolderId	文字列
newName	文字列
newFolderName	文字列
info	文字列
access	com.box.sdk.BoxSharedLink.Access

Name	Type
unshareDate	Date
permissions	com.box.sdk.BoxSharedLink.Permissions

50.4.6. エンドポイント接頭辞 グループ

Box グループの詳細は、https://developer.box.com/reference#group-object を参照してください。以下のエンドポイントは、以下のようにプレフィックス グループ で呼び出すことができます。

> *box:groups/endpoint?[options]*

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
createGroup	create	name, [provenance, externalSyncIdentifier, description, invitabilityLevel, memberViewabilityLevel]	com.box.sdk.BoxGroup
addGroupMembership	createMembership	groupId, userId, role	com.box.sdk.BoxGroupMembership
deleteGroup	削除	groupId	
getAllGroups	groups		java.util.Collection
getGroupInfo	info	groupId	com.box.sdk.BoxGroup.Info

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
updateGroupInfo	updateInfo	groupId, groupInfo	com.box.sdk.BoxGroup
addGroupMembership	addMembership	groupId, userId, role	com.box.sdk.BoxGroupMembership
deleteGroupMembership	deleteMembership	groupMembershipId	
getGroupMemberships	メンバーシップ	groupId	java.uti.Collection
getGroupMembershipInfo	membershipInfo	groupMemebershipId	com.box.sdk.BoxGroup.Info
updateGroupMembershipInfo	updateMembershipInfo	groupMemebershipId, info	com.box.sdk.BoxGroupMembership

グループの URI オプション

Name	Type
name	文字列
groupId	文字列
userId	文字列
role	com.box.sdk.BoxGroupMembership.Role

Name	Type
groupMembershipId	文字列
info	com.box.sdk.BoxGroupMembership.Info

50.4.7. エンドポイント接頭辞の 検索

Box 検索 API の詳細は、https://developer.box.com/reference#searching-for-content を参照してください。 以下のエンドポイントは、以下のように接頭辞 search で呼び出すことができます。

```
box:search/endpoint?[options]
```

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
searchFolder	search	folderId, query	java.util.Collection

検索の URI オプション

Name	Type
folderId	文字列
query	文字列

50.4.8. エンドポイント接頭辞 タスク

Box タスクの詳細は、https://developer.box.com/reference#task-object-1 を参照してください。以下のエンドポイントは、以下のように接頭辞の タスク を使用して呼び出すことができます。

```
box:tasks/endpoint?[options]
```

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
addFileTask	add	fileId, action, dueAt, [message]	com.box.sdk.BoxUser
deleteTask	削除	taskId	
getFileTasks	tasks	fileId	java.util.List
getTaskInfo	info	taskId	com.box.sdk.BoxTask.Info
updateTaskInfo	updateInfo	taskid, info	com.box.sdk.BoxTask
addAssignmentToTask	addAssignment	taskId, assignTo	com.box.sdk.BoxTask
deleteTaskAssignment	deleteAssignment	taskAssignmentId	
getTaskAssignments	割り当て	taskId	java.util.List
getTaskAssignmentInfo	assignmentInfo	taskAssignmentId	com.box.sdk.BoxTaskAssignment.Info

タスクの URI オプション

Name	Type
fileId	文字列

Name	Type
action	com.box.sdk.BoxTask.Action
dueAt	Date
message	文字列
taskId	文字列
info	com.box.sdk.BoxTask.Info
assignTo	com.box.sdk.BoxUser
taskAssignmentId	文字列

50.4.9. エンドポイント接頭辞 ユーザー

Box ユーザーの詳細は、https://developer.box.com/reference#user-object を参照してください。以下のエンドポイントは、以下のように接頭辞 user を使用して呼び出すことが できます。

> *box:users/endpoint?[options]*

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
getCurrentUser	currentUser		com.box.sdk.BoxUser
getAllEnterpriseOrExternalUsers	users	filterTerm, [fields]	com.box.sdk.BoxUser
createAppUser	create	name, [params]	com.box.sdk.BoxUser

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
createEnterpriseUser	create	login、name、[params]	com.box.sdk.BoxUser
deleteUser	削除	userId, notifyUser, force	
getUserEmailAlias	emailAlias	userId	com.box.sdk.BoxUser
deleteUserEmailAlias	deleteEmailAlias	userId, emailAliasId	java.util.List
getUserInfo	info	userId	com.box.sdk.BoxUser.Info
updateUserInfo	updateInfo	userId, info	com.box.sdk.BoxUser
moveFolderToUser	-	userId, sourceUserId	com.box.sdk.BoxFolder.Info

ユーザーの URI オプション

Name	Type
defaultRequest	com.box.restclientv2.requestsbase.BoxDefaultRequestObject
emailAliasRequest	com.box.boxjavalibv2.requests.requestobjects.BoxEmailAliasRequestObject

Name	Type
emailId	文字列
filterTerm	文字列
folderId	文字列
simpleUserRequest	com.box.boxjavalibv2.requests.requestobjects.BoxSimpleUserRequestObject
userDeleteRequest	com.box.boxjavalibv2.requests.requestobjects.BoxUserDeleteRequestObject
userId	文字列
userRequest	com.box.boxjavalibv2.requests.requestobjects.BoxUserRequestObject
userUpdateLoginRequest	com.box.boxjavalibv2.requests.requestobjects.BoxUserUpdateLoginRequestObject

50.5. コンシューマーエンドポイント：

Box イベントの詳細は、https://developer.box.com/reference#events を参照してください。コンシューマーエンドポイントは、次の例のようにエンドポイントプレフィックス イベント のみを使用できます。

```
box:events/endpoint?[options]
```

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
events		[startingPosition]	com.box.sdk.BoxEvent

イベントの URI オプション

Name	Type
startingPosition	Long

50.6. メッセージヘッダー

任意のオプションは、CamelBox 接頭辞を持つプロデューサーエンドポイントのメッセージヘッダーに提供できます。

50.7. メッセージボディー

すべての結果メッセージ本文は、Box Java SDK によって提供されるオブジェクトを使用します。プロデューサーエンドポイントは、inBody エンドポイントパラメーターに受信メッセージボディーのオプション名を指定できます。

50.8. サンプル

以下のルートは、新しいファイルをユーザーの root フォルダーにアップロードします。

```
from("file:...")
    .to("box://files/upload/inBody=fileUploadRequest");
```

以下のルートは、ユーザーのアカウントをポーリングして更新を行います。

```
from("box://events/listen?startingPosition=-1")
    .to("bean:blah");
```

以下のルートは、動的ヘッダーオプションでプロデューサーを使用します。 fileId プロパティーには Box ファイル ID があり、output プロパティーにはファイルの内容の出力ストリームがあるため、以下のように CamelBox.fileId ヘッダーおよび CamelBox.output ヘッダーにそれぞれ割り当てられます。

```
from("direct:foo")
    .setHeader("CamelBox.fileId", header("fileId"))
    .setHeader("CamelBox.output", header("output"))
    .to("box://files/download")
    .to("file://...");
```

第51章 BRAINTREE COMPONENT

Camel バージョン 2.17 から利用可能

Braintree コンポーネントは、Java SDK 経由で Braintree Payments trough へのアクセスを提供します。

すべてのクライアントアプリケーションには、支払いの処理に API 認証情報が必要です。アカウントで camel-braintree を使用するには、新しい Sandbox または Production アカウントを作成する必要があります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-braintree</artifactId>
  <version>${camel-version}</version>
</dependency>
```

51.1. BRAINTREE のオプション

Braintree コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有設定の使用		BraintreeConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Braintree エンドポイントは URI 構文を使用して設定します。

braintree:apiName/methodName

以下の path パラメーターおよびクエリーパラメーターを使用します。

51.1.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
apiName	**必要な** 操作の種類		BraintreeApiName
methodName	選択した操作に使用するサブ操作		文字列

51.1.2. クエリーパラメーター（14 パラメーター）：

Name	説明	デフォルト	Type
環境 （共通）	SANDBOX または PRODUCTION のいずれかの環境		文字列
inBody (common)	エクスチェンジ In Body で渡されるパラメーターの名前を設定します。		文字列
merchantId (common)	Braintree が提供する以前の ID。		文字列
privateKey (common)	Braintree が提供する秘密鍵。		文字列
publicKey (common)	Braintree が提供する公開鍵。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
accessToken (advanced)	代理的にトランザクションを処理するために別のものに付与するアクセストークン。environment、merchant id、public key、および private key フィールドの代わりに使用されます。		文字列
httpReadTimeout (advanced)	http 呼び出しの読み取りタイムアウトを設定します。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
httpLogLevel (logging)	http 呼び出しのロギングレベルを設定します。java.util.logging.Level を参照してください。		文字列
proxyHost (proxy)	プロキシーホスト		文字列
proxyPort (proxy)	プロキシーポート		整数

51.2. URI 形式

> ***braintree://endpoint-prefix/endpoint?[options]***

エンドポイントプレフィックスは以下のいずれかになります。

- ***addOn***

- ***address***
- ***clientToken***
- ***creditCardverification***
- ***customer***
- ***discount***
- ***merchantAccount***
- ***paymentmethod***
- ***paymentmethodNonce***
- ***plan***
- ***settlementBatchSummary***
- ***subscription***
- ***transaction***
- ***webhookNotification***

51.3. BRAINTREECOMPONENT

Braintree コンポーネントは、以下のオプションで設定できます。これらのオプションは、org.apache.camel.component.braintree.BraintreeConfiguration タイプのコンポーネントの bean プロパティー 設定 を使用して指定できます。

オプション	型	説明
environment	文字列	要求を転送すべき場所を指定する値（サンドボックスまたは実稼働）
merchantId	文字列	ゲートウェイアカウントの一意識別子。以前のアカウント ID とは異なります。
publicKey	文字列	ユーザー固有のパブリック識別子
privateKey	文字列	ユーザー固有のセキュアな識別子と共有してはいけない！
accessToken	文字列	Braintree 認証を使用して付与されるトークンにより、Braintree 認証を使用してトランザクションを処理することができます。environment、merchantId、publicKey、および privateKey オプションの代わりに使用されます。

上記のオプションはすべて Braintree Payments で提供されます。

51.4. プロデューサーエンドポイント：

プロデューサーエンドポイントは、エンドポイント接頭辞の後にエンドポイント名および関連するオプションを使用できます。省略形のエイリアスは、一部のエンドポイントに使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションは [] で表されます。エンドポイントに必須オプションがない場合は、[] オプションのセットの 1 つを指定する必要があります。プロデューサーエンドポイントは、特別なオプション inBody を使用することもできます。そのオプションには、値が Camel Exchange In メッセージに含まれる endpoint オプションの名前が含まれる必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は CamelBraintree.<option> 形式である必要があります。inBody オプションはメッセージヘッダーを上書きすることに注意してください。つまり、endpoint オプション inBody=option は CamelBraintree.option ヘッダーを上書きすることに注意してください。

エンドポイントおよびオプションについての詳細は、https://developers.braintreepayments.com/reference/overviewの Braintree references を参照してください。

51.4.1. エンドポイント接頭辞 addOn

以下のようにプレフィックス addOn を使用して以下のエンドポイントを起動できます。

braintree://addOn/endpoint

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
all			List<com.braintreegateway.Addon>

51.4.2. エンドポイントプレフィックス アドレス

以下のエンドポイントは、以下のようにプレフィックス アドレス を使用して呼び出すことができます。

braintree://address/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
create		customerId, request	com.braintreegateway.Result<com.braintreegateway.Address>

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
削除		customerId, id	com.braintreegateway.Result<com.braintreegateway.Address>
find		customerId, id	com.braintreegateway.Address
更新		customerId, id, request	com.braintreegateway.Result<com.braintreegateway.Address>

アドレスの URI オプション

Name	Type
customerId	文字列
request	com.braintreegateway.AddressRequest
id	文字列

51.4.3. エンドポイント接頭辞 clientToken

以下のエンドポイントは、以下のようにプレフィックス clientToken を使用して呼び出すことができます。

braintree://clientToken/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
generate		request	文字列

clientTokenの *URI オプション*

Name	Type
request	com.braintreegateway.ClientTokenrequest

51.4.4. エンドポイントプレフィックス creditCardVerification

以下のエンドポイントは、以下のようにプレフィックス creditCardverification で呼び出すことができます。

braintree://creditCardVerification/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
find		id	com.braintreegateway.CreditCardVerification
search		query	com.braintreegateway.ResourceCollection<com.braintreegateway.CreditCardVerification>

creditCardVerificationの *URI オプション*

Name	Type
id	文字列
query	com.braintreegateway.CreditCardVerificationSearchRequest

51.4.5. endpoint prefix customer

以下のように、プレフィックス customer を使用して以下のエンドポイントを呼び出します。

braintree://customer/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
all			
create		request	com.braintreegateway.Result<com.braintreegateway.Customer>
削除		id	com.braintreegateway.Result<com.braintreegateway.Customer>
find		id	com.braintreegateway.Customer
search		query	com.braintreegateway.ResourceCollection<com.braintreegateway.Customer>
更新		id、request	com.braintreegateway.Result<com.braintreegateway.Customer>

顧客の *URI* オプション

Name	Type
id	文字列
request	com.braintreegateway.CustomerRequest
query	com.braintreegateway.CustomerSearchRequest

51.4.6. エンドポイント接頭辞の 割引

*以下のエンドポイントは、以下のように接頭辞の **discount** を使用して呼び出すことができます。*

braintree://discount/endpoint

–

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
all			List<com.braintreegateway.Discount>

+

+

51.4.7. エンドポイント接頭辞 merchantAccount

以下のエンドポイントは、以下のように接頭辞 merchantAccount で呼び出すことができます。

braintree:*//merchantAccount/endpoint?[options]*

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
create		request	com.braintreegateway.Result<com.braintreegateway.MerchantAccount>
createForCurrency		currencyRequest	com.braintreegateway.Result<com.braintreegateway.MerchantAccount>
find		id	com.braintreegateway.MerchantAccount
更新		id、request	com.braintreegateway.Result<com.braintreegateway.MerchantAccount>

merchantAccount*の URI オプション*

Name	Type
id	文字列

Name	Type
request	com.braintreegateway.MerchantAccountRequest
currencyRequest	com.braintreegateway.MerchantAccountCreateForCurrencyRequest

51.4.8. エンドポイント接頭辞 paymentMethod

*以下のエンドポイントは、以下のようにプレフィックス **paymentMethod** を使用して呼び出すことができます。*

> ***braintree:**//paymentMethod/endpoint?[options]*

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
create		request	com.braintreegateway.Result<com.braintreegateway.PaymentMethod>
削除		token, deleteRequest	com.braintreegateway.Result<com.braintreegateway.PaymentMethod>
find		token	com.braintreegateway.PaymentMethod
更新		token、request	com.braintreegateway.Result<com.braintreegateway.PaymentMethod>

paymentMethodの *URI* オプション

Name	Type
token	文字列
request	com.braintreegateway.PaymentMethodRequest

Name	Type
deleteRequest	com.braintreegateway.PaymentMethodDeleteRequest

51.4.9. エンドポイントプレフィックス paymentMethodNonce

以下のエンドポイントは、以下のようにプレフィックス ***paymentMethodNonce*** *で呼び出すことができます。*

braintree:*//paymentMethodNonce/endpoint?[options]*

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
create		paymentMethodToken	com.braintreegateway.Result<com.braintreegateway.PaymentMethodNonce>
find		paymentMethodNonce	com.braintreegateway.PaymentMethodNonce

URI Options for **paymentMethodNonce**

Name	Type
paymentMethodToken	文字列
paymentMethodNonce	文字列

51.4.10. エンドポイントプレフィックス プラン

以下のエンドポイントは、以下のようにプレフィックス プラン を使用して呼び出すことができます。

braintree://plan/endpoint

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
all			List<com.braintreegateway.Plan>

51.4.11. エンドポイント接頭辞 settlementBatchSummary

*以下のエンドポイントは、以下のようにプレフィックス **settlementBatchSummary** を使用して呼び出すことができます。*

braintree://settlementBatchSummary/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
generate		request	com.braintreegateway.Result<com.braintreegateway.SettlementBatchSummary>

***settlement* BatchSummary の *URI* オプション**

Name	Type
settlementDate	calendar
groupByCustomField	文字列

51.4.12. エンドポイントプレフィックス サブスクリプション

*以下のエンドポイントは、以下のようにプレフィックス **subscription** を使用して呼び出すことができます。*

braintree://subscription/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
取り消し		id	com.braintreegateway.Result<com.braintreegateway.Subscription>
create		request	com.braintreegateway.Result<com.braintreegateway.Subscription>
削除		customerId, id	com.braintreegateway.Result<com.braintreegateway.Subscription>
find		id	com.braintreegateway.Subscription
retryCharge		subscriptionId, amount	com.braintreegateway.Result<com.braintreegateway.Transaction>
search		searchRequest	com.braintreegateway.ResourceCollection<com.braintreegateway.Subscription>
更新		id、request	com.braintreegateway.Result<com.braintreegateway.Subscription>

サブスクリプションの *URI* オプション

Name	Type
id	文字列
request	com.braintreegateway.SubscriptionRequest
customerId	文字列
subscriptionId	文字列

Name	Type
amount	BigDecimal
searchRequest	com.braintreegateway.SubscriptionSearchRequest.

51.4.13. エンドポイント接頭辞の トランザクション

以下のエンドポイントは、以下のように接頭辞 トランザクション を使用して呼び出すことができます。

braintree://transaction/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
cancelRelease		id	com.braintreegateway.Result<com.braintreegateway.Transaction>
cloneTransaction		id, cloneRequest	com.braintreegateway.Result<com.braintreegateway.Transaction>
クレジット		request	com.braintreegateway.Result<com.braintreegateway.Transaction>
find		id	com.braintreegateway.Transaction
holdInEscrow		id	com.braintreegateway.Result<com.braintreegateway.Transaction>
releaseFromEscrow		id	com.braintreegateway.Result<com.braintreegateway.Transaction>

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
払い戻し		id, amount, refundRequest	com.braintreegateway.Result<com.braintreegateway.Transaction>
sale		request	com.braintreegateway.Result<com.braintreegateway.Transaction>
search		query	com.braintreegateway.ResourceCollection<com.braintreegateway.Transaction>
submitForPartialSettlement		id、Amount	com.braintreegateway.Result<com.braintreegateway.Transaction>
submitForSettlement		id、amount、request	com.braintreegateway.Result<com.braintreegateway.Transaction>
voidTransaction		id	com.braintreegateway.Result<com.braintreegateway.Transaction>

トランザクションの *URI* オプション

Name	Type
id	文字列
request	com.braintreegateway.TransactionCloneRequest
cloneRequest	com.braintreegateway.TransactionCloneRequest
refundRequest	com.braintreegateway.TransactionRefundRequest
amount	BigDecimal
query	com.braintreegateway.TransactionSearchRequest

51.4.14. エンドポイント接頭辞 webhookNotification

以下のエンドポイントは、以下のように webhookNotification の接頭辞で呼び出すことができます。

```
braintree://webhookNotification/endpoint?[options]
```

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
parse		署名、ペイロード	com.braintreegateway.WebhookNotification
検証		challenge	文字列

webhookNotificationの URI オプション

Name	Type
署名	文字列
payload	文字列
challenge	文字列

51.5. コンシューマーエンドポイント

プロデューサーエンドポイントはいずれもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、 consumer. プレフィックスと共に Scheduled Poll Consumer オプション を使用してエンドポイント呼び出しをスケジュールできます。デフォルトでは、配列またはコレクションを返すコンシューマーエンドポイントは要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 度実行されます。この動作を変更するには、 consumer.splitResults=true プロパティーを使用してリストまたはアレイ全体の単一のエクスチェンジを返します。

51.6. メッセージヘッダー

プロデューサーエンドポイントのメッセージヘッダーには、任意の URI オプション に CamelBraintree. プレフィックスを指定することができます。

51.7. メッセージボディー

結果メッセージ本文はすべて、Braintree Java SDK によって提供されるオブジェクトを使用します。プロデューサーエンドポイントは、 inBody エンドポイントパラメーターに受信メッセージボディーのオプション名を指定できます。

51.8. 例

Blueprint

```
<?xml version="1.0"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
          xsi:schemaLocation="
            http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0
http://aries.apache.org/schemas/blueprint-cm/blueprint-cm-1.0.0.xsd
            http://www.osgi.org/xmlns/blueprint/v1.0.0
https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
            http://camel.apache.org/schema/blueprint http://camel.apache.org/schema/blueprint/camel-
blueprint.xsd">


    <cm:property-placeholder id="placeholder" persistent-id="camel.braintree">
    </cm:property-placeholder>

    <bean id="braintree" class="org.apache.camel.component.braintree.BraintreeComponent">
        <property name="configuration">
            <bean class="org.apache.camel.component.braintree.BraintreeConfiguration">
                <property name="environment" value="${environment}"/>
                <property name="merchantId" value="${merchantId}"/>
                <property name="publicKey" value="${publicKey}"/>
                <property name="privateKey" value="${privateKey}"/>
            </bean>
        </property>
    </bean>

    <camelContext trace="true" xmlns="http://camel.apache.org/schema/blueprint" id="braintree-
```

```
example-context">
        <route id="braintree-example-route">
            <from uri="direct:generateClientToken"/>
            <to uri="braintree://clientToken/generate"/>
            <to uri="stream:out"/>
        </route>
    </camelContext>

</blueprint>
```

51.9. 関連項目

* Configuring Camel * Component * Endpoint * Getting Started（Camel * コンポーネント * エンドポイント * スタートガイド）

第52章 コンポーネントの閲覧

Camel バージョン 1.3 で利用可能

Browse コンポーネントは、テスト、可視化ツール、またはデバッグに役立つ簡単な BrowsableEndpoint を提供します。エンドポイントに送信されたエクスチェンジはすべて参照できます。

52.1. URI 形式

```
browse:someName[?options]
```

someName には、エンドポイントを一意に識別する文字列を使用できます。

52.2. オプション

Browse コンポーネントにはオプションがありません。

Browse エンドポイントは、URI 構文を使用して設定します。

```
browse:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

52.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	エンドポイントを一意に識別するための任意の文字列の名前が **必要です**。		文字列

52.2.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

52.3. 例

以下のルートで browse: コンポーネントを挿入して、渡されたエクスチェンジを閲覧できるようにします。

```
from("activemq:order.in").to("browse:orderReceived").to("bean:processOrder");
```

これで、受信したエクスチェンジを Java コード内から検査することができます。

```
private CamelContext context;

public void inspectRecievedOrders() {
    BrowsableEndpoint browse = context.getEndpoint("browse:orderReceived",
BrowsableEndpoint.class);
    List<Exchange> exchanges = browse.getExchanges();

    // then we can inspect the list of received exchanges from Java
    for (Exchange exchange : exchanges) {
        String payload = exchange.getIn().getBody();
```

```
        // do something with payload
    }
}
```

52.4. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第53章 EHCACHE コンポーネント（非推奨）

Camel バージョン 2.1 で利用可能

キャッシュ コンポーネントを使用すると、EHCache をキャッシュ実装として使用してキャッシュ操作を実行できます。キャッシュ自体はオンデマンドで作成されるか、その名前のキャッシュがすでに存在する場合、元の設定で使用されます。

このコンポーネントは、プロデューサーおよびイベントベースのコンシューマーエンドポイントをサポートします。

キャッシュコンシューマーはイベントベースのコンシューマーで、特定のキャッシュアクティビティーをリッスンして応答するために使用できます。既存のキャッシュからの選択を実行する必要がある場合は、キャッシュコンポーネントに定義されたプロセッサーを使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cache</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

53.1. URI 形式

```
cache://cacheName[?options]
```

URI には、?option=value&option=#beanRef&... という形式でクエリーオプションを追加できます。

53.2. オプション

EHCache コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
cacheManagerFactory (advanced)	指定の CacheManagerFactory を使用して CacheManager を作成するには、以下を行います。デフォルトでは、DefaultCacheManagerFactory が使用されます。		CacheManagerFactory
設定 （共通）	キャッシュ設定を設定します。		CacheConfiguration
configurationFile (common)	クラスパスまたはファイルシステムからロードする ehcache.xml ファイルの場所を設定します。デフォルトでは、ファイルは classpath:ehcache.xml から読み込まれます。	classpath:ehcache.xml	文字列
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

EHCache エンドポイントは、URI 構文を使用して設定します。

```
cache:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

53.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	キャッシュの **必須** 名		文字列

53.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
diskExpiryThreadInterval Seconds (common)	ディスク期限切れスレッドの実行間隔（秒単位）。		Long

Name	説明	デフォルト	Type
diskPersistent (common)	アプリケーションの再起動後、ディスクストアが永続化するかどうか。	false	boolean
diskStorePath (common)	**非推奨** 化されました。このパラメーターは無視されます。CacheManager は、セッターインジェクションを使用して設定します。		文字列
Teternal (common)	要素が気づくかどうかを設定します。お気に入りの場合、タイムアウトは無視され、要素は期限切れになりません。	false	boolean
キー (common)	使用するデフォルトのキー。キーがメッセージヘッダーで提供される場合、ヘッダーのキーが優先されます。		文字列
maxElementsInMemory (common)	定義されたキャッシュにメモリーに保存できる要素の数。	1000	int
memoryStoreEvictionPolicy (common)	メモリー内の要素の最大数に達したときに使用するエビクションストラテジー。ストラテジーは、削除する要素を定義します。LRU - Lest Recently Used LFU - Lest Frequently Used FIFO - First In First Out	LFU	MemoryStoreEviction ポリシー
objectCache (common)	キャッシュにシリアライズ不可能なオブジェクトを格納できるようにするかどうか。このオプションを有効にすると、ディスクへのオーバーフローも有効にできません。	false	boolean
操作 (common)	使用するデフォルトのキャッシュ操作。メッセージヘッダーの操作がある場合は、ヘッダーからの操作が優先されます。		文字列
overflowToDisk (common)	キャッシュがディスクにオーバーフローするべきかどうかを指定します。	true	boolean
timeToIdleSeconds (common)	要素が期限切れになるまでのアクセスの最大期間	300	Long
timeToLiveSeconds (common)	作成時間と要素の有効期限が切れるまでの最大時間。要素が気にない場合にのみ使用されます	300	Long

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cacheLoaderRegistry (advanced)	CacheLoaderRegistry を使用したキャッシュローダーの設定		CacheLoaderRegistry
cacheManagerFactory (advanced)	カスタムの CacheManagerFactory を使用してこのエンドポイントが使用する CacheManager を作成するには、以下を行います。デフォルトでは、コンポーネントに設定された CacheManagerFactory が使用されます。		CacheManagerFactory
eventListenerRegistry (advanced)	CacheEventListenerRegistry を使用してイベントリスナーを設定するには、以下を行います。		CacheEventListener Registry
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

53.3. キャッシュへのメッセージの送受信

53.3.1. Camel 2.7 までのメッセージヘッダー

ヘッダー	説明

ヘッダー	説明
CACHE_OPERATION	キャッシュで実行される操作。有効なオプションは以下のとおりです。 * GET * CHECK * ADD * UPDATE * DELETE * DELETEALL **GET** and **CHECK** requires **Camel 2.3** onwards
CACHE_KEY	キャッシュに Message を保存するために使用されるキャッシュキー。 CACHE_OPERATION が DELETEALL の場合、キャッシュキーはオプションになります。

53.3.2. Message Headers Camel 2.8+

Camel 2.8 でのヘッダー変更

ヘッダー名とサポートされる値は 'CamelCache' で始まり、混合ケースを使用します。これにより、他のヘッダーからの特定と分離が容易になります。CacheConstants 変数名は変更されませんが、値が変更されただけです。また、キャッシュ操作の実行後に、これらのヘッダーがエクスチェンジから削除されるようになりました。

ヘッダー	説明
CamelCacheOperation	キャッシュで実行される操作。有効なオプションは以下のとおりです。 * CamelCacheGet * CamelCacheCheck * CamelCacheAdd * CamelCacheUpdate * CamelCacheDeleteAll
CamelCacheKey	キャッシュに Message を保存するために使用されるキャッシュキー。 CamelCacheOperation が CamelCacheDeleteAll の場合、キャッシュキーは任意となります。

CamelCacheAdd および CamelCacheUpdate 操作は追加のヘッダーをサポートします。

ヘッダー	Type	説明
CamelCacheTimeToLive	**整数**	Camel 2.11: 存続する時間（秒単位）。

ヘッダー	Type	説明
CamelCacheTimeToIdle	**整数**	Camel 2.11: 秒単位でのアイドル時間。
CamelCacheEternal	**ブール値**	Camel 2.11: コンテンツが気づくかどうか。

53.3.3. キャッシュプロデューサー

データをキャッシュに送信するには、エクスチェンジにペイロードを直接送信して、既存のキャッシュまたは作成済みのオンデマンドキャッシュに保存する必要があります。これを行う仕組み

- *上記のメッセージエクスチェンジヘッダーの設定。*
- *メッセージエクスチェンジボディーにキャッシュにダイレクトされたメッセージが含まれるようにする*

53.3.4. キャッシュコンシューマー

キャッシュからデータを受信するには、Event リスナーを使用して既存または作成済みのオンデマンドキャッシュをリッスンし、キャッシュアクティビティーの発生時に自動通知を受け取る必要があります(CamelCacheGet/CamelCacheUpdate/CamelCacheDelete/CamelCacheDelete)。このようなアクティビティーの発生時

- *Message Exchange Headers と、追加したばかりのペイロードを含む Message Exchange Body が含まれるエクスチェンジが配置され、送信されます。*
- *CamelCacheDeleteAll 操作の場合、Message Exchange Header CamelCacheKey と Message Exchange Body は入力されません。*

53.3.5. キャッシュプロセッサー

キャッシュルックアップを実行し、ペイロードコンテンツを選択的に置き換える機能を持つ優れたプロセッサーのセットがあります。

- ボディー
- token
- XPath レベル

53.4. キャッシュ使用サンプル

53.4.1. 例 1: キャッシュの設定

```
from("cache://MyApplicationCache" +
        "?maxElementsInMemory=1000" +
        "&memoryStoreEvictionPolicy=" +
           "MemoryStoreEvictionPolicy.LFU" +
        "&overflowToDisk=true" +
        "&eternal=true" +
        "&timeToLiveSeconds=300" +
        "&timeToIdleSeconds=true" +
        "&diskPersistent=true" +
        "&diskExpiryThreadIntervalSeconds=300")
```

53.4.2. 例 2: キャッシュへのキーの追加

```
RouteBuilder builder = new RouteBuilder() {
   public void configure() {
    from("direct:start")
    .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_ADD))
    .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson"))
    .to("cache://TestCache1")
  }
};
```

53.4.3. 例 2: キャッシュの既存キーの更新

```
RouteBuilder builder = new RouteBuilder() {
   public void configure() {
    from("direct:start")
    .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_UPDATE))
    .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson"))
```

```
    .to("cache://TestCache1")
  }
};
```

53.4.4. 例 3: キャッシュ内の既存キーの削除

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
   from("direct:start")
   .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_DELETE))
   .setHeader(CacheConstants.CACHE_KEY", constant("Ralph_Waldo_Emerson"))
   .to("cache://TestCache1")
  }
};
```

53.4.5. 例 4: キャッシュ内の既存のキーをすべて削除

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
   from("direct:start")
   .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_DELETEALL))
   .to("cache://TestCache1");
  }
};
```

53.4.6. 例 5: キャッシュで登録された変更をプロセッサーおよびその他のプロデューサーに通知する

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
   from("cache://TestCache1")
   .process(new Processor() {
     public void process(Exchange exchange)
         throws Exception {
       String operation = (String)
exchange.getIn().getHeader(CacheConstants.CACHE_OPERATION);
       String key = (String) exchange.getIn().getHeader(CacheConstants.CACHE_KEY);
       Object body = exchange.getIn().getBody();
       // Do something
     }
   })
  }
};
```

53.4.7. 例 6: プロセッサーを使用したペイロードをキャッシュ値に置き換える

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
   //Message Body Replacer
   from("cache://TestCache1")
```

```
     .filter(header(CacheConstants.CACHE_KEY).isEqualTo("greeting"))
     .process(new CacheBasedMessageBodyReplacer("cache://TestCache1","farewell"))
     .to("direct:next");

    //Message Token replacer
    from("cache://TestCache1")
    .filter(header(CacheConstants.CACHE_KEY).isEqualTo("quote"))
    .process(new CacheBasedTokenReplacer("cache://TestCache1","novel","#novel#"))
    .process(new CacheBasedTokenReplacer("cache://TestCache1","author","#author#"))
    .process(new CacheBasedTokenReplacer("cache://TestCache1","number","#number#"))
    .to("direct:next");

    //Message XPath replacer
    from("cache://TestCache1").
    .filter(header(CacheConstants.CACHE_KEY).isEqualTo("XML_FRAGMENT"))
    .process(new CacheBasedXPathReplacer("cache://TestCache1","book1","/books/book1"))
    .process (new CacheBasedXPathReplacer("cache://TestCache1","book2","/books/book2"))
    .to("direct:next");
   }
};
```

53.4.8. 例 7: キャッシュからエントリーの取得

```
from("direct:start")
    // Prepare headers
    .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_GET))
    .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson")).
    .to("cache://TestCache1").
    // Check if entry was not found
    .choice().when(header(CacheConstants.CACHE_ELEMENT_WAS_FOUND).isNull()).
        // If not found, get the payload and put it to cache
        .to("cxf:bean:someHeavyweightOperation").
        .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_ADD))
        .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson"))
        .to("cache://TestCache1")
    .end()
    .to("direct:nextPhase");
```

53.4.9. 例 8: キャッシュのエントリーの確認

注記： CHECK コマンドはキャッシュにエントリーが存在するかどうかをテストしますが、ボディーにはメッセージを配置しません。

```
from("direct:start")
    // Prepare headers
    .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_CHECK))
    .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson")).
    .to("cache://TestCache1").
    // Check if entry was not found
```

```
    .choice().when(header(CacheConstants.CACHE_ELEMENT_WAS_FOUND).isNull()).
        // If not found, get the payload and put it to cache
        .to("cxf:bean:someHeavyweightOperation").
        .setHeader(CacheConstants.CACHE_OPERATION,
constant(CacheConstants.CACHE_OPERATION_ADD))
        .setHeader(CacheConstants.CACHE_KEY, constant("Ralph_Waldo_Emerson"))
        .to("cache://TestCache1")
    .end();
```

53.5. EHCACHE の管理

ehcache には、JMX からの独自の統計および管理があります。

以下は、Spring アプリケーションコンテキストで JMX 経由で公開するスニペットです。

```
<bean id="ehCacheManagementService" class="net.sf.ehcache.management.ManagementService"
init-method="init" lazy-init="false">
  <constructor-arg>
    <bean class="net.sf.ehcache.CacheManager" factory-method="getInstance"/>
  </constructor-arg>
  <constructor-arg>
    <bean class="org.springframework.jmx.support.JmxUtils" factory-method="locateMBeanServer"/>
  </constructor-arg>
  <constructor-arg value="true"/>
  <constructor-arg value="true"/>
  <constructor-arg value="true"/>
  <constructor-arg value="true"/>
</bean>
```

当然ながら、Java で直接実行できます。

```
ManagementService.registerMBeans(CacheManager.getInstance(), mbeanServer, true, true,
true, true);
```

このようにキャッシュヒット、ミス、メモリー内ヒット、ディスクヒット、サイズ統計を受け取ることができます。すぐに CacheConfiguration パラメーターを変更することもできます。

53.6. キャッシュレプリケーション CAMEL 2.8

Camel Cache コンポーネントは、RMI、JGroups、JMS、および Cache Server などの複数の異なるレプリケーションメカニズムを使用して、キャッシュをサーバーノードに分散できます。

これを機能させる方法は 2 つあります。

1.ehcache.xml は手動で設定できます。

あるいは

2.以下の 3 つのオプションを設定できます。

- *cacheManagerFactory*
- *eventListenerRegistry*
- *cacheLoaderRegistry*

1 つ目のオプションを使用した Camel Cache レプリケーションの設定は、すべてのキャッシュを個別に設定する必要があるため、少しのハード作業になります。そのため、キャッシュの名前がすべて不明な場合は、ehcache.xml を使用することが適切ではありません。

キャッシュごとにオプションを定義する必要がないため、2 番目のオプションは、多くの異なるキャッシュを使用したい方がはるかに適しています。これは、レプリケーションオプションが CacheManager ごとに設定され、CacheEndpoint ごとに設定されるためです。また、キャッシュ名が開発フェーズで不明な唯一の方法です。

注記: Camel Cache レプリケーションメカニズムをよりよく理解するには、EHCache マニュアル を読むと便利です。

53.6.1. 例： JMS キャッシュのレプリケーション

JMS レプリケーションは、最も強力でセキュアなレプリケーションメソッドです。Camel Cache レプリケーションとともに使用されると、シンプルになります。サンプルは 別のページで 利用できます。

第54章 CAFFEINE キャッシュコンポーネント

Camel バージョン 2.20 で利用可能

caffeine-cache コンポーネントを使用すると、Caffeine からの単純なキャッシュを使用してキャッシュ操作を実行できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-caffeine</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

54.1. URI 形式

```
caffeine-cache://cacheName[?options]
```

URI には、 ?option=value&option=#beanRef&... という形式でクエリーオプションを追加できます。

54.2. オプション

Caffeine Cache コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	グローバルコンポーネントの設定を設定します。		CaffeineConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Caffeine Cache エンドポイントは、URI 構文を使用して設定します。

caffeine-cache:cacheName

以下の path パラメーターおよびクエリーパラメーターを使用します。

54.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	キャッシュ名が **必要です**。		文字列

54.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
createCacheIfNotExist (common)	キャッシュが存在する場合や事前設定できない場合は、キャッシュを作成する必要がある場合を設定します。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
アクション （プロデューサー）	デフォルトのキャッシュアクションを設定します。アクションがメッセージヘッダーに設定されている場合は、ヘッダーからの操作が優先されます。		文字列
キャッシュ （プロデューサー）	デフォルトのインスタントキャッシュが使用されるように設定します。		Cache

Name	説明	デフォルト	Type
cacheLoader (producer)	LoadCache の使用時の CacheLoader の設定		CacheLoader
evictionType (producer)	このキャッシュのエビクションタイプの設定	SIZE_BASED	EvictionType
expireAfterAccessTime (producer)	時間ベースのエビクションの場合に有効期限の経過後の時間の設定（秒単位）	300	int
expireAfterWriteTime (producer)	時間ベースのエビクションの場合に有効期限の After Access Write を設定する（秒単位）	300	int
initialCapacity (producer)	キャッシュの初期容量の設定	10000	int
キー （プロデューサー）	デフォルトのアクションキーを設定します。キーがメッセージヘッダーに設定されている場合は、ヘッダーのキーが優先されます。		オブジェクト
maximumSize (producer)	キャッシュの最大サイズの設定	10000	int
removalListener (producer)	キャッシュに特定の削除リスナーを設定		RemovalListener
statsCounter (producer)	キャッシュ統計に特定の Stats カウンターを設定します。		StatsCounter
statsEnabled (producer)	キャッシュの統計の有効化	false	boolean
keyType (advanced)	キャッシュキータイプ（デフォルトは java.lang.Object）	java.lang.Object	文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
valueType (advanced)	キャッシュ値タイプ（デフォルトは java.lang.Object）	java.lang.Object	文字列

第55章 CAFFEINE LOADCACHE コンポーネント

Camel バージョン 2.20 で利用可能

caffeine-loadcache コンポーネントを使用すると、Caffeine からの Load キャッシュを使用してキャッシュ操作を実行できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-caffeine</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

55.1. URI 形式

```
caffeine-loadcache://cacheName[?options]
```

URI には、 ?option=value&option=#beanRef&... という形式でクエリーオプションを追加できます。

55.2. オプション

Caffeine LoadCache コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	グローバルコンポーネントの設定を設定します。		CaffeineConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Caffeine LoadCache エンドポイントは、URI 構文を使用して設定します。

caffeine-loadcache:cacheName

以下の path パラメーターおよびクエリーパラメーターを使用します。

55.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	キャッシュ名が **必要です**。		文字列

55.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
createCacheIfNotExist (common)	キャッシュが存在する場合や事前設定できない場合は、キャッシュを作成する必要がある場合を設定します。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
アクション （プロデューサー）	デフォルトのキャッシュアクションを設定します。アクションがメッセージヘッダーに設定されている場合は、ヘッダーからの操作が優先されます。		文字列
キャッシュ （プロデューサー）	デフォルトのインスタントキャッシュが使用されるように設定します。		Cache

Name	説明	デフォルト	Type
cacheLoader (producer)	LoadCache の使用時の CacheLoader の設定		CacheLoader
evictionType (producer)	このキャッシュのエビクションタイプの設定	SIZE_BASED	EvictionType
expireAfterAccessTime (producer)	時間ベースのエビクションの場合に有効期限の経過後の時間の設定（秒単位）	300	int
expireAfterWriteTime (producer)	時間ベースのエビクションの場合に有効期限の After Access Write を設定する（秒単位）	300	int
initialCapacity (producer)	キャッシュの初期容量の設定	10000	int
キー （プロデューサー）	デフォルトのアクションキーを設定します。キーがメッセージヘッダーに設定されている場合は、ヘッダーのキーが優先されます。		オブジェクト
maximumSize (producer)	キャッシュの最大サイズの設定	10000	int
removalListener (producer)	キャッシュに特定の削除リスナーを設定		RemovalListener
statsCounter (producer)	キャッシュ統計に特定の Stats カウンターを設定します。		StatsCounter
statsEnabled (producer)	キャッシュの統計の有効化	false	boolean
keyType (advanced)	キャッシュキータイプ（デフォルトは java.lang.Object）	java.lang.Object	文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
valueType (advanced)	キャッシュ値タイプ（デフォルトは java.lang.Object）	java.lang.Object	文字列

第56章 CASTOR DATAFORMAT（非推奨）

Camel バージョン 2.1 で利用可能

Castor は Data Format で、Cast or XML ライブラリーを使用して XML ペイロードを Java オブジェクトにアンマーシャリングするか、Java オブジェクトを XML ペイロードにマーシャリングします。

通常、Java DSL または Spring XML を使用して Castor Data Format と連携できます。

56.1. JAVA DSL の使用

```
from("direct:order").
  marshal().castor().
  to("activemq:queue:order");
```

たとえば、以下は、デフォルトの Castor データバインディング機能を使用する Castor の名前付き DataFormat を使用します。

```
CastorDataFormat castor = new CastorDataFormat ();

from("activemq:My.Queue").
  unmarshal(castor).
  to("mqseries:Another.Queue");
```

データフォーマットへの名前付き参照を使用したい場合は、Spring XML ファイルなどを介してレジストリーで定義できます。

```
from("activemq:My.Queue").
  unmarshal("mycastorType").
  to("mqseries:Another.Queue");
```

マッピングファイルを指定してデフォルトのマッピングスキーマをオーバーライドする場合は、以下のように設定できます。

```
CastorDataFormat castor = new CastorDataFormat ();
castor.setMappingFile("mapping.xml");
```

また、Castor Marshaller および Unmarshaller でさらに制御したい場合は、以下のようにそれらにアクセスできます。

```
castor.getMarshaller();
castor.getUnmarshaller();
```

56.2. SPRING XML の使用

以下の例は、キャストデータタイプの設定で Castor を使用してアンマーシャリングする方法を示しています。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <unmarshal>
      <castor validation="true" />
    </unmarshal>
    <to uri="mock:result"/>
  </route>
</camelContext>
```

この例は、データ型を一度だけ設定し、複数のルートに再利用する方法を示しています。<castor> 要素を <camelContext> に直接設定する必要があります。

```
<camelContext>
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <dataFormats>
    <castor id="myCastor"/>
  </dataFormats>

  <route>
    <from uri="direct:start"/>
    <marshal ref="myCastor"/>
    <to uri="direct:marshalled"/>
  </route>
  <route>
    <from uri="direct:marshalled"/>
    <unmarshal ref="myCastor"/>
    <to uri="mock:result"/>
  </route>

</camelContext>
```

56.3. オプション

Castor データフォーマットは、以下に示す 9 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
mappingFile		文字列	クラスパスからロードする Castor マッピングファイルへのパス。
whitelistEnabled	true	ブール値	ホワイトリスト機能が有効かどうかを定義します。
allowedUnmarshallObjects		文字列	アンマーシャリングできるオブジェクトを定義します。許可されるオブジェクトの FQN クラス名を指定でき、コンマを使用して複数のエントリーを分けることができます。リンク org.apache.camel.util.EndpointHelpermatchPattern(String, String)で定義されたパターンに基づいたワイルドカードおよび正規表現を使用することもできます。拒否されたオブジェクトは、許可されたオブジェクトよりも優先されます。
deniedUnmarshallObjects		文字列	アンマーシャリングする拒否されたオブジェクトを定義します。デインされたオブジェクトの FQN クラス名を指定でき、コンマを使用して複数のエントリーを区切ることができます。リンク org.apache.camel.util.EndpointHelpermatchPattern(String, String)で定義されたパターンに基づいたワイルドカードおよび正規表現を使用することもできます。拒否されたオブジェクトは、許可されたオブジェクトよりも優先されます。
検証	true	ブール値	検証がオンまたはオフであるかどうか。デフォルトは true です。
encoding	UTF-8	文字列	XML にオブジェクトをマーシャリングする際に使用するエンコーディング。デフォルトは UTF-8 です。
パッケージ		String []	Castor XmlContext にパッケージを追加
classes		String []	Castor XmlContext にクラス名を追加
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

56.4. 依存関係

camel ルートで Castor を使用するには、このデータ形式を実装する camel-castor の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-castor</artifactId>
  <version>x.x.x</version>
</dependency>
```

第57章 CAMEL CDI

Camel CDI コンポーネントは、CDI を convention-over-configuration をベースとした依存性注入フレームワークとして使用して Apache Camel の自動設定を提供します。これは、アプリケーションで利用可能な Camel ルートを自動検出し、Endpoint、FluentProducerTemplate、ProducerTemplate、TypeConverter などの共通の Camel プリミティブの Bean を提供します。CDI Bean で @Consume、@Produce、@PropertyInject などの Camel アノテーションをシームレスに使用できるように、標準の Camel Bean インテグレーションを実装します。さらに、Camel イベント（RouteAddedEvent、CamelContextStartedEvent、ExchangeCompletedEvent など）を CDI イベントとしてブリッジし、/ から Camel ルートに CDI イベントを消費/生成するのに使用できる CDI イベントエンドポイントを提供します。

Camel CDI コンポーネントは Camel 2.10 の時点で利用できますが、CDI プログラミングモデルにより適するように Camel 2.17 で書き換えられました。そのため、Camel イベントなどの機能の一部を CDI イベントブリッジと CDI events エンドポイントが起動する Camel 2.17 にのみ適用されます。

Camel CDI アプリケーションをテストする方法の詳細は、「Camel CDI testing」を参照してください。

注意

camel-cdi は OSGi で非推奨となり、サポートされていません。Camel を OSGi と併用する場合は、OSGi Blueprint を使用します。

57.1. 自動設定された CAMEL コンテキスト

Camel CDI は CamelContext Bean を自動的にデプロイし、設定します。CDI コンテナーの初期化（シャットダウン）時に、CamelContext Bean は自動的にインスタンス化され、設定され、起動（停止）されます。以下のように、アプリケーションに注入することができます。

```
@Inject
CamelContext context;
```

デフォルトの CamelContext Bean は組み込みの @Default 修飾子で修飾され、スコープは @ApplicationScoped で、タイプは DefaultCamelContext です。

この Bean は、プログラムを用いてカスタマイズでき、他の Camel コンテキスト Bean もアプリケーションにデプロイできます。

57.2. CAMEL ルートの自動検出

Camel CDI は、アプリケーション内のすべての RoutesBuilder Bean を自動的に収集し、CDI コンテナーの初期化時に CamelContext Bean インスタンスにインスタンス化し、追加します。たとえば、Camel ルートを追加するのは、以下のようにクラスを宣言するのと同じくらい簡単です。

```
class MyRouteBean extends RouteBuilder {

    @Override
    public void configure() {
        from("jms:invoices").to("file:/invoices");
    }
}
```

RoutesBuilder Bean は必要なだけ宣言できることに注意してください。さらに、 RouteContainer Bean も自動的に収集され、インスタンス化され、コンテナーの初期化時に Camel CDI によって管理される CamelContext Bean インスタンスに追加されます。

Camel 2.19 から利用可能

場合によっては、RouteBuilder Bean および RouteContainer Bean の自動設定を無効にする必要がある場合があります。これは、CdiCamelConfiguration イベントで確認できます。以下に例を示します。

```
static void configuration(@Observes CdiCamelConfiguration configuration) {
    configuration.autoConfigureRoutes(false);
}
```

同様に、設定された CamelContext Bean の自動開始を非アクティブにできます。以下に例を示します。

```
static void configuration(@Observes CdiCamelConfiguration configuration) {
    configuration.autoStartContexts(false);
}
```

57.3. 自動設定された CAMEL プリミティブ

Camel CDI は、CDI Bean にインジェクトできる共通の Camel プリミティブに Bean を提供します。以下に例を示します。

```
@Inject
```

```
@Uri("direct:inbound")
ProducerTemplate producerTemplate;

@Inject
@Uri("direct:inbound")
FluentProducerTemplate fluentProducerTemplate;

@Inject
MockEndpoint outbound; // URI defaults to the member name, i.e. mock:outbound

@Inject
@Uri("direct:inbound")
Endpoint endpoint;

@Inject
TypeConverter converter;
```

57.4. CAMEL コンテキスト設定

デフォルトの CamelContext Bean の名前を変更する場合は、Camel CDI によって提供される @ContextName 修飾子を使用できます。以下に例を示します。

```
@ContextName("camel-context")
class MyRouteBean extends RouteBuilder {

    @Override
    public void configure() {
        from("jms:invoices").to("file:/invoices");
    }
}
```

それ以外は、カスタマイズが必要な場合は、すべての CamelContext クラスを使用してカスタムの Camel コンテキスト Bean を宣言することができます。次に、カスタマイズを行うために @PostConstruct および @PreDestroy ライフサイクルコールバックを実行できます。以下に例を示します。

```
@ApplicationScoped
class CustomCamelContext extends DefaultCamelContext {

    @PostConstruct
    void customize() {
        // Set the Camel context name
        setName("custom");
        // Disable JMX
        disableJMX();
    }

    @PreDestroy
    void cleanUp() {
```

```
        // ...
    }
}
```

プロデューサー メソッドや *破棄メソッド* を使用して、Camel コンテキスト Bean をカスタマイズすることもできます。以下に例を示します。

```
class CamelContextFactory {

    @Produces
    @ApplicationScoped
    CamelContext customize() {
        DefaultCamelContext context = new DefaultCamelContext();
        context.setName("custom");
        return context;
    }

    void cleanUp(@Disposes CamelContext context) {
        // ...
    }
}
```

同様に、 *プロデューサーフィールド* を使用できます。以下に例を示します。

```
@Produces
@ApplicationScoped
CamelContext context = new CustomCamelContext();

class CustomCamelContext extends DefaultCamelContext {

    CustomCamelContext() {
        setName("custom");
    }
}
```

このパターンを使用すると、 setAutoStartup メソッドを呼び出すことで、コンテナーの初期化時に Camel コンテキストルートが自動的に起動されないようにすることができます。以下に例を示します。

```
@ApplicationScoped
class ManualStartupCamelContext extends DefaultCamelContext {

    @PostConstruct
    void manual() {
        setAutoStartup(false);
    }
}
```

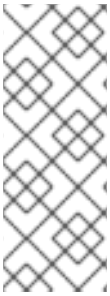
57.5. 複数の CAMEL コンテキスト

上記で文書化されているように、任意の数の CamelContext Bean を実際にアプリケーションで宣言できます。この場合、これらの CamelContext Bean で宣言された CDI 修飾子は Camel ルートおよびその他の Camel プリミティブを対応する Camel コンテキストにバインドするために使用されます。この例では、以下の Bean が宣言されます。

```
@ApplicationScoped
@ContextName("foo")
class FooCamelContext extends DefaultCamelContext {
}

@ApplicationScoped
@BarContextQualifier
class BarCamelContext extends DefaultCamelContext {
}

@ContextName("foo")
class RouteAddedToFooCamelContext extends RouteBuilder {

    @Override
    public void configure() {
        // ...
    }
}

@BarContextQualifier
class RouteAddedToBarCamelContext extends RouteBuilder {

    @Override
    public void configure() {
        // ...
    }
}

@ContextName("baz")
class RouteAddedToBazCamelContext extends RouteBuilder {

    @Override
    public void configure() {
        // ...
    }
}

@MyOtherQualifier
class RouteNotAddedToAnyCamelContext extends RouteBuilder {

    @Override
    public void configure() {
        // ...
    }
}
```

@ContextName で修飾された RoutesBuilder Bean は、Camel CDI によって対応する CamelContext Bean に自動的に追加されます。該当する CamelContext Bean が存在しない場合

は、RouteAddedToBazCamelContext Bean のように自動的に作成されます。これは、Camel CDI によって提供される @ContextName 修飾子でのみ発生することに注意してください。そのため、ユーザー定義の @MyOtherQualifier 修飾子で修飾された RouteNotAddedToAnyCamelContext Bean は Camel コンテキストに追加されません。これは、たとえば、アプリケーションの実行中に後で追加する必要がある Camel ルートなどに役立ちます。



注記

Camel バージョン 2.17.0 以降、Camel CDI はあらゆる種類の CamelContext Bean（デフォルトCamelContext など）を管理 できます。以前のバージョンでは、CdiCamelContext タイプの Bean のみを管理することができるため、拡張する必要がありました。

CamelContext Bean で宣言された CDI 修飾子は、対応する Camel プリミティブをバインドするために使用されます。以下に例を示します。

```
@Inject
@ContextName("foo")
@Uri("direct:inbound")
ProducerTemplate producerTemplate;

@Inject
@ContextName("foo")
@Uri("direct:inbound")
FluentProducerTemplate fluentProducerTemplate;

@Inject
@BarContextQualifier
MockEndpoint outbound; // URI defaults to the member name, i.e. mock:outbound

@Inject
@ContextName("baz")
@Uri("direct:inbound")
Endpoint endpoint;
```

57.6. 構成プロパティー

Camel がプロパティープレースホルダーを解決するために使用する設定プロパティーのソースを設定するには、@Named("properties") で修飾された PropertiesComponent Bean を宣言します。以下に例を示します。

```
@Produces
@ApplicationScoped
@Named("properties")
PropertiesComponent propertiesComponent() {
    Properties properties = new Properties();
    properties.put("property", "value");
```

```
    PropertiesComponent component = new PropertiesComponent();
    component.setInitialProperties(properties);
    component.setLocation("classpath:placeholder.properties");
    return component;
}
```

DeltaSpike 設定メカニズム を使用する場合は、以下の PropertiesComponent Bean を宣言できます。

```
@Produces
@ApplicationScoped
@Named("properties")
PropertiesComponent properties(PropertiesParser parser) {
    PropertiesComponent component = new PropertiesComponent();
    component.setPropertiesParser(parser);
    return component;
}

// PropertiesParser bean that uses DeltaSpike to resolve properties
static class DeltaSpikeParser extends DefaultPropertiesParser {
    @Override
    public String parseProperty(String key, String value, Properties properties) {
        return ConfigResolver.getPropertyValue(key);
    }
}
```

DeltaSpike 設定メカニズムを使用すると、Camel CDI アプリケーションの作業例は camel-example-cdi-properties の例で確認できます。

57.7. 自動設定された型コンバーター

@Converter アノテーションが付けられた CDI Bean は、以下のように、デプロイされた Camel コンテキストに自動的に登録されます。

```
@Converter
public class MyTypeConverter {

    @Converter
    public Output convert(Input input) {
        //...
    }
}
```

CDI インジェクションは型コンバーター内でサポートされることに注意してください。

57.8. CAMEL BEAN インテグレーション

57.8.1. Camel アノテーション

Camel Bean インテグレーション の一部として、Camel CDI によってシームレスにサポートされる アノテーション のセットが Camel CDI で提供されます。そのため、CDI Bean でこれらのアノテーションのいずれかを使用できます。以下に例を示します。

Camel アノテーション	CDI equivalent
設定プロパティー	@PropertyInject("key") String value; DeltaSpike 設定メカニズム を使用する場合： @Inject @ConfigProperty(name = "key") String value; 詳細は、「設定プロパティー」を参照してください。
プロデューサーテンプレートインジェクション（デフォルトの Camel コンテキスト）	@Produce(uri = "mock:outbound") ProducerTemplate producer; @Produce(uri = "mock:outbound") FluentProducerTemplate producer;	@Inject @Uri("direct:outbound") ProducerTemplate producer; @Produce(uri = "direct:outbound") FluentProducerTemplate producer;
エンドポイントインジェクション（デフォルトの Camel コンテキスト）	@EndpointInject(uri = "direct:inbound") Endpoint endpoint;	@Inject @Uri("direct:inbound") Endpoint endpoint;
エンドポイントインジェクション（名前による Camel コンテキスト）	@EndpointInject(uri = "direct:inbound", context = "foo") Endpoint contextEndpoint;	@Inject @ContextName("foo") @Uri("direct:inbound") Endpoint contextEndpoint;
Bean インジェクション（型別）	@BeanInject MyBean bean;	@Inject MyBean bean;
Bean インジェクション（名前別）	@BeanInject("foo") MyBean bean;	@Inject @Named("foo") MyBean bean;

	Camel アノテーション	CDI equivalent

POJO の消費	```
	@Consume(uri = "seda:inbound")
	void consume(@Body String
	body) {
	    //...
	}
	``	

57.8.2. Bean コンポーネント

Java Camel DSL のように、型または名前のいずれかで、Camel DSL から CDI Bean を参照できます。

```
class MyBean {
    //...
}

from("direct:inbound").bean(MyBean.class);
```

または、Java DSL から CDI Bean を名前で検索するには、以下を実行します。

```
@Named("foo")
class MyNamedBean {
    //...
}

from("direct:inbound").bean("foo");
```

57.8.3. エンドポイント URI からの Bean の参照

URI 構文を使用してエンドポイントを設定する場合、# 表記を使用してレジストリーの Bean を参照できます。 URI パラメーターの値が # 記号で始まる場合、Camel CDI は名前で指定されたタイプの Bean を検索します。以下に例を示します。

```
from("jms:queue:{{destination}}?
transacted=true&transactionManager=#jtaTransactionManager").to("...");
```

@Named("jtaTransactionManager") で修飾された CDI Bean があるとします。

```
@Produces
@Named("jtaTransactionManager")
```

```
PlatformTransactionManager createTransactionManager(TransactionManager
transactionManager, UserTransaction userTransaction) {
    JtaTransactionManager jtaTransactionManager = new JtaTransactionManager();
    jtaTransactionManager.setUserTransaction(userTransaction);
    jtaTransactionManager.setTransactionManager(transactionManager);
    jtaTransactionManager.afterPropertiesSet();
    return jtaTransactionManager;
}
```

57.9. CDI イベントへの CAMEL イベント

Camel 2.17 から利用可能

Camel は、Camel コンテキスト、サービス、ルート、および交換 イベントをリッスンするためにサブスクライブできる管理 イベントのセットを提供します。Camel CDI は、CDI オブザーバーメソッドを使用して確認できる CDI イベントにこれらの Camel イベントをシームレスに変換します。以下に例を示します。

```
void onContextStarting(@Observes CamelContextStartingEvent event) {
    // Called before the default Camel context is about to start
}
```

Camel 2.18 の時点で、特定のルート（RouteAddedEvent、RouteStartedEvent、RouteStoppedEvent および RouteRemovedEvent）のイベントを明示的に定義しているはずです。以下に例を示します。

```
from("...").routeId("foo").to("...");

void onRouteStarted(@Observes @Named("foo") RouteStartedEvent event) {
    // Called after the route "foo" has started
}
```

CDI コンテナーに複数の Camel コンテキストが存在する場合、@ContextName などの Camel コンテキスト Bean 修飾子を使用して、オブザーバー解決で指定された特定の Camel コンテキストにオブザーバーメソッドの 解決 を改良できます。以下に例を示します。

```
void onRouteStarted(@Observes @ContextName("foo") RouteStartedEvent event) {
    // Called after the route 'event.getRoute()' for the Camel context 'foo' has started
}

void onContextStarted(@Observes @Manual CamelContextStartedEvent event) {
    // Called after the the Camel context qualified with '@Manual' has started
}
```

同様に、@Default 修飾子は、複数のコンテキストが存在する場合に デフォルト の Camel コンテキ

ストの Camel イベントを監視するために使用できます。以下に例を示します。

```
void onExchangeCompleted(@Observes @Default ExchangeCompletedEvent event) {
    // Called after the exchange 'event.getExchange()' processing has completed
}
```

この例では、修飾子が指定されていない場合、 @Any 修飾子は暗黙的に考慮され、すべての Camel コンテキストの対応するイベントが受信されます。

CDI イベントへの Camel イベント変換のサポートは、Camel イベントをリッスンするオブザーバーメソッドがデプロイメントで検出され、Camel コンテキストごとに検出される場合にのみアクティベートされることに注意してください。

57.10. CDI イベントエンドポイント

Camel 2.17 から利用可能

CDI イベントエンドポイントは、 CDI イベント を Camel ルートとブリッジします。これにより、CDI イベントが Camel コンシューマー（Camel プロデューサーにより）Camel コンシューマーからシームレスに監視および消費（生成/実行される）が可能になります。

Camel CDI によって提供される CdiEventEndpoint<T > Bean は、イベントタイプ が T である CDI イベントを監視し、消費するために使用できます。以下に例を示します。

```
@Inject
CdiEventEndpoint<String> cdiEventEndpoint;

from(cdiEventEndpoint).log("CDI event received: ${body}");
```

これは以下を記述するのと同じです。

```
@Inject
@Uri("direct:event")
ProducerTemplate producer;

void observeCdiEvents(@Observes String event) {
    producer.sendBody(event);
}

from("direct:event").log("CDI event received: ${body}");
```

逆に、 CdiEventEndpoint<T> Bean を使用して、イベントタイプ が T である CDI イベントを生成/実行することができます。以下に例を示します。

```
@Inject
CdiEventEndpoint<String> cdiEventEndpoint;

from("direct:event").to(cdiEventEndpoint).log("CDI event sent: ${body}");
```

これは以下を記述するのと同じです。

```
@Inject
Event<String> event;

from("direct:event").process(new Processor() {
    @Override
    public void process(Exchange exchange) {
        event.fire(exchange.getBody(String.class));
    }
}).log("CDI event sent: ${body}");
```

または、Java 8 lambda 式を使用します。

```
@Inject
Event<String> event;

from("direct:event")
    .process(exchange -> event.fire(exchange.getIn().getBody(String.class)))
    .log("CDI event sent: ${body}");
```

特定の CdiEventEndpoint< T > インジェクションポイントの type 変数 T （修飾子）は、以下のようにパラメーター化された イベントタイプ (resp.)に自動変換されます。

```
@Inject
@FooQualifier
CdiEventEndpoint<List<String>> cdiEventEndpoint;

from("direct:event").to(cdiEventEndpoint);

void observeCdiEvents(@Observes @FooQualifier List<String> event) {
    logger.info("CDI event: {}", event);
}
```

CDI コンテナーに複数の Camel コンテキストが存在する場合、 @ContextName などの Camel コンテキスト Bean 修飾子は、 CdiEventEndpoint<T > インジェクションポイントの認定に使用できます。

以下に例を示します。

```
@Inject
@ContextName("foo")
CdiEventEndpoint<List<String>> cdiEventEndpoint;
// Only observes / consumes events having the @ContextName("foo") qualifier
from(cdiEventEndpoint).log("Camel context (foo) > CDI event received: ${body}");
// Produces / fires events with the @ContextName("foo") qualifier
from("...").to(cdiEventEndpoint);

void observeCdiEvents(@Observes @ContextName("foo") List<String> event) {
   logger.info("Camel context (foo) > CDI event: {}", event);
}
```

CDI イベント Camel エンドポイントは、イベント型と イベント 修飾 子の一意の組み合わせごとに オブザーバーメソッド を動的に追加し、コンテナータイプ セーフ オブザーバー解決 のみに依存することに注意してください。これにより、実装ができるだけ効率的になります。

さらに、CDI の型セーフ 性質と Camel コンポーネント モデルの 動的 な性質間の移動は非常に高いため、 URI 経由で CDI イベント Camel エンドポイントのインスタンスを作成することはできません。実際、CDI イベントコンポーネントの URI 形式は次のとおりです。

```
cdi-event://PayloadType<T1,...,Tn>[?qualifiers=QualifierType1[,...[,QualifierTypeN]...]]
```

authority PayloadType (resp. QualifierType)は、ペイロード(resp. qualifier)の raw タイプで、その後にペイロードパラメーター化された値の括弧で区切られた type parameters セクションです。これにより、以下のように、 分かりやすい URI が発生 します。

```
cdi-event://org.apache.camel.cdi.example.EventPayload%3Cjava.lang.Integer%3E?
qualifiers=org.apache.camel.cdi.example.FooQualifier%2Corg.apache.camel.cdi.example.Bar
Qualifier
```

しかし、基本的には、CDI コンテナーにはデプロイメントフェーズ中に Camel コンテキストモデルを検出する方法がないため、エンドポイントインスタンスとオブザーバーメソッドとの間の効率的なバインディングを防ぐことができます。

57.11. CAMEL XML 設定のインポート

Camel 2.18 から利用可能

CDI は型安全なディペンデンシーインジェクションメカニズムを優先しますが、既存の Camel XML 設定ファイルを Camel CDI アプリケーションに再利用することが役に立つ場合があります。 その他の

ユースケースでは、Camel XML DSL を使用して Camel コンテキストを設定すると便利です。

CDI Bean で Camel CDI によって提供される @ImportResource アノテーションを使用でき、Camel CDI は指定された場所で Camel XML 設定を自動的にロードします。以下に例を示します。

```
@ImportResource("camel-context.xml")
class MyBean {
}
```

Camel CDI は、指定の場所でクラスパスからリソースをロードします（他のプロトコルは今後追加される可能性があります）。

インポートされたリソースからの CamelContext 要素およびその他の Camel プリミティブ はすべて、コンテナーブートストラップ中に CDI Bean として自動的にデプロイされ、Camel CDI が提供する自動設定から利点を受け、ランタイム時にインジェクションに利用できるようになります。このような要素に明示的な id 属性が設定されている場合、対応する CDI Bean は @Named 修飾子で修飾されます（例： 以下の Camel XML 設定）。

```
<camelContext id="foo">
    <endpoint id="bar" uri="seda:inbound">
        <property key="queue" value="#queue"/>
        <property key="concurrentConsumers" value="10"/>
    </endpoint>
<camelContext/>
```

対応する CDI Bean は自動的にデプロイされ、注入することができます。以下に例を示します。

```
@Inject
@ContextName("foo")
CamelContext context;

@Inject
@Named("bar")
Endpoint endpoint;
```

CamelContext Bean は @Named 修飾子と @ContextName 修飾子の両方で自動的に修飾されることに注意してください。インポートされた CamelContext 要素に id 属性がない場合、対応する Bean は組み込みの @Default 修飾子でデプロイされます。

逆に、アプリケーションにデプロイされた CDI Bean は Camel XML 設定から参照できます。通常、以下の Bean が宣言されている場合は ref 属性を使用します。

```
@Produces
@Named("baz")
Processor processor = exchange -> exchange.getIn().setHeader("qux", "quux");
```

Bean への参照は、インポートされた Camel XML 設定で宣言できます。以下に例を示します。

```
<camelContext id="foo">
    <route>
        <from uri="..."/>
        <process ref="baz"/>
    </route>
<camelContext/>
```

57.12. トランザクションサポート

Camel 2.19 から利用可能

Camel CDI は、JTA を使用した Camel トランザクションクライアントをサポートします。

このサポートはオプションであるため、Maven を使用する際に JTA を依存関係として明示的に追加することで、アプリケーションクラスパスに JTA が必要になります。

```
<dependency>
    <groupId>javax.transaction</groupId>
    <artifactId>javax.transaction-api</artifactId>
    <scope>runtime</scope>
</dependency>
```

アプリケーションを JTA 対応コンテナーにデプロイしたり、スタンドアロンの JTA 実装を提供する必要があります。

注意

その間、トランザクションマネージャーは java:/TransactionManager キーを使用して JNDI リソースとしてルックアップされることに注意してください。

各種のデプロイメントシナリオをサポートするために、より柔軟なストラテジーが今後追加されます。

57.12.1. トランザクションポリシー

Camel CDI は、CDI Bean としてサポートされる Camel TransactedPolicy の実装を提供します。これらのポリシーをトランザクション EIP を使用して名前で検索できます。以下に例を示します。

```
class MyRouteBean extends RouteBuilder {

    @Override
    public void configure() {
        from("activemq:queue:foo")
            .transacted("PROPAGATION_REQUIRED")
            .bean("transformer")
            .to("jpa:my.application.entity.Bar")
            .log("${body.id} inserted");
    }
}
```

これは以下に相当します。

```
class MyRouteBean extends RouteBuilder {

    @Inject
    @Named("PROPAGATION_REQUIRED")
    Policy required;

    @Override
    public void configure() {
        from("activemq:queue:foo")
            .policy(required)
            .bean("transformer")
            .to("jpa:my.application.entity.Bar")
            .log("${body.id} inserted");
    }
}
```

サポートされるトランザクションポリシー名の一覧は、以下のとおりです。

- PROPAGATION_NEVER,
- PROPAGATION_NOT_SUPPORTED,
- PROPAGATION_SUPPORTS,

- *PROPAGATION_REQUIRED,*
- *PROPAGATION_REQUIRES_NEW,*
- *PROPAGATION_NESTED,*
- *PROPAGATION_MANDATORY.*

57.12.2. トランザクションエラーハンドラー

Camel CDI は再配信エラーハンドラーを拡張するトランザクションエラーハンドラーを提供し、例外が発生し、再配信ごとに新しいトランザクションを作成します。

Camel CDI は、transactionErrorHandler ヘルパーメソッドを公開して、設定への迅速なアクセスを可能にする CdiRouteBuilder クラスを提供します。以下に例を示します。

```
class MyRouteBean extends CdiRouteBuilder {

    @Override
    public void configure() {
        errorHandler(transactionErrorHandler()
            .setTransactionPolicy("PROPAGATION_SUPPORTS")
            .maximumRedeliveries(5)
            .maximumRedeliveryDelay(5000)
            .collisionAvoidancePercent(10)
            .backOffMultiplier(1.5));
    }
}
```

57.13. 自動設定された OSGI 統合

Camel 2.17 から利用可能

Camel コンテキスト Bean は Camel CDI によって自動的に適応され、OSGi サービスとして登録され、各種リゾルバー（ComponentResolver や DataFormatResolverなど）が OSGi レジストリーと統合されます。つまり、Karaf Camel コマンドを使用して、Camel CDI によって自動設定された Camel コンテキストを操作できます。以下に例を示します。

```
karaf@root()> camel:context-list
 Context       Status            Total #       Failed #     Inflight #   Uptime
 -------       ------            -------       --------     ----------   ------
 camel-cdi     Started                 1              0              0   1 minute
```

Camel CDI OSGi インテグレーションの作業例は、 camel-example-cdi-osgi の例を参照してください。

57.14. LAZY INJECTION / PROGRAMMATIC LOOKUP

CDI プログラムによるモデルはアプリケーションの初期化時に発生する タイプセーフ解決 メカニズムを優先しますが、後でプログラムに よるルックアップ メカニズムを使用してアプリケーションの実行時に動的な/レイジーインジェクションを実行できます。

Camel CDI は、Camel プリミティブの標準インジェクションに使用できる CDI 修飾子に対応するアノテーションリテラルを提供します。これらのアノテーションリテラルは、レイジーインジェクション/プログラムによるルックアップを実行する CDI エントリーポイントである javax.enterprise.inject.Instance インターフェースと組み合わせて使用できます。

たとえば、@Uri 修飾子に提供されたアノテーションリテラルを使用して、ProducerTemplate Bean などの Camel プリミティブの遅延検索を行うことができます。

```
@Any
@Inject
Instance<ProducerTemplate> producers;

ProducerTemplate inbound = producers
    .select(Uri.Literal.of("direct:inbound"))
    .get();
```

または、エンドポイント Bean 用です。以下に例を示します。

```
@Any
@Inject
Instance<Endpoint> endpoints;

MockEndpoint outbound = endpoints
    .select(MockEndpoint.class, Uri.Literal.of("mock:outbound"))
    .get();
```

同様に、@ContextName 修飾子に提供されたアノテーションリテラルを使用して、CamelContext Bean の遅延検索を行うことができます。以下に例を示します。

```
@Any
@Inject
Instance<CamelContext> contexts;

CamelContext context = contexts
    .select(ContextName.Literal.of("foo"))
    .get();
```

Camel コンテキストタイプに基づいて選択を調整することもできます。以下に例を示します。

```
@Any
@Inject
Instance<CamelContext> contexts;

// Refine the selection by type
Instance<DefaultCamelContext> context = contexts.select(DefaultCamelContext.class);

// Check if such a bean exists then retrieve a reference
if (!context.isUnsatisfied())
    context.get();
```

または、以下のような Camel コンテキストの選択を繰り返し処理することもできます。

```
@Any
@Inject
Instance<CamelContext> contexts;

for (CamelContext context : contexts)
    context.setUseBreadcrumb(true);
```

57.15. MAVEN ARCHETYPE

利用可能な Camel Maven archetype の中で、提供される camel-archetype-cdi を使用して Camel CDI Maven プロジェクトを生成できます。以下に例を示します。

```
mvn archetype:generate -DarchetypeGroupId=org.apache.camel.archetypes -
DarchetypeArtifactId=camel-archetype-cdi
```

57.16. サポートされるコンテナー

Camel CDI コンポーネントは、CDI 1.0、CDI 1.1、および CDI 1.2 準拠のランタイムと互換性があります。以下のランタイムに対して正常にテストされています。

Container	バージョン	ランタイム
Weld SE	**1.1.28.Final**	CDI 1.0 / Java SE 7
OpenWebBeans	**1.2.7**	CDI 1.0 / Java SE 7
Weld SE	**2.4.2.Final**	CDI 1.2 / Java SE 7
OpenWebBeans	**1.7.2**	CDI 1.2 / Java SE 7
WildFly	**8.2.1.Final**	CDI 1.2 / Java EE 7
WildFly	**9.0.1.Final**	CDI 1.2 / Java EE 7
WildFly	**10.1.0.Final**	CDI 1.2 / Java EE 7

57.17. 例

以下の例は、Camel プロジェクトの サンプル ディレクトリーにあります。

例	説明
camel-example-cdi	では、CDI を使用して Camel と連携してコンポーネント、エンドポイント、および Bean を設定する方法を説明します。
camel-example-cdi-kubernetes	は、Camel、CDI、Kubernetes 間のインテグレーションを示しています。
camel-example-cdi-metrics	は、Camel、Dropwizard Metrics と CDI 間のインテグレーションを示しています。
camel-example-cdi-properties	は、設定プロパティーの Camel、DeltaSpike と CDI 間のインテグレーションを示しています。
camel-example-cdi-osgi	PAX CDI を使用して OSGi コンテナー内で実行できる SJMS コンポーネントを使用する CDI アプリケーション
camel-example-cdi-rest-servlet	は、CDI を依存性注入フレームワークとして使用する Web アプリケーションで使用される Camel REST DSL を示しています。
camel-example-cdi-test	Camel と CDI 間のインテグレーションの一部として提供されるテスト機能を実証します。
camel-example-cdi-xml	は、Camel XML 設定ファイルを使用した Camel CDI アプリケーションでの使用について示しています。

例	説明
camel-example-swagger-cdi	CDI で REST DSL と Swagger Java の使用例
camel-example-widget-gadget-cdi	CDI ディペンデンシーインジェクションで Java に実装された EIP ドキュメンテーションの Widget および Gadget ユースケース

57.18. 関連項目

- ***Camel CDI テスト***
- ***CDI 仕様の Web サイト***
- ***CDI エコシステム***
- ***Weld ホームページ***
- ***OpenWebBeans ホームページ***
- ***CDI および Camel を使用する方法（Camel CDI セクションを参照）***

第58章 CHRONICLE ENGINE コンポーネント

Camel バージョン 2.18 から利用可能

camel chronicle-engine コンポーネントを使用すると、OpenHFT の Chronicle-Engine の機能を活用できます。

58.1. URI 形式

```
chronicle-engine:addresses/path[?options]
```

58.2. URI オプション

Chronicle Engine コンポーネントにはオプションがありません。

Chronicle Engine エンドポイントは、URI 構文を使用して設定します。

```
chronicle-engine:addresses/path
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

58.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
addresses	**必要な** エンジンアドレス。複数のアドレスはコンマで区切ることができます。		文字列
path	**必要な** エンジンパス		文字列

58.2.2. クエリーパラメーター（12 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
アクション （共通）	実行するデフォルトの動作。有効な値は - PUBLISH - PPUBLISH_AND_INDEX - PPUT - PGET_AND_PUT - PPUT_ALL - PPUT_IF_ABSENT - PGET - PGET_AND_REMOVE - PREMOVE - PIS_EMPTY - PSIZE		文字列
clusterName (common)	キューのクラスター名		文字列
filteredMapEvents (common)	Map イベントタイプのコンマ区切りのリスト(filer)。有効な値は INSERT、UPDATE、REMOVE です。		文字列
Persistent (common)	データ永続性の有効化/無効化	true	boolean
subscribeMapEvents (common)	コンシューマーがイベントをマッピングする必要がある場合に設定されます。デフォルトは true です。	true	boolean
subscribeTopicEvents (common)	コンシューマーが TopicEvents,d efault false にサブスクライブすべきかどうかを設定します。	false	boolean
subscribeTopologicalEvents (common)	コンシューマーが TopologicalEvents,d efault false にサブスクライブする必要がある場合に設定します。	false	boolean
wireType (common)	使用する Wire タイプ。デフォルトはバイナリーワイヤです。	BINARY	文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

第59章 チャンクコンポーネント

Camel バージョン 2.15 から利用可能

chunk: コンポーネントは、チャンクテンプレートを使用してメッセージを処理できるようにします。http://www.x5software.com/chunk/examples/ChunkExample?loc=en_USこれは、Templating を使用してリクエストの応答を生成する場合に便利です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
<groupId>org.apache.camel</groupId>
<artifactId>camel-chunk</artifactId>
<version>x.x.x</version> <!-- use the same version as your Camel core version -->
</dependency>
```

59.1. URI 形式

```
chunk:templateName[?options]
```

templateName は、呼び出すテンプレートのクラスパスローカル URI です。

URI にクエリーオプションを追加するには、 ?option=value&option=value&…

59.2. オプション

Chunk コンポーネントにはオプションがありません。

チャンクエンドポイントは、URI 構文を使用して設定します。

```
chunk:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

59.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceUri	リソースへの **必須** パス。プレフィックス：classpath、file、http、ref、または bean. classpath、file、http は、これらのプロトコルを使用してリソースをロードします（classpath はデフォルト）。ref はレジストリーのリソースを検索します。Bean はリソースとして使用される Bean でメソッドを呼び出します。Bean には、ドットの後にメソッド名を指定できます（例：bean:myBean.myMethod）。		文字列

59.2.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
encoding （プロデューサー）	ボディーのエンコーディングの定義		文字列
エクステンション （プロデューサー）	テンプレートのファイル拡張子を定義します。		文字列
themeFolder (producer)	スキャンする themes フォルダーを定義します。		文字列
themeLayer (producer)	説明するテーマレイヤーを定義します。		文字列
themeSubfolder (producer)	スキャンする themes サブフォルダーの定義		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

チャンクコンポーネントは、拡張子 .chtml または _.cxml で themes フォルダーで特定のテンプレートを検索します。別のフォルダーまたは拡張を指定する必要がある場合には、上記の特定のオプションを使用する必要があります。

59.3. チャンクコンテキスト

Camel は Chunk コンテキスト（マップのみ）で交換情報を提供し ます。エクスチェンジは以下 の ように転送されます。

key	value
exchange	エクスチェンジ **自体**。
exchange.properties	エクスチェンジ **プロパティー**。
ヘッダー	In メッセージのヘッダー。
camelContext	Camel コンテキスト。
request	In メッセージ。
ボディー	In メッセージのボディー。
response	Out メッセージ（InOut メッセージ交換パターンのみ）。

59.4. 動的テンプレート

Camel は 2 つのヘッダーを提供し、テンプレートまたはテンプレートコンテンツ自体に異なるリソースの場所を定義できます。これらのヘッダーのいずれかが設定されている場合、Camel はこれを設定されたエンドポイントで使用します。これにより、ランタイム時に動的テンプレートを提供できます。

ヘッダー	Type	説明	バージョンのサポート
ChunkConstants.CHUNK_RESOURCE_URI	文字列	設定されたエンドポイントの代わりに使用するテンプレートリソースの URI。	

ヘッダー	Type	説明	バージョンのサポート
ChunkConstants.CHUNK_TEMPLATE	文字列	設定されたエンドポイントの代わりに使用するテンプレート。	

59.5. サンプル

たとえば、以下のようなものを使用できます。

```
from("activemq:My.Queue").
to("chunk:template");
```

Chunk テンプレートを使用して InOut メッセージエクスチェンジのメッセージの応答（JMSReplyTo ヘッダーがある）を作成するには、以下を行います。

InOnly を使用してメッセージを消費し、別の宛先に送信する場合は、以下を使用できます。

```
from("activemq:My.Queue").
to("chunk:template").
to("activemq:Another.Queue");
```

コンポーネントがヘッダーで動的に使用する必要のあるテンプレートを指定できます。以下に例を示します。

```
from("direct:in").
setHeader(ChunkConstants.CHUNK_RESOURCE_URI).constant("template").
to("chunk:dummy");
```

Chunk コンポーネントオプションの例：

```
from("direct:in").
to("chunk:file_example?
themeFolder=template&themeSubfolder=subfolder&extension=chunk");
```

この例では、Chunk コンポーネントはフォルダーの template/subfolder で file_example.chunk ファイルを検索します。

59.6. メールサンプル

この例では、順序の確認メールに Chunk template を使用します。メールテンプレートは以下のように Chunk でレイアウトされます。

```
Dear {$headers.lastName}, {$headers.firstName}

Thanks for the order of {$headers.item}.

Regards Camel Riders Bookstore
{$body}
```

59.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第60章 クラスコンポーネント

Camel バージョン 2.4 で利用可能

クラス： コンポーネントは Bean を Camel メッセージエクスチェンジにバインドします。これは Bean コンポーネントと同じように動作しますが、レジストリーから Bean を検索する代わりに、クラス名を基に Bean を作成します。

60.1. URI 形式

```
class:className[?options]
```

className は、Bean として作成して使用する完全修飾クラス名です。

60.2. オプション

Class コンポーネントにはオプションがありません。

Class エンドポイントは、URI 構文を使用して設定します。

```
class:beanName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

60.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
beanName	**Required**: 呼び出す Bean の名前を設定します。		文字列

60.2.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
メソッド （プロデューサー）	Bean で呼び出すメソッドの名前を設定します。		文字列
キャッシュ （詳細）	有効にすると、Camel は最初のレジストリールックアップの結果をキャッシュします。レジストリーの Bean がシングルトンスコープとして定義されている場合は、キャッシュを有効にすることができます。	false	boolean
multiParameterArray (advanced)	**非推奨**: メッセージボディーから渡されたパラメーターを処理する方法。true の場合は、メッセージボディーはパラメーターの配列である必要があります。注記： このオプションは Camel によって内部に使用され、エンドユーザー向けの目的ではありません。非推奨注記： このオプションは Camel によって内部的に使用され、エンドユーザー向けの予定はありません。	false	boolean
パラメーター （詳細）	Bean の追加プロパティーの設定に使用されます。		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

60.3. 使用

クラス コンポーネントを Bean コンポーネントとして使用するだけで、完全修飾クラス名を指定します。
たとえば MyFooBean を使用するには、以下のようにする必要があります。

```
from("direct:start").to("class:org.apache.camel.component.bean.MyFooBean").to("mock:result");
```

MyFooBean で呼び出すメソッド（例： hello ）を指定することもできます。

```
from("direct:start").to("class:org.apache.camel.component.bean.MyFooBean?method=hello").to("mock:result");
```

60.4. 作成されたインスタンスでのプロパティーの設定

エンドポイント URI で、たとえば setPrefix メソッドがある場合など、作成されたインスタンスに設

定するプロパティーを指定できます。

```
// Camel 2.17 onwards
from("direct:start")
    .to("class:org.apache.camel.component.bean.MyPrefixBean?bean.prefix=Bye")
    .to("mock:result");

// Camel 2.16 and older
from("direct:start")
    .to("class:org.apache.camel.component.bean.MyPrefixBean?prefix=Bye")
    .to("mock:result");
```

また、# 構文を使用してレジストリーで検索するプロパティーを参照することもできます。

```
// Camel 2.17 onwards
from("direct:start")
    .to("class:org.apache.camel.component.bean.MyPrefixBean?bean.cool=#foo")
    .to("mock:result");

// Camel 2.16 and older
from("direct:start")
    .to("class:org.apache.camel.component.bean.MyPrefixBean?cool=#foo")
    .to("mock:result");
```

id foo で Registry から Bean を検索し、MyPrefixBean クラスの作成されたインスタンスで setCool メソッドを呼び出します。

TIP: クラス コンポーネントがほぼ同じ方法で機能するため、Bean コンポーネントで詳細を確認してください。

60.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

- *Bean*
- *Bean バインディング*
- *Bean インテグレーション*

第61章 CMIS コンポーネント

Camel バージョン 2.11 で利用可能

cmis コンポーネントは Apache Chemistry クライアント API を使用し、CMIS 対応のコンテンツリポジトリーに/からノードを追加/読み取りできるようにします。

61.1. URI 形式

```
cmis://cmisServerUrl[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

61.2. CMIS オプション

CMIS コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
sessionFacadeFactory (common)	カスタム CMISSessionFacadeFactory を使用して CMISSessionFacade インスタンスを作成するには、以下を行います。		CMISSessionFacade Factory
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

CMIS エンドポイントは、URI 構文を使用して設定します。

```
cmis:cmsUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

61.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cmsUrl	cmis リポジトリーに **必要な** URL		文字列

61.2.2. クエリーパラメーター（13 パラメーター）：

Name	説明	デフォルト	Type
pageSize (common)	ページごとに取得するノード数	100	int
readContent (common)	true に設定すると、属性に加えてドキュメントノードの内容が取得されます。	false	boolean
readCount (common)	読み取るノードの最大数		int
repositoryId (common)	使用するリポジトリーの ID。指定がない場合は、最初に利用可能なリポジトリーが使用されます。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
クエリー（コンシューマー）	リポジトリーに対して実行する cmis クエリー。指定されていない場合、コンシューマーはコンテンツツリーを再帰的に反復してコンテンツリポジトリーからすべてのノードを取得します。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
queryMode (producer)	true の場合、メッセージのボディーから cmis クエリーを実行し、結果を返します。それ以外の場合は、cmis リポジトリーにノードが作成されます。	false	boolean
sessionFacadeFactory (advanced)	カスタム CMISSessionFacadeFactory を使用して CMISSessionFacade インスタンスを作成するには、以下を行います。		CMISSessionFacade Factory
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
パスワード （セキュリティー）	cmis リポジトリーのパスワード		文字列
ユーザー名 （セキュリティー）	cmis リポジトリーのユーザー名		文字列

61.3. 用途

61.3.1. プロデューサーによって評価されるメッセージヘッダー

ヘッダー	デフォルト値	説明
CamelCMISFolderPath	/	実行中に使用する現在のフォルダー。指定されていない場合は root フォルダーを使用します。
CamelCMISRetrieveContent	**false**	**queryMode** では、このヘッダーは、プロデューサーがドキュメントノードの内容を取得するよう強制します。
CamelCMISReadSize	**0**	読み取るノードの最大数。
cmis:path	**null**	**CamelCMISFolderPath** が設定されていない場合、この cmis プロパティーからノードのパスを検索し、名前が付けられます。

ヘッダー	デフォルト値	説明
cmis:name	**null**	**CamelCMISFolderPath** が設定されていない場合、この cmis プロパティーからノードのパスを検索し、path になります。
cmis:objectTypeId	**null**	ノードのタイプ
cmis:contentStreamMimeType	**null**	ドキュメントに設定する mimetype

61.3.2. プロデューサー操作のクエリー中に設定されたメッセージヘッダー

ヘッダー	Type	説明
CamelCMISResultCount	**整数**	クエリーから返されるノードの数。

メッセージボディーにはマップのリストが含まれ、マップの各エントリーは cmis プロパティーとその値になります。CamelCMISRetrieveContent ヘッダーが true に設定されている場合、CamelCMISContent キーのあるマップの追加エントリーには、ノードのドキュメントタイプの InputStream が含まれます。

61.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cmis</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョン（2.11 以降）に置き換える必要があります。

61.5. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第62章 CM SMS GATEWAY コンポーネント

Camel バージョン 2.18 から利用可能

Camel-Cm-Sms は [CM SMS Gateway](https://www.cmtelecom.com)の Apache Camel コンポーネントです。

これにより、CM SMS APIをアプリケーション内で camel コンポーネントとして統合できます。

有効なアカウントが必要です。詳細は、CM Telecom を参照してください。

```
cm-sms://sgw01.cm.nl/gateway.ashx?
defaultFrom=DefaultSender&defaultMaxNumberOfParts=8&productToken=xxxxx
```

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
 <groupId>org.apache.camel</groupId>
 <artifactId>camel-cm-sms</artifactId>
 <version>x.x.x</version>
 <!-- use the same version as your Camel core version -->
</dependency>
```

62.1. オプション

CM SMS Gateway コンポーネントにはオプションがありません。

CM SMS Gateway エンドポイントは、URI 構文を使用して設定します。

```
cm-sms:host
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

62.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
host	**必要な** SMS Provider HOST とスキーム		文字列

62.1.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
defaultFrom (producer)	これは送信者の名前です。最大長は 11 文字です。		文字列
defaultMaxNumberOfParts (producer)	複数パートメッセージの場合は、最大数を強制します。メッセージは切り捨てられます。技術的には、ゲートウェイはまずメッセージが 160 文字よりも大きいかどうかを確認します。その場合、メッセージは、これらのパラメーターによって制限されている複数の 153 文字の部分にカットされます。	8	int
productToken (producer)	使用する一意のトークンが **必要です**。		文字列
testConnectionOnStartup (producer)	起動時に SMS Gateway への接続をテストするかどうか。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

62.2. 例

このプロジェクト を試すと、camel-cm-sms が Camel ルートに統合されるかを確認できます。

第63章 COAP コンポーネント

Camel バージョン 2.16 から利用可能

Camel-CoAP は Apache Camel コンポーネントで、マシン間の操作の軽量 REST タイププロトコルである CoAP と連携できます。CoAP, Constrained Application Protocol(Constrained Application Protocol)は、制限されたノードで使用するための特殊な Web 転送プロトコルであり、ネットワークに制約があり、RFC 7252 をベースにしています。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
 <groupId>org.apache.camel</groupId>
 <artifactId>camel-coap</artifactId>
 <version>x.x.x</version>
 <!-- use the same version as your Camel core version -->
</dependency>
```

63.1. オプション

CoAP コンポーネントにはオプションがありません。

CoAP エンドポイントは、URI 構文を使用して設定します。

```
coap:uri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

63.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
uri	CoAP エンドポイントの URI		URI

63.1.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
coapMethodRestrict (consumer)	CoAP コンシューマーがバインドするメソッドのコンマ区切りリスト。デフォルトでは、すべてのメソッド（DELETE、GET、POST、PUT）にバインドします。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

63.2. メッセージヘッダー

Name	タイプ	説明
CamelCoapMethod	**文字列**	ターゲット CoAP サーバー URI を呼び出すときに CoAP プロデューサーが使用する要求メソッド。有効なオプションは DELETE、GET、PING、POST、PUT です。
CamelCoapResponseCode	**文字列**	外部サーバーから送信される CoAP 応答コード。各コードの意味については、RFC 7252 を参照してください。
CamelCoapUri	**文字列**	呼び出しする CoAP サーバーの URI。エンドポイントで直接設定された既存の URI を上書きします。

63.2.1. CoAP プロデューサーリクエストメソッドの設定

以下のルールは、CoAP プロデューサーがターゲット URI を呼び出すために使用するリクエストメソッドを決定します。

1. CamelCoapMethod ヘッダーの値
2. クエリー文字列がターゲット CoAP サーバー URI にある場合は GET。
3. メッセージエクスチェンジボディーが null でない場合 POST。
4. そうでない場合は GET。

第64章 CONSTANT LANGUAGE

Camel バージョン 1.5 で利用可能

Constant Expression Language は、定数文字列を式のタイプとして指定する方法です。

注記

これは、ルートの起動時に一度だけ設定される固定定数値で、ルーティング中に動的な値が必要な場合には使用しないでください。

64.1. 定数オプション

Constant 言語は、以下に示す 1 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
trim	**true**	**ブール値**	値をトリミングして先頭および末尾の空白と改行を削除するかどうか。

64.2. 使用例

Spring DSL の setHeader 要素は、以下のように定数式を使用できます。

```
<route>
  <from uri="seda:a"/>
  <setHeader headerName="theHeader">
    <constant>the value</constant>
  </setHeader>
  <to uri="mock:b"/>
</route>
```

この場合、seda:a Endpoint からのメッセージでは、'theHeader' ヘッダーが定数値 'the value' に設定されます。

Java DSL を使用した場合の同じ例：

```
from("seda:a")
  .setHeader("theHeader", constant("the value"))
  .to("mock:b");
```

64.3. 依存関係

Constant 言語は camel-core の一部です。

第65章 COMETD コンポーネント

Camel バージョン 2.0 で利用可能

cometd: component は、cometd/bayeux プロトコル の jetty 実装と連携するトランスポートです。このコンポーネントを dojo ツールキットライブラリーと組み合わせて使用すると、AJAX ベースのメカニズムを使用して Camel メッセージをブラウザーに直接プッシュできます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cometd</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

65.1. URI 形式

```
cometd://host:port/channelName[?options]
```

channelName は、Camel エンドポイントがサブスクライブできるトピックを表します。

65.2. 例

```
cometd://localhost:8080/service/mychannel
cometds://localhost:8443/service/mychannel
```

ここでの cometds: SSL が設定されたエンドポイントを表します。

65.3. オプション

CometD コンポーネントは、以下に示す 8 個のオプションをサポートします。

Name	説明	デフォルト	Type
sslKeyPassword (security)	SSL 使用時のキーストアのパスワード。		文字列
sslPassword (security)	SSL 使用時のパスワード。		文字列
sslKeystore (security)	キーストアへのパス。		文字列
securityPolicy (security)	承認を制御するためのカスタム設定済みの SecurityPolicy を使用する場合		SecurityPolicy
extensions (common)	受信および送信要求の変更を可能にするカスタム BayeuxServer.Extension の一覧を使用します。		リスト
sslContextParameters (security)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

CometD エンドポイントは、URI 構文を使用して設定します。

```
cometd:host:port/channelName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

65.3.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
host	**必須** のホスト名		文字列
port	**必要な** ホストのポート番号		int

Name	説明	デフォルト	Type
channelName	**Required** The channelName は、Camel エンドポイントがサブスクライブできるトピックを表します。		文字列

65.3.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
allowedOrigins（共通）	crosssOriginFilterOn が true の場合、cross に対応する元のドメイン	*	文字列
baseResource (common)	Web リソースまたはクラスパスのルートディレクトリー。プロトコル file: または classpath: コンポーネントがファイルシステムまたはクラスパスからリソースを読み込むかどうかによって異なります。クラスパスは、リソースが jar にパッケージ化される OSGI デプロイメントに必要です。		文字列
crossOriginFilterOn (common)	true の場合、サーバーはドメイン間のフィルタリングをサポートします。	false	boolean
filterPath (common)	crosssOriginFilterOn が true の場合、filterPath は CrossOriginFilterFilter によって使用されます。		文字列
間隔 (common)	クライアント側のポーリングのタイムアウト（ミリ秒単位）。クライアントが再接続まで待機する時間		int
jsonCommented (common)	true の場合、サーバーは JSON をコメントでラップし、コメントに JSON ラップして生成します。This is a defence a defenceing.（cjacking に対するフェンシングです）。	true	boolean
logLevel (common)	ログレベル。0=none、1=info, 2=debug	1	int
maxInterval (common)	クライアント側の最大ポーリングタイムアウト（ミリ秒単位）。この時点で接続が受信されない場合、クライアントが削除されます。	30000	int
multiFrameInterval (common)	同じブラウザーから複数の接続が検出されると、クライアント側のポーリングタイムアウト。	1500	int
タイムアウト (common)	サーバー側のポーリングのタイムアウト（ミリ秒単位）。これは、サーバーが応答する前に再接続要求を保持する期間です。	240000	int

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sessionHeadersEnabled (consumer)	受信リクエストの Camel メッセージの作成時に、Camel メッセージにサーバーセッションヘッダーを含めるかどうか。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
disconnectLocalSession (producer)	メッセージをチャネルに公開した後にローカルセッションを切断するかどうか。ローカルセッションの切断は、デフォルトで CometD の影響を受けないため、接続解除が必要となります。したがって、メモリーが不足する可能性があります。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

URI にクエリーオプションを追加するには、?option=value&option=value&...

パラメーターを渡す方法の例を、以下に示します。

ファイルの場合（Web アプリケーションディレクトリー -→ cometd://localhost:8080?resourceBase=file./webapp
Web アプリケーションディレクトリーにある webapp リソースの場合には）クラスパスの場合（たと

えば、Web リソースは webapp フォルダー -→ cometd://localhost:8080?resourceBase=classpath:webapp）内にパッケージ化されます。

65.4. AUTHENTICATION

Camel 2.8 から利用可能

カスタム SecurityPolicy および Extension を 'CometdComponent に 設定できます。これにより、ここで説明する認証を使用できます。

65.5. COMETD COMPONENT の SSL の設定

65.5.1. JSSE 設定ユーティリティーの使用

Camel 2.9 の時点で、Competd コンポーネントは Camel JSSE 設定ユーティリティーを介して SSL/TLS 設定をサポートします。 このユーティリティーは、エンドポイントおよびコンポーネントレベルで記述し、設定する必要のあるコンポーネント固有のコードの量を大幅に削減します。 以下の例は、Competd コンポーネントでユーティリティーを使用する方法を示しています。CometdComponent で SSL を設定する必要があります。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

TrustManagersParameters tmp = new TrustManagersParameters();
tmp.setKeyStore(ksp);

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);
scp.setTrustManagers(tmp);

CometdComponent commetdComponent = getContext().getComponent("cometds",
CometdComponent.class);
commetdComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
   <camel:keyManagers
      keyPassword="keyPassword">
    <camel:keyStore
       resource="/users/home/server/keystore.jks"
       password="keystorePassword"/>
   </camel:keyManagers>
   <camel:trustManagers>
    <camel:keyStore
       resource="/users/home/server/keystore.jks"
       password="keystorePassword"/>
   </camel:keyManagers>
  </camel:sslContextParameters>...

  <bean id="cometd" class="org.apache.camel.component.cometd.CometdComponent">
   <property name="sslContextParameters" ref="sslContextParameters"/>
  </bean>
...
  <to uri="cometds://127.0.0.1:443/service/test?baseResource=file:./target/test-
classes/webapp&timeout=240000&interval=0&maxInterval=30000&multiFrameInterval=1500&jsonCom
mented=true&logLevel=2"/>...
```

65.6. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第66章 CONSUL コンポーネント

Camel バージョン 2.18 から利用可能

Consul コンポーネントは、アプリケーションと Consul を統合するコンポーネントです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-consul</artifactId>
  <version>${camel-version}</version>
</dependency>
```

66.1. URI 形式

```
consul://domain?[options]
```

以下の形式で URI にクエリーオプションを追加できます。

```
?option=value&option=value&...
```

66.2. オプション

Consul コンポーネントは、以下に示す 9 個のオプションをサポートします。

Name	説明	デフォルト	Type
URL (common)	Consul エージェントの URL		文字列
Datacenter (common)	データセンター		文字列
sslContextParameters (common)	org.apache.camel.util.jsse.SSLContextParameters インスタンスを使用した SSL 設定。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean

Name	説明	デフォルト	Type
aclToken (common)	Consul で使用する ACL トークンを設定します。		文字列
userName (common)	Basic 認証に使用するユーザー名を設定します。		文字列
パスワード (common)	Basic 認証に使用するパスワードを設定します。		文字列
Configuration (advanced)	エンドポイント間で共有される共通設定を設定します。		ConsulConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Consul エンドポイントは、URI 構文を使用して設定します。

```
consul:apiEndpoint
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

66.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
apiEndpoint	**必須** の API エンドポイント		文字列

66.2.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

66.3. HEADERS

Name	タイプ	説明
Camel Consul Action	文字列	Producer アクション
Camel Consul Key	文字列	アクションが適用されるキー
Camel Consul EventId	文字列	イベント ID（コンシューマーのみ）
Camel Consul EventName	文字列	イベント名（コンシューマーのみ）

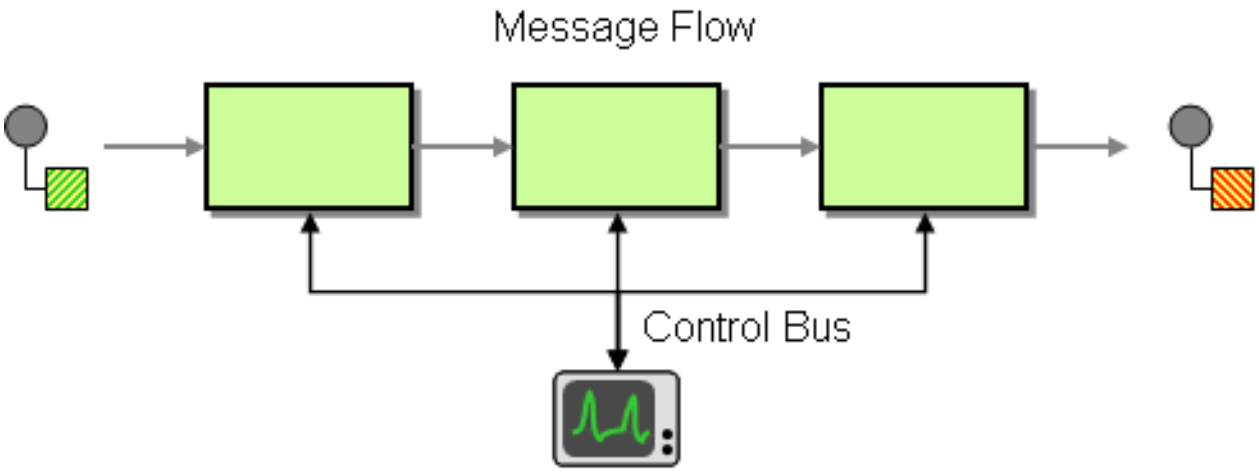
Name	タイプ	説明
Camel Consul EventLTime	Long	イベント LTime
Camel Consul NodeFilter	文字列	Node フィルター
Camel Consul TagFilter	文字列	タグフィルター
Camel Consul SessionFilter	文字列	セッションフィルター
Camel Consul Version	int	データバージョン
Camel Consul Flags	Long	値に関連付けられたフラグ
Camel Consul CreateIndex	Long	エントリーが作成されたタイミングを表す内部インデックス値
Camel Consul LockIndex	Long	このキーがロックで正常に取得された回数
Camel Consul ModifyIndex	Long	このキーを変更した最後のインデックス
Camel Consul Options	オブジェクト	要求に関連付けられたオプション

Name	タイプ	説明
Camel Consul Result	boolean	応答の結果の場合は True
Camel Consul Session	文字列	セッション ID
Camel Consul ValueAsString	boolean	KV エンドポイントで取得された値を string に変換するには、以下を行います。

第67章 バスコンポーネントの制御

Camel バージョン 2.11 で利用可能

EIP パターンの Control Bus を使用すると、統合システムをフレームワーク内から監視および管理できます。



Control Bus を使用してエンタープライズ統合システムを管理します。Control Bus はアプリケーションデータで使用されるメッセージングメカニズムを使用しますが、個別のチャネルを使用して、メッセージフローに関連するコンポーネントの管理に関連するデータを送信します。

Camel では、JMX を使用するか、CamelContext からの Java API を使用するか、org.apache.camel.api.management パッケージから Java API を使用するか、ここに例があるイベント通知機能を使用することができます。

Camel 2.11 以降では、新しい ControlBus コンポーネント を導入し、それに応じて反応するコントロールバスエンドポイントへメッセージを送信できるようになりました。

67.1. CONTROLBUS コンポーネント

Camel 2.11 から利用可能

controlbus: コンポーネントは Control Bus EIP パターンに基づいて Camel アプリケーションを簡単に管理できます。たとえば、メッセージをエンドポイントに送信することで、ルートのライフサイク

ルを制御したり、パフォーマンス統計を収集したりできます。

```
controlbus:command[?options]
```

*ここでの **command** には、使用するコマンドのタイプを特定するための文字列を指定できます。*

67.2. コマンド

コマンド	説明
route	**routeId** および **action** パラメーターを使用してルートを制御する。
言語	メッセージボディーの評価に 使用する言語 を指定できます。評価の結果がある場合は、結果はメッセージボディーに配置されます。

67.3. オプション

Control Bus コンポーネントにはオプションがありません。

Control Bus エンドポイントは、URI 構文を使用して設定します。

```
controlbus:command:language
```

*以下の **path** パラメーターおよびクエリーパラメーターを使用します。*

67.3.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
command	**必要な** コマンドは、route または language のいずれかです。		文字列
言語	メッセージボディーの評価に使用する言語の名前を指定できます。評価の結果がある場合は、結果はメッセージボディーに配置されます。		言語

67.3.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
アクション （プロデューサー）	start、stop、または status のいずれかであるアクションを示します。ルートを開始または停止するか、またはメッセージボディーの出力としてルートのステータスを取得します。Camel 2.11.1 以降の一時停止および再開を使用して、ルートの一時停止または再開を行うことができます。また、Camel 2.11.1 以降では、統計を使用して XML 形式で返されたパフォーマンス静的を得ることができます。routeId オプションを使用して、routeId が定義されていない場合にパフォーマンス統計を取得するルートを定義できます。restart アクションはルートを再起動します。		文字列
async (producer)	コントロールバスタスクを非同期的に実行するかどうか。重要： このオプションを有効にすると、タスクの結果は Exchange に設定されません。これは、タスクを同期的に実行している場合のみ可能です。	false	boolean
loggingLevel (producer)	タスクの完了時や、タスクの処理中に例外が発生した場合のロギングに使用されるロギングレベル。	INFO	LoggingLevel
restartDelay (producer)	ルートの再起動時に使用する遅延（ミリ秒単位）。	1000	int
routeId (producer)	id でルートを指定します。special キーワード current は、現在のルートを示します。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

URI にクエリーオプションを追加するには、 ?option=value&option=value&...

67.4. ROUTE コマンドの使用

route コマンドを使用すると、ルートの開始など、指定のルートで共通のタスクを簡単に実行できます。たとえば、空のメッセージをこのエンドポイントに送信できます。

```
template.sendBody("controlbus:route?routeId=foo&action=start", null);
```

ルートのステータスを取得するには、以下を実行できます。

```
String status = template.requestBody("controlbus:route?routeId=foo&action=status", null,
String.class);
```

67.5. パフォーマンス統計の取得

Camel 2.11.1 から利用可能

これには JMX を有効にする必要があります（デフォルトでは）、ルートごとまたは CamelContext のパフォーマンス静的な情報を取得できます。たとえば、foo という名前のルートの静的を取得するには、以下を行うことができます。

```
String xml = template.requestBody("controlbus:route?routeId=foo&action=stats", null,
String.class);
```

返される静的は XML 形式になります。ManagedRouteMBean で dumpRouteStatsAsXml 操作を使用して JMX から取得できるデータ。

CamelContext 全体の静的を取得するには、以下のように routeId パラメーターを省略します。

```
String xml = template.requestBody("controlbus:route?action=stats", null, String.class);
```

67.6. SIMPLE 言語の使用

Simple 言語をコントロールバスと共に使用して、特定のルートを停止するなどして、以下のメッセージを含む 「controlbus:language:simple」 エンドポイントにメッセージを送信できます。

```
template.sendBody("controlbus:language:simple", "${camelContext.stopRoute('myRoute')}");
```

これは void 操作であるため、結果が返されません。ただし、ルートのステータスが必要な場合は、以下を行うことができます。

```
String status = template.requestBody("controlbus:language:simple",
"${camelContext.getRouteStatus('myRoute')}", String.class);
```

route コマンドを使用してルートのライフサイクルを制御するのが容易になります。language コマンドを使用すると、Groovy などの強力な電源を持つ言語スクリプトを実行するか、Simple 言語を拡張することができます。

たとえば、Camel 自体をシャットダウンするには、以下を行うことができます。

```
template.sendBody("controlbus:language:simple?async=true", "${camelContext.stop()}");
```

async=true を使用して Camel を非同期的に停止します。それ以外の場合は、制御バスコンポーネントに送信されたメッセージをインフライト処理中に Camel を停止しようとします。

ヒント

Groovy などの他の言語を使用することもできます。

第68章 COUCHBASE コンポーネント

Camel バージョン 2.19 から利用可能

couchbase: コンポーネントを使用すると、CouchBase インスタンスをメッセージのプロデューサーまたはコンシューマーとして処理できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-couchbase</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

68.1. URI 形式

```
couchbase:url
```

68.2. オプション

Couchbase コンポーネントにはオプションがありません。

Couchbase エンドポイントは URI 構文を使用して設定されます。

```
couchbase:protocol:hostname:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

68.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
protocol	使用するプロトコルが **必要です**。		文字列
hostname	使用するホスト名が **必要です**。		文字列

Name	説明	デフォルト	Type
port	使用するポート番号	8091	int

68.2.2. クエリーパラメーター（47 パラメーター）：

Name	説明	デフォルト	Type
バケット （共通）	使用するバケット		文字列
キー (common)	使用するキー		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumerProcessedStrategy (consumer)	使用するコンシューマープロセスストラテジーの定義	none	文字列
降順 （コンシューマー）	この操作が降順にするかどうかを定義します。	false	boolean
designDocumentName (consumer)	使用する設計ドキュメント名	beer	文字列
制限 （コンシューマー）	使用する出力制限	-1	int
rangeEndKey (consumer)	終了キーの範囲を定義します。		文字列
rangeStartKey (consumer)	開始キーの範囲を定義します。		文字列
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

Name	説明	デフォルト	Type
skip (consumer)	使用するスキップを定義します。	-1	int
viewName (consumer)	使用するビュー名	brewery_beers	文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
autoStartIdForInserts (producer)	挿入操作を実行する際に自動起動 ID が必要かどうかを定義します。	false	boolean
操作 （プロデューサー）	実行する操作	CCB_PUT	文字列
persistTo (producer)	データを永続化する場所	0	int
producerRetryAttempts (producer)	再試行試行回数を定義します。	2	int
producerRetryPause (producer)	異なる試行間の再試行一時停止を定義します。	5000	int
replicateTo (producer)	データを複製する場所	0	int
startingIdForInsertsFrom (producer)	挿入操作を実行する開始 ID を定義します。		Long
additionalHosts (advanced)	追加ホスト		文字列

Name	説明	デフォルト	Type
maxReconnectDelay (advanced)	再接続時の最大遅延を定義します。	30000	Long
obsPollInterval (advanced)	ポーリング間隔の定義	400	Long
obsTimeout (advanced)	可観測性のタイムアウトの定義	-1	Long
opQueueMaxBlockTime (advanced)	操作がキューに入れられる最大時間を定義します。	10000	Long
opTimeOut (advanced)	操作タイムアウトの定義	2500	Long
readBufferSize (advanced)	バッファーサイズの定義	16384	int
shouldOptimize (advanced)	可能な場合は最適化を使用するかどうかを定義します。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
timeoutExceptionThreshold (advanced)	タイムアウト例外をスローするためのしきい値の定義	998	int
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int

Name	説明	デフォルト	Type
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
パスワード （セキュリティー）	使用するパスワード		文字列
ユーザー名 （セキュリティー）	使用するユーザー名		文字列

第69章 COUCHDB コンポーネント

Camel バージョン 2.11 で利用可能

couchdb: コンポーネントを使用すると、CouchDB インスタンスをメッセージのプロデューサーまたはコンシューマーとして処理できます。軽量 LightCouch API を使用すると、この Camel コンポーネントには以下の機能があります。

- コンシューマーとして、これらはメッセージとしてメッセージとして挿入、更新、および削除、およびパブリッシュするために、コモック変更セットを監視します。
- プロデューサーとして、（CouchDbMethod を DELETE 値で使用して）Camel 2.18 delete を保存し、更新し、そこから couch にドキュメントを保存したり、更新したりできます。
- 複数のインスタンスにわたる複数データベースなど、必要な数だけエンドポイントをサポートすることができます。
- イベントトリガーは削除に対してのみ行われ、挿入/更新のみ（デフォルト）できます。
- sequenceId、ドキュメントリビジョン、ドキュメント ID、および HTTP メソッドタイプに設定されたヘッダー。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-couchdb</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

69.1. URI 形式

```
couchdb:http://hostname[:port]/database?[options]
```

hostname は、実行中の couchdb インスタンスのホスト名です。port はオプションで、指定されていない場合はデフォルトの 5984 に設定されます。

69.2. オプション

CouchDB コンポーネントにはオプションがありません。

CouchDB エンドポイントは URI 構文を使用して設定されます。

```
couchdb:protocol:hostname:port/database
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

69.2.1. パスパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
protocol	データベースとの通信に使用するプロトコルが **必要です**。		文字列
hostname	実行中の couchdb インスタンスに **必要な** ホスト名		文字列
port	実行中の couchdb インスタンスのポート番号	5984	int
database	使用するデータベースに **必要な** 名前		文字列

69.2.2. クエリーパラメーター（12 パラメーター）：

Name	説明	デフォルト	Type
createDatabase (common)	データベースが存在しない場合は作成します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
削除（コンシューマー）	ドキュメントの削除はイベントとして公開されます。	true	boolean
ハートビート（コンシューマー）	ソケットをミリ秒単位で維持するために空のメッセージを送信する頻度	30000	Long
以降（コンシューマー）	指定の更新シーケンスの直後に変更の追跡を開始します。デフォルトの null は、最新のシーケンスから監視を開始します。		文字列
スタイル（コンシューマー）	changes アレイで返されるリビジョンの数を指定します。default の main_only は、現在の進化版のみを返します。all_docs はすべてのリーフリビジョンを返します（競合を含む）。	main_only	文字列
更新（コンシューマー）	ドキュメントの挿入/更新はイベントとして公開されます。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
パスワード（セキュリティー）	認証されたデータベースのパスワード		文字列
ユーザー名（セキュリティー）	認証されたデータベースの場合のユーザー名		文字列

69.3. HEADERS

以下のヘッダーは、メッセージトランスポート中にエクスチェンジに設定されます。

プロパティー	値
CouchDbDatabase	メッセージを取得したデータベース
CouchDbSeq	更新/削除のメッセージセットシーケンス番号
CouchDbId	couchdb ドキュメント ID
CouchDbRev	couchdb ドキュメントのリビジョン
CouchDbMethod	メソッド（削除/更新）

メッセージが受信されると、ヘッダーはコンシューマーによって設定されます。また、プロデューサーは、挿入/更新が完了したらダウンストリームプロセッサーのヘッダーも設定します。プロデューサーの前に設定されたヘッダーは無視されます。たとえば、CouchDbId をヘッダーとして設定すると、これは挿入の ID としては使用されません。

69.4. メッセージボディー

コンポーネントは、メッセージボディーを、挿入するドキュメントとして使用します。ボディーが String のインスタンスである場合、挿入前に GSON オブジェクトにマーシャルされます。つまり、文字列が有効である JSON であるか、挿入/更新は失敗します。ボディーが com.google.gson.JsonElement のインスタンスである場合、そのまま挿入されます。それ以外の場合は、プロデューサーは、サポート対象外のボディータイプの例外をスローします。

69.5. サンプル

たとえば、ローカルで実行している CouchDB インスタンスからすべての挿入、更新、および削除をポート 9999 で使用する場合、以下を使用できます。

```
from("couchdb:http://localhost:9999").process(someProcessor);
```

削除のみに興味がある場合は、以下を使用できます。

```
from("couchdb:http://localhost:9999?updates=false").process(someProcessor);
```

メッセージをドキュメントとして挿入したい場合は、エクスチェンジの本文が使用されます。

```
from("someProducingEndpoint").process(someProcessor).to("couchdb:http://localhost:9999"
)
```

第70章 CASSANDRA CQL COMPONENT

Camel バージョン 2.15 から利用可能

Apache Cassandra はオープンソースのNoSQLデータベースで、企業のハードウェア上で大量のデータを処理するように設計されています。AmazonのDynamoDBと同様に、Cassandraにはピアツーピアとマスターレスアーキテクチャがあり、単一障害点や高可用性を回避します。Google の BigTable と同様に、Cassandra データは Thrift RPC API または CQL と呼ばれる SQL のような API 経由でアクセスできる列ファミリーを使用して構成されています。

このコンポーネントは、CQL3 API（Trift API ではない）を使用して Cassandra 2.0 以降を統合することを目指しています。DataStax が提供する Cassandra Java Driver をベースにしています。

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cassandraql</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

70.1. URI 形式

エンドポイントは Cassandra 接続を開始したり、既存の接続を使用したりできます。

URI	説明
cql:localhost/keyspace	単一ホスト、デフォルトポート、テストには通常
cql:host1,host2/keyspace	マルチホスト、デフォルトポート
cql:host1,host2:9042/keyspace	マルチホスト、カスタムポート
cql:host1,host2	デフォルトのポートおよびキースペース
cql:bean:sessionRef	提供されるセッション参照

URI	説明
cql:bean:clusterRef/keyspace	提供されるクラスター参照

Cassandra 接続（SSL オプション、プーリングオプション、負荷分散ポリシー、再試行ポリシーなど）を調整するには、独自の Cluster インスタンスを作成し、Camel エンドポイントに付与します。

70.2. CASSANDRA オプション

Cassandra CQL コンポーネントにはオプションがありません。

Cassandra CQL エンドポイントは、URI 構文を使用して設定します。

```
cql:beanRef:hosts:port/keyspace
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

70.2.1. パスパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
beanRef	beanRef は bean:id を使用して定義されます。		文字列
hosts	hostname(s)cassansdra server(s)複数のホストをコンマで区切ることができます。		文字列
port	cassansdra サーバーのポート番号		整数
keyspace	使用するキースペース		文字列

70.2.2. クエリーパラメーター（29 パラメーター）：

Name	説明	デフォルト	Type
Cluster (common)	Cluster インスタンスを使用する場合（通常はこのオプションを使用しません）。		Cluster

Name	説明	デフォルト	Type
clusterName (common)	クラスター名		文字列
consistencyLevel (common)	使用する整合性レベル		ConsistencyLevel
CQL (common)	実行する CQL クエリー。CamelCqlQuery キーでメッセージヘッダーで上書きできます。		文字列
loadBalancingPolicy (common)	特定の LoadBalancingPolicy の使用		文字列
パスワード (common)	セッション認証のパスワード		文字列
prepareStatements (common)	PreparedStatements または regular Statements を使用するかどうか	true	boolean
resultSetConversionStrategy (common)	ResultSet をメッセージのボディー ALL、ONE、LIMIT_10、LIMIT_100... に変換するロジックを実装するカスタムクラスを使用するには、以下を実行します。		文字列
セッション （共通）	セッションインスタンスを使用する場合（通常はこのオプションを使用しません）。		Session
username (common)	セッション認証のユーザー名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean

Name	説明	デフォルト	Type
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

70.3. メッセージ

70.3.1. 受信メッセージ

Camel Cassandra エンドポイントは、クエリーパラメーターとして CQL ステートメントにバインドされる単純なオブジェクト（Object または Object[] または Collection<Object>）を想定します。メッセージボディーが null または空の場合、CQL クエリーはバインディングパラメーターなしで実行されます。

ヘッダー：

-

CamelCqlQuery（任意、String または RegularStatement）: CQL はプレーンな String としてクエリーするか、QueryBuilder を使用してビルドします。

70.3.2. 送信メッセージ

Camel Cassandra エンドポイントは、 resultSetConversionStrategy に応じて 1 つまたは複数の Cassandra Row オブジェクトを生成します。

- List<Row& gt; if resultSetConversionStrategy が ALL または LIMIT_[0-9]+
- resultSetConversionStrategy が ONEの場合、単一の Row'
- resultSetConversionStrategy が ResultSetConversionStrategy のカスタム実装である場合、それ以外のものはすべて ResultSetConversionStrategyのカスタム実装です。

70.4. リポジトリー

Cassandra は、べき等および集約 EIP のメッセージキーまたはメッセージを保存するために使用できます。

Cassandra は、ユースケースをキューに入れるのに最適なツールではない可能性がありますが、Cassandra のアンチパターンキューやデータセットなどのキューの読み取りが最適ではない可能性があり ます。これらのテーブルには、LeveledCompaction および small GC grace 設定を使用して、tombstoned 行を迅速に削除することが推奨されます。

70.5. べき等リポジトリー

NamedCassandraIdempotentRepository は、以下のように Cassandra テーブルにメッセージキーを保存します。

CAMEL_IDEMPOTENT.cql

```
CREATE TABLE CAMEL_IDEMPOTENT (
```

```
  NAME varchar,   -- Repository name
  KEY varchar,    -- Message key
  PRIMARY KEY (NAME, KEY)
) WITH compaction = {'class':'LeveledCompactionStrategy'}
  AND gc_grace_seconds = 86400;
```

このリポジトリー実装は、軽量トランザクション（Compare および Set とも呼ばれます）を使用し、Cassandra 2.0.7 以降が必要になります。

または、CassandraIdempotentRepository には NAME 列がなく、別のデータモデルを使用するように拡張できます。

オプション	デフォルト	説明
表	CAMEL_IDEMPOTENT	テーブル名
pkColumns	名前,' キー'	プライマリーキー列
name		リポジトリー名（NAME 列に使用する値）
ttl		キーの有効期間
writeConsistencyLevel		キーの挿入/削除に使用される整合性レベル：ANY、ONE、TWO、QUORUM、LOCAL_QUORUM...
readConsistencyLevel		キーの読み取り/確認に使用される整合性レベル：ONE、TWO、QUORUM、LOCAL_QUORUM...

70.6. 集約リポジトリー

NamedCassandraAggregationRepository は、以下のような Cassandra テーブルに相関キーによってエクスチェンジを保存します。

CAMEL_AGGREGATION.cql

```
CREATE TABLE CAMEL_AGGREGATION (
  NAME varchar,        -- Repository name
  KEY varchar,         -- Correlation id
  EXCHANGE_ID varchar, -- Exchange id
```

```
    EXCHANGE blob,      -- Serialized exchange
    PRIMARY KEY (NAME, KEY)
  ) WITH compaction = {'class':'LeveledCompactionStrategy'}
    AND gc_grace_seconds = 86400;
```

または、CassandraAggregationRepository には NAME 列がなく、別のデータモデルを使用するように拡張できます。

オプション	デフォルト	説明
表	**CAMEL_AGGREGATION**	テーブル名
pkColumns	**NAME,KEY**	プライマリーキー列
exchangeIdColumn	**EXCHANGE_ID**	エクスチェンジ ID 列
exchangeColumn	**EXCHANGE**	コンテンツ列の交換
name		リポジトリー名（**NAME** 列に使用する値）
ttl		Exchange time to live
writeConsistencyLevel		交換の挿入/削除に使用される整合性レベル：**ANY**、**ONE**、**TWO**、**QUORUM**、**LOCAL_QUORUM**...
readConsistencyLevel		エクスチェンジの読み取り/確認に使用される整合性レベル：**ONE**、**TWO**、**QUORUM**、**LOCAL_QUORUM**...

第71章 CRYPTO(JCE)コンポーネント

Camel バージョン 2.3 の時点で利用可能

Camel 暗号化エンドポイントと Java の Cryptographic エクステンションを使用すると、Exchange 用のデジタル署名を簡単に作成できます。Camel は、エクスチェンジのワークフローの 1 つの部分でエクスチェンジの署名を作成するために使用する柔軟なエンドポイントのペアを提供し、後続のワークフローで署名を検証します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-crypto</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

71.1. はじめに

デジタル署名は、非対称暗号方式を使用してメッセージの署名を行います。（未検証）高レベルから、アルゴリズムは、1 つの鍵で暗号化した特別なプロパティーと必須の鍵のペアを使用します。これは、一方の鍵で暗号化したデータを他の鍵とのみ復号できます。1 つは秘密鍵だけで、メッセージが「署名」するために使用されます。一方、もう一方の公開鍵は、署名されたメッセージを検証することを検討している人に共有されます。メッセージは秘密鍵を使用してメッセージのダイジェストを暗号化することで署名されます。この暗号化されたダイジェストはメッセージと共に送信されます。反対に、検証子はメッセージダイジェストを再計算し、パブリックキーを使用して署名のダイジェストを復号化します。両方のダイジェストが検証者と一致する場合は、秘密鍵のホルダーのみが署名を作成できます。

Camel は Java Cryptographic Extension の Signature サービスを使用して、交換署名の作成に必要なすべての大きな暗号化状態を実行します。以下は、暗号、メッセージダイジェスト、デジタル署名、および JCE で活用する方法を説明するための優れたリソースです。

- Bruce Schneier の Applied Cryptography
- Welcomed Hook による Java による暗号化の開始
- 発見された Wikipedia Digital_signatures

71.2. URI 形式

説明したように、Camel は署名を作成および検証するための暗号エンドポイントのペアを提供します。

```
crypto:sign:name[?options]
crypto:verify:name[?options]
```

- crypto:sign は署名を作成し、定数 org.apache.camel.component.crypto.DigitalSignatureConstants.SIGNATURE で指定された Header キーに保存します。"CamelDigitalSignature".

- crypto:verify はこのヘッダーの内容に読み込まれ、検証の計算を行います。

適切に機能させるには、署名と検証のプロセスで共有する鍵のペアが必要で、PrivateKey を要求し、PublicKey （または 1 つが含まれる 証明書 ）を検証する必要があります。JCE を使用すると、これらのキーペアを生成することは非常に簡単ですが、通常は KeyStore を使用してキーを保管して共有することが最も安全です。DSL はキーの指定方法に非常に柔軟で、多くのメカニズムを提供します。

crypto:sign エンドポイントは通常 1 つのルートで定義され、複雑なエンドポイントを別のルートで定義します。ただし、このエンドポイントはもう一方のルートの後に表示されます。これは、署名と検証の両方で同じ設定が必要であることを示唆せずに行われます。

71.3. オプション

Crypto(JCE)コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有デジタル署名Configurationを設定として使用する場合		digitalSignature の設定
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Crypto(JCE)エンドポイントは、URI 構文を使用して設定します。

```
crypto:cryptoOperation:name
```

*以下の **path** パラメーターおよびクエリーパラメーターを使用します。*

71.3.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
cryptoOperation	**required** Set the Crypto operation from that provided after the crypto scheme in the endpoint uri（例： crypto:sign は署名を操作として設定します）。		CryptoOperation
name	***必須***。この操作の論理名。		文字列

71.3.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
アルゴリズム（プロデューサー）	署名側の使用する必要のあるアルゴリズムの JCE 名を設定します。	SHA1WithDSA	文字列
エイリアス（プロデューサー）	キーの KeyStore のクエリーに使用するエイリアスを設定し、エクスチェンジの署名および検証に使用される java.security.cert.Certificate 証明書をリンクします。この値は、メッセージヘッダーリンク org.apache.camel.component.crypto.DigitalSignatureConstantsKEYSTORE_ALIAS を介してランタイム時に指定できます。		文字列
certificateName (producer)	レジストリーで fond できる PrivateKey の参照名を設定します。		文字列
キーストア（プロデューサー）	キーおよびエクスチェンジの署名および検証に使用できるキーが含まれる KeyStore を設定します。KeyStore は通常、Route 定義で指定されるエイリアスか、またはメッセージヘッダー CamelSignatureKeyStoreAlias を介して動的に使用されます。エイリアスが指定されておらず、キーストアにエントリーが 1 つしかない場合は、この単一エントリーが使用されます。		KeyStore
keystoreName (producer)	レジストリーで fond できるキーストアの参照名を設定します。		文字列

Name	説明	デフォルト	Type
privateKey (producer)	エクスチェンジの署名に使用する PrivateKey を設定する		PrivateKey
privateKeyName (producer)	レジストリーで fond できる PrivateKey の参照名を設定します。		文字列
プロバイダー （プロデューサー）	設定された署名アルゴリズムを提供するセキュリティープロバイダーの ID を設定します。		文字列
publicKeyName (producer)	コンテキストの変更時に解決する必要のある参照		文字列
secureRandomName (producer)	レジストリーで fond できる SecureRandom の参照名を設定します。		文字列
signatureHeaderName (producer)	base64 でエンコードされた署名を保存するために使用されるメッセージヘッダーの名前を設定します。デフォルトは 'CamelDigitalSignature' です。		文字列
bufferSize (advanced)	Exchange ペイロードデータで読み取るために使用されるバッファーサイズを設定します。	2048	整数
証明書 （詳細）	ペイロードに基づいてエクスチェンジの署名を検証するために使用する証明書を設定します。		証明書
clearHeaders (advanced)	署名および検証後に署名固有のヘッダーを消去するかどうかを決定します。デフォルトは true で、キーやパスワードが設定されていないとエスケープするなど、重要なプライベート情報として極端に設定しないといけません。	true	boolean
keyStoreParameters (advanced)	指定された KeyStoreParameters に基づいてエクスチェンジの署名および検証に使用できるキーおよび証明書が含まれる KeyStore を設定します。KeyStore は通常、Route 定義で指定されるエイリアスか、またはメッセージヘッダー CamelSignatureKeyStoreAlias を介して動的に使用されます。エイリアスが指定されておらず、キーストアにエントリーが 1 つしかない場合は、この単一エントリーが使用されます。		KeyStoreParameters
publicKey (advanced)	エクスチェンジの署名の検証に使用する PublicKey を設定します。		PublicKey
secureRandom (advanced)	署名サービスの初期化に使用する SecureRandom の設定		SecureRandom

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
パスワード （セキュリティー）	KeyStore でエイリアスされた PrivateKey にアクセスするために使用されるパスワードを設定します。		文字列

71.4. 使用

71.4.1. raw キー

エクスチェンジに署名および検証する最も基本的な方法は、以下のように KeyPair を使用することです。

キーへの参照を使用して Spring XML エクステンション を使用して、同じことを実現できます。

71.4.2. キーストアおよびエイリアス。

JCE は、秘密鍵と証明書のペアを格納するための非常に多目的なキーストアの概念を提供し、暗号化とパスワードを保護します。これらは、エイリアスを取得 API に適用して取得できます。キーストアに鍵と証明書を取得する方法は複数あります。ほとんどの場合、これは外部の「keytool」アプリケーションで行います。これ は、keytool を使用して自己署名証明書と秘密鍵を持つ KeyStore を作成するのに適した例です。

この例では、「bob」がキーおよび証明書エイリアスが設定されたキーストアを使用します。キーストアとキーのパスワードは「letmein」です。

以下は、Fluent ビルダーでキーストアを使用する方法を示しています。また、キーストアのロードおよび初期化方法も示しています。

Spring でも、実際のキーストアインスタンスを検索するのに ref が使用されます。

71.4.3. JCE プロバイダーおよびアルゴリズムの変更

署名アルゴリズムまたはセキュリティープロバイダーの変更は、名前を指定するだけで簡単です。また、選択したアルゴリズムと互換性のあるキーを使用する必要があります。

または

71.4.4. 署名メッセージヘッダーの変更

署名の保存に使用されるメッセージヘッダーを変更することが望ましい場合があります。以下のように、ルート定義で異なるヘッダー名を指定できます。

または

71.4.5. buffersize の変更

バッファーのサイズを更新する必要があります。

または

71.4.6. キーを動的に指定。

Recipient list または同様の EIP を使用する場合、エクスチェンジの受信者は動的に異なる可能性があります。すべての受信者で同じキーを使用することは、現実的でも望ましくないこともあります。署名鍵は、交換ごとに動的に指定できるので便利です。その後、署名前にそのターゲット受信者のキーでエクスチェンジを動的に強化できます。この署名メカニズムを容易にするために、以下のメッセージヘッダーを使用してキーを動的に指定できます。

- *Exchange.SIGNATURE_PRIVATE_KEY, "CamelSignaturePrivateKey"*
- *Exchange.SIGNATURE_PUBLIC_KEY_OR_CERT, "CamelSignaturePublicKeyOrCert"*

または

キーストアエイリアスを動的に指定することは適切です。この場合もエイリアスをメッセージヘッダーに指定できます。

- *Exchange.KEYSTORE_ALIAS, "CamelSignatureKeyStoreAlias"*

または

ヘッダーは以下のように設定されます。

```
Exchange unsigned = getMandatoryEndpoint("direct:alias-sign").createExchange();
unsigned.getIn().setBody(payload);
unsigned.getIn().setHeader(DigitalSignatureConstants.KEYSTORE_ALIAS, "bob");
unsigned.getIn().setHeader(DigitalSignatureConstants.KEYSTORE_PASSWORD,
"letmein".toCharArray());
template.send("direct:alias-sign", unsigned);
Exchange signed = getMandatoryEndpoint("direct:alias-sign").createExchange();
signed.getIn().copyFrom(unsigned.getOut());
signed.getIn().setHeader(KEYSTORE_ALIAS, "bob");
template.send("direct:alias-verify", signed);
```

71.5. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第72章 CRYPTO CMS コンポーネント

Camel バージョン 2.20 で利用可能

暗号化メッセージ構文(CMS) は、メッセージの署名および暗号化に適切に確立された標準です。Apache Crypto CMS コンポーネントは、この規格の以下の部分をサポートします。* Content Type "Enveloped Data" with Key Transport(asymmetric key)、* Content Type "Signed Data"CMS Enveloped Data インスタンスを作成し、CMS Enveloped Data インスタンスを復号化し、CMS 署名データインスタンスを作成して、CMS 署名データインスタンスを検証できます。

コンポーネントは Bouncy Castle ライブラリー bcprov-jdk15on および bcpkix-jdk15on を使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-crypto-cms</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

このコンポーネントのエンドポイントを呼び出す前に、アプリケーションに Bouncy Castle セキュリティープロバイダーを登録することが推奨されます。

```
Security.addProvider(new BouncyCastleProvider());
```

Bouncy Castle セキュリティープロバイダーが登録されていない場合は、Crypto CMS コンポーネントはプロバイダーを登録します。

72.1. オプション

Crypto CMS コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
signedDataVerifier Configuration (advanced)	検証操作の uri パラメーターを決定する shared SignedDataVerifierConfiguration を設定します。		SignedDataVerifier Configuration
envelopedDataDecryptor Configuration (advanced)	共有 EnvelopedDataDecryptorConfiguration を設定するには、復号化操作の uri パラメーターを決定します。		EnvelopedDataDecryptor 設定
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Crypto CMS エンドポイントは、URI 構文を使用して設定します。

```
crypto-cms:cryptoOperation:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

72.1.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
cryptoOperation	**required** Set the Crypto operation from that provided after the crypto scheme in the endpoint uri（たとえば crypto-cms:sign は署名を操作として設定します）。使用できる値は、Sign、verify、crypting、または decrypt です。		CryptoOperation
name	***必須***: URI の name 部分を選択して、camel コンテキスト内の異なる署名者/検証者/暗号化/暗号エンドポイント間を区別できます。		文字列

72.1.2. クエリーパラメーター（15 パラメーター）：

Name	説明	デフォルト	Type
keyStore (common)	署名側の秘密鍵、検証元公開鍵、公開鍵、公開鍵、操作に応じて復号化する秘密鍵が含まれるキーストア。このパラメーターまたはパラメーター「keyStoreParameters」を使用します。		KeyStore
keyStoreParameters (common)	操作に応じて、署名側の秘密鍵、検証公開鍵、公開鍵の暗号化、秘密鍵、秘密鍵の復号化が含まれるキーストア。このパラメーターまたはパラメーター「keystore」を使用します。		KeyStoreParameters
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
パスワード （暗号解除）	秘密鍵のパスワードを設定します。キーストアのすべての秘密鍵に同じパスワードがあることを前提とします。設定されていない場合は、KeyStoreParameters で指定されたキーストアパスワードによって秘密鍵のパスワードが指定されていることが想定されます。		char[]
fromBase64 (decrypt_verify)	true の場合、CMS メッセージはベース 64 でエンコードされ、処理中にデコードする必要があります。デフォルト値は false です。	false	ブール値
contentEncryptionAlgorithm (encrypt)	暗号化アルゴリズム（DESede/CBC/PKCS5Padding など）。その他の可能な値：DESede/CBC/PKCS5Padding、AES/CBC/PKCS5Padding、Camellia/CBC/PKCS5Padding、CAST5/CBC/PKCS5Padding		文字列
originatorInformation Provider (encrypt)	originator 情報のプロバイダー。See https://tools.ietf.org/html/rfc5652section-6.1.デフォルト値は null です。		OriginatorInformation Provider
Recipient (encrypt)	Recipient Info: インターフェース org.apache.camel.component.crypto.cms.api.TransRecipientInfo を実装する Bean への参照		リスト

Name	説明	デフォルト	Type
secretKeyLength (encrypt)	コンテンツの暗号化に使用される秘密対称鍵のキーの長さ。指定した content-encryption アルゴリズムで異なるサイズの鍵が許可されている場合にのみ使用します。contentEncryptionAlgorithm=AES/CBC/PKCS5Padding または Camellia/CBC/PKCS5Padding または 128; contentEncryptionAlgorithm=DESede/CBC/PKCS5Padding 場合は、192、128 です。強力な暗号化が有効な場合は、AES/CBC/PKCS5Padding および Camellia/CBC/PKCS5Padding もキーの長さ 192 および 256 を追加できます。		int
unprotectedAttributesGeneratorProvider (encrypt)	保護されていない属性のジェネレーターのプロバイダー。デフォルト値は null で、保護されていない属性が Enveloped Data オブジェクトに追加されないことを意味します。See https://tools.ietf.org/html/rfc5652section-6.1.		AttributesGenerator Provider
toBase64 (encrypt_sign)	Signed Data インスタンスまたは Enveloped Data インスタンスがベース 64 エンコードされているかどうかを示します。デフォルト値は false です。	false	ブール値
includeContent (sign)	署名済みコンテンツを Signed Data インスタンスに含めるかどうかを示します。false の場合、デタッチされた Signed Data インスタンスがヘッダー CamelCryptoCmsSignedData に作成されます。	true	ブール値
署名側の（署名 ）	署名側の情報：org.apache.camel.component.crypto.cms.api.SignerInfo を実装する Bean への参照		リスト
signedDataHeaderBase64 (verify)	ヘッダー CamelCryptoCmsSignedData の値が base64 でエンコードされるかどうかを示します。デフォルト値は false です。デタッチされた署名にのみ関連します。デタッチされた署名の場合、ヘッダーに Signed Data オブジェクトが含まれます。	false	ブール値
verifySignaturesOfAll Signers (verify)	true の場合、Signed Data オブジェクトに含まれるすべての署名者の署名が検証されます。false の場合、署名側の情報が指定された証明書のいずれかに一致する署名 1 つのみが検証されます。デフォルト値は true です。	true	ブール値

72.2. ENVELOPED DATA

crypto-cms:encypt エンドポイントは、通常 1 つのルートで定義され、別のルートでは

complimentary crypto-cms:decrypt で定義されます。ただし、これは他のエンドポイントの後に表示される例を簡単にするためです。

以下の例は、Enveloped Data メッセージを作成する方法と、Enveloped Data メッセージを復号化する方法を示しています。

Java DSL の基本例

```
import org.apache.camel.util.jsse.KeyStoreParameters;
import org.apache.camel.component.crypto.cms.crypt.DefaultKeyTransRecipientInfo;
...
KeyStoreParameters keystore  = new KeyStoreParameters();
keystore.setType("JCEKS");
keystore.setResource("keystore/keystore.jceks);
keystore.setPassword("some_password"); // this password will also be used for accessing the private key if not specified in the crypto-cms:decrypt endpoint

DefaultKeyTransRecipientInfo recipient1 = new DefaultKeyTransRecipientInfo();
recipient1.setCertificateAlias("rsa"); // alias of the public key used for the encryption
recipient1.setKeyStoreParameters(keystore);

simpleReg.put("keyStoreParameters", keystore); // register keystore in the registry
simpleReg.put("recipient1", recipient1); // register recipient info in the registry

from("direct:start")
   .to("crypto-cms:encrypt://testencrpyt?toBase64=true&recipient=#recipient1&contentEncryptionAlgorithm=DESede/CBC/PKCS5Padding&secretKeyLength=128")
   .to("crypto-cms:decrypt://testdecrypt?fromBase64=true&keyStoreParameters=#keyStoreParameters")
   .to("mock:result");
```

Spring XML の基本例

```
  <keyStoreParameters xmlns="http://camel.apache.org/schema/spring"
      id="keyStoreParameters1" resource="./keystore/keystore.jceks"
      password="some_password" type="JCEKS" />
   <bean id="recipient1"
      class="org.apache.camel.component.crypto.cms.crypt.DefaultKeyTransRecipientInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="certificateAlias" value="rsa" />
   </bean>
...
   <route>
      <from uri="direct:start" />
      <to uri="crypto-cms:encrypt://testencrpyt?toBase64=true&amp;recipient=#recipient1&amp;contentEncryptionAlgorithm=DESede/CBC/PKCS5Padding&amp;secretKeyLength=128" />
      <to uri="crypto-cms:decrypt://testdecrypt?
```

```
fromBase64=true&amp;keyStoreParameters=#keyStoreParameters1" />
        <to uri="mock:result" />
    </route>
```

Java DSL の 2 つの推奨事項

```
import org.apache.camel.util.jsse.KeyStoreParameters;
import org.apache.camel.component.crypto.cms.crypt.DefaultKeyTransRecipientInfo;
...
KeyStoreParameters keystore  = new KeyStoreParameters();
keystore.setType("JCEKS");
keystore.setResource("keystore/keystore.jceks);
keystore.setPassword("some_password"); // this password will also be used for accessing
the private key if not specified in the crypto-cms:decrypt endpoint

DefaultKeyTransRecipientInfo recipient1 = new DefaultKeyTransRecipientInfo();
recipient1.setCertificateAlias("rsa"); // alias of the public key used for the encryption
recipient1.setKeyStoreParameters(keystore);

DefaultKeyTransRecipientInfo recipient2 = new DefaultKeyTransRecipientInfo();
recipient2.setCertificateAlias("dsa");
recipient2.setKeyStoreParameters(keystore);

simpleReg.put("keyStoreParameters", keystore); // register keystore in the registry
simpleReg.put("recipient1", recipient1); // register recipient info in the registry

from("direct:start")
    .to("crypto-cms:encrypt://testencrpyt?
toBase64=true&recipient=#recipient1&recipient=#recipient2&contentEncryptionAlgorithm=DE
Sede/CBC/PKCS5Padding&secretKeyLength=128")
    //the decryptor will automatically choose one of the two private keys depending which one
is in the decryptor keystore
    .to("crypto-cms:decrypt://testdecrypt?
fromBase64=true&keyStoreParameters=#keyStoreParameters")
    .to("mock:result");
```

Spring XML での 2 つの推奨事項

```
  <keyStoreParameters xmlns="http://camel.apache.org/schema/spring"
      id="keyStoreParameters1" resource="./keystore/keystore.jceks"
      password="some_password" type="JCEKS" />
    <bean id="recipient1"
      class="org.apache.camel.component.crypto.cms.crypt.DefaultKeyTransRecipientInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="certificateAlias" value="rsa" />
    </bean>
    <bean id="recipient2"
      class="org.apache.camel.component.crypto.cms.crypt.DefaultKeyTransRecipientInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="certificateAlias" value="dsa" />
    </bean>
```

```
...
    <route>
        <from uri="direct:start" />
        <to uri="crypto-cms:encrypt://testencrpyt?
toBase64=true&amp;recipient=#recipient1&amp;recipient=#recipient2&amp;contentEncryptionAlgorithm
=DESede/CBC/PKCS5Padding&amp;secretKeyLength=128" />
        <!-- the decryptor will automatically choose one of the two private keys depending which
one is in the decryptor keystore -->
        <to uri="crypto-cms:decrypt://testdecrypt?
fromBase64=true&amp;keyStoreParameters=#keyStoreParameters1" />
         <to uri="mock:result" />
    </route>
```

72.3. 署名付きデータ

crypto-cms:sign エンドポイントは、通常 1 つのルートで定義され、別のルートでは complimentary crypto-cms:verify で定義されますが、このエンドポイントはもう一方のルートの後に表示されます。

以下の例は、Signed Data メッセージを作成する方法と、Signed Data メッセージを検証する方法を示しています。

Java DSL の基本例

```
import org.apache.camel.util.jsse.KeyStoreParameters;
import org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo;
...
KeyStoreParameters keystore  = new KeyStoreParameters();
keystore.setType("JCEKS");
keystore.setResource("keystore/keystore.jceks);
keystore.setPassword("some_password"); // this password will also be used for accessing
the private key if not specified in the signerInfo1 bean

//Signer Information, by default the following signed attributes are included: contentType,
signingTime, messageDigest, and cmsAlgorithmProtect; by default no unsigned attribute is
included.
// If you want to add your own signed attributes or unsigned attributes, see methods
DefaultSignerInfo.setSignedAttributeGenerator and
DefaultSignerInfo.setUnsignedAttributeGenerator.
DefaultSignerInfo signerInfo1 = new DefaultSignerInfo();
signerInfo1.setIncludeCertificates(true); // if set to true then the certificate chain of the private
key will be added to the Signed Data object
signerInfo1.setSignatureAlgorithm("SHA256withRSA"); // signature algorithm; attention, the
signature algorithm must fit to the signer private key.
signerInfo1.setPrivateKeyAlias("rsa"); // alias of the private key used for the signing
signerInfo1.setPassword("private_key_pw".toCharArray()); // optional parameter, if not set
then the password of the KeyStoreParameters will be used for accessing the private key
signerInfo1.setKeyStoreParameters(keystore);
```

```
simpleReg.put("keyStoreParameters", keystore); //register keystore in the registry
simpleReg.put("signer1", signerInfo1); //register signer info in the registry

from("direct:start")
   .to("crypto-cms:sign://testsign?signer=#signer1&includeContent=true&toBase64=true")
   .to("crypto-cms:verify://testverify?
keyStoreParameters=#keyStoreParameters&fromBase64=true"")
   .to("mock:result");
```

Spring XML の基本例

```
  <keyStoreParameters xmlns="http://camel.apache.org/schema/spring"
      id="keyStoreParameters1" resource="./keystore/keystore.jceks"
      password="some_password" type="JCEKS" />
   <bean id="signer1"
      class="org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="privateKeyAlias" value="rsa" />
      <property name="signatureAlgorithm" value="SHA256withRSA" />
      <property name="includeCertificates" value="true" />
      <!-- optional parameter 'password', if not set then the password of the
KeyStoreParameters will be used for accessing the private key -->
      <property name="password" value="private_key_pw" />
   </bean>
...
   <route>
      <from uri="direct:start" />
      <to uri="crypto-cms:sign://testsign?
signer=#signer1&amp;includeContent=true&amp;toBase64=true" />
      <to uri="crypto-cms:verify://testverify?
keyStoreParameters=#keyStoreParameters1&amp;fromBase64=true" />
      <to uri="mock:result" />
   </route>
```

Java DSL の 2 つの署名者の例

```
import org.apache.camel.util.jsse.KeyStoreParameters;
import org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo;
...
KeyStoreParameters keystore  = new KeyStoreParameters();
keystore.setType("JCEKS");
keystore.setResource("keystore/keystore.jceks);
keystore.setPassword("some_password"); // this password will also be used for accessing
the private key if not specified in the signerInfo1 bean

//Signer Information, by default the following signed attributes are included: contentType,
signingTime, messageDigest, and cmsAlgorithmProtect; by default no unsigned attribute is
included.
// If you want to add your own signed attributes or unsigned attributes, see methods
DefaultSignerInfo.setSignedAttributeGenerator and
DefaultSignerInfo.setUnsignedAttributeGenerator.
```

```
DefaultSignerInfo signerInfo1 = new DefaultSignerInfo();
signerInfo1.setIncludeCertificates(true); // if set to true then the certificate chain of the private key will be added to the Signed Data object
signerInfo1.setSignatureAlgorithm("SHA256withRSA"); // signature algorithm; attention, the signature algorithm must fit to the signer private key.
signerInfo1.setPrivateKeyAlias("rsa"); // alias of the private key used for the signing
signerInfo1.setPassword("private_key_pw".toCharArray()); // optional parameter, if not set then the password of the KeyStoreParameters will be used for accessing the private key
signerInfo1.setKeyStoreParameters(keystore);

DefaultSignerInfo signerInfo2 = new DefaultSignerInfo();
signerInfo2.setIncludeCertificates(true);
signerInfo2.setSignatureAlgorithm("SHA256withDSA");
signerInfo2.setPrivateKeyAlias("dsa");
signerInfo2.setKeyStoreParameters(keystore);


simpleReg.put("keyStoreParameters", keystore); //register keystore in the registry
simpleReg.put("signer1", signerInfo1); //register signer info in the registry
simpleReg.put("signer2", signerInfo2); //register signer info in the registry

from("direct:start")
   .to("crypto-cms:sign://testsign?signer=#signer1&signer=#signer2&includeContent=true")
   .to("crypto-cms:verify://testverify?keyStoreParameters=#keyStoreParameters")
   .to("mock:result");
```

Spring XML の 2 つの署名者の例

```
  <keyStoreParameters xmlns="http://camel.apache.org/schema/spring"
      id="keyStoreParameters1" resource="./keystore/keystore.jceks"
      password="some_password" type="JCEKS" />
   <bean id="signer1"
      class="org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="privateKeyAlias" value="rsa" />
      <property name="signatureAlgorithm" value="SHA256withRSA" />
      <property name="includeCertificates" value="true" />
      <!-- optional parameter 'password', if not set then the password of the
KeyStoreParameters will be used for accessing the private key -->
      <property name="password" value="private_key_pw" />
   </bean>
   <bean id="signer2"
      class="org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo">
      <property name="keyStoreParameters" ref="keyStoreParameters1" />
      <property name="privateKeyAlias" value="dsa" />
      <property name="signatureAlgorithm" value="SHA256withDSA" />
      <!-- optional parameter 'password', if not set then the password of the
KeyStoreParameters will be used for accessing the private key -->
      <property name="password" value="private_key_pw2" />
   </bean>
...
   <route>
      <from uri="direct:start" />
      <to uri="crypto-cms:sign://testsign?
```

```
signer=#signer1&amp;signer=#signer2&amp;includeContent=true" />
        <to uri="crypto-cms:verify://testverify?keyStoreParameters=#keyStoreParameters1" />
        <to uri="mock:result" />
    </route>
```

Java DSL での接続解除署名の例

```
import org.apache.camel.util.jsse.KeyStoreParameters;
import org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo;
...
KeyStoreParameters keystore  = new KeyStoreParameters();
keystore.setType("JCEKS");
keystore.setResource("keystore/keystore.jceks);
keystore.setPassword("some_password"); // this password will also be used for accessing
the private key if not specified in the signerInfo1 bean

//Signer Information, by default the following signed attributes are included: contentType,
signingTime, messageDigest, and cmsAlgorithmProtect; by default no unsigned attribute is
included.
// If you want to add your own signed attributes or unsigned attributes, see methods
DefaultSignerInfo.setSignedAttributeGenerator and
DefaultSignerInfo.setUnsignedAttributeGenerator.
DefaultSignerInfo signerInfo1 = new DefaultSignerInfo();
signerInfo1.setIncludeCertificates(true); // if set to true then the certificate chain of the private
key will be added to the Signed Data object
signerInfo1.setSignatureAlgorithm("SHA256withRSA"); // signature algorithm; attention, the
signature algorithm must fit to the signer private key.
signerInfo1.setPrivateKeyAlias("rsa"); // alias of the private key used for the signing
signerInfo1.setPassword("private_key_pw".toCharArray()); // optional parameter, if not set
then the password of the KeyStoreParameters will be used for accessing the private key
signerInfo1.setKeyStoreParameters(keystore);

simpleReg.put("keyStoreParameters", keystore); //register keystore in the registry
simpleReg.put("signer1", signerInfo1); //register signer info in the registry

from("direct:start")
    //with the option includeContent=false the SignedData object without the signed text will be
written into the header "CamelCryptoCmsSignedData"
    .to("crypto-cms:sign://testsign?signer=#signer1&includeContent=false&toBase64=true")
    //the verifier reads the Signed Data object form the header CamelCryptoCmsSignedData
and assumes that the signed content is in the message body
    .to("crypto-cms:verify://testverify?
keyStoreParameters=#keyStoreParameters&signedDataHeaderBase64=true")
    .to("mock:result");
```

Spring XML でデタッチされた署名の例

```
  <keyStoreParameters xmlns="http://camel.apache.org/schema/spring"
      id="keyStoreParameters1" resource="./keystore/keystore.jceks"
      password="some_password" type="JCEKS" />
  <bean id="signer1"
```

```
        class="org.apache.camel.component.crypto.cms.sig.DefaultSignerInfo">
        <property name="keyStoreParameters" ref="keyStoreParameters1" />
        <property name="privateKeyAlias" value="rsa" />
        <property name="signatureAlgorithm" value="SHA256withRSA" />
        <property name="includeCertificates" value="true" />
        <!-- optional parameter 'password', if not set then the password of the
KeyStoreParameters will be used for accessing the private key -->
        <property name="password" value="private_key_pw" />
    </bean>
...
    <route>
        <from uri="direct:start" />
        <!-- with the option includeContent=false the SignedData object without the signed text
will be written into the header "CamelCryptoCmsSignedData" -->
        <to uri="crypto-cms:sign://testsign?
signer=#signer1&amp;includeContent=false&amp;toBase64=true" />
        <!-- the verifier reads the Signed Data object form the header
CamelCryptoCmsSignedData and assumes that the signed content is in the message body -->
        <to uri="crypto-cms:verify://testverify?
keyStoreParameters=#keyStoreParameters1&amp;signedDataHeaderBase64=true" />
        <to uri="mock:result" />
    </route>
```

第73章 CRYPTO(JAVA CRYPTOGRAPHIC EXTENSION)DATAFORMAT

Camel バージョン 2.3 の時点で利用可能

Crypto Data Format は、Java Cryptographic Extension を Camel に統合し、Camel の一般的なマーシャルおよびアンマーシャリングフォーマットメカニズムを使用して、メッセージのシンプルで柔軟な暗号化と復号を可能にします。元の平文に復号化するために、cyphertext および unmarshalling の暗号化を意味するマーシャリングを想定しています。このデータ形式は、対称（共有キー）暗号化とデイジションのみを実装します。

73.1. CRYPTODATAFORMAT オプション

Crypto(Java Cryptographic Extension)のデータ形式は、以下に示す 10 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
algorithm	**DES/CBC/PKCS5Padding**	**文字列**	使用される暗号化アルゴリズムを示す JCE アルゴリズム名。デフォルトでは DES/CBC/PKCS5Padding です。
cryptoProvider		**文字列**	使用する JCE セキュリティープロバイダーの名前。
keyRef		**文字列**	使用する登録からルックアップする秘密鍵を参照します。
initVectorRef		**文字列**	暗号の初期化に使用される Initialization Vector を含むバイトアレイを参照します。
algorithmParameterRef		**文字列**	暗号の初期化に使用される JCE AlgorithmParameterSpec。指定の名前を java.security.spec.AlgorithmParameterSpec タイプとして使用してタイプを検索します。
buffersize		**整数**	署名プロセスで使用されるバッファーサイズ
macAlgorithm	**HmacSHA1**	**文字列**	Message Authentication アルゴリズムを示す JCE アルゴリズム名。
shouldAppendHMAC	**false**	**ブール値**	Message Authentication Code を計算し、暗号化されたデータに追加する必要があることを示すフラグ。
inline	**false**	**ブール値**	設定された IV を暗号化されたデータストリームにインライン化する必要があることを示すフラグ。デフォルトは false です。

Name	デフォルト	Java タイプ	説明
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

73.2. 基本的な使用方法

交換の暗号化/復号化に必要となる最も基本的なものはすべて、共有秘密キーです。Crypto データフォーマットの 1 つ以上のインスタンスがこのキーで設定されている場合、フォーマットを使用してあるルート（または一部）でペイロードを暗号化し、別のルートで復号化できます。たとえば、以下のように Java DSL を使用します。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES", generator.generateKey());

from("direct:basic-encryption")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(cryptoFormat)
    .to("mock:unencrypted");
```

Spring では、最初に dataformat を設定し、ルートで使用されます。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <dataFormats>
    <crypto id="basic" algorithm="DES" keyRef="desKey" />
  </dataFormats>
    ...
  <route>
    <from uri="direct:basic-encryption" />
    <marshal ref="basic" />
    <to uri="mock:encrypted" />
    <unmarshal ref="basic" />
    <to uri="mock:unencrypted" />
  </route>
</camelContext>
```

73.3. 暗号化アルゴリズムの指定

アルゴリズムの変更は、JCE アルゴリズム名を提供することです。アルゴリズムを変更する場合は、互換性のあるキーを使用する必要があります。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES", generator.generateKey());
cryptoFormat.setShouldAppendHMAC(true);
cryptoFormat.setMacAlgorithm("HmacMD5");

from("direct:hmac-algorithm")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(cryptoFormat)
    .to("mock:unencrypted");
```

Java 7 で利用可能なアルゴリズムの一覧は、『Java Cryptography Architecture Standard Algorithm Name Documentation』で確認できます。

73.4. 初期化ベイゼーションベクトルの指定

暗号アルゴリズム（特にブロックアルゴリズム）によっては、初期ブロックが Initialization Vector として知られるデータの初期ブロックの設定を必要とするものもあります。JCE では、これは Cipher が初期化されたときに AlgorithmParameterSpec として渡されます。このようなベクトルを CryptoDataFormat と併用するには、必要なデータが含まれる byte[] で設定できます。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");
byte[] initializationVector = new byte[] {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07};

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES/CBC/PKCS5Padding",
generator.generateKey());
cryptoFormat.setInitializationVector(initializationVector);

from("direct:init-vector")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(cryptoFormat)
    .to("mock:unencrypted");
```

または、spring の場合は、byte[] への参照を一時停止します。

```
<crypto id="initvector" algorithm="DES/CBC/PKCS5Padding" keyRef="desKey"
initVectorRef="initializationVector" />
```

暗号化フェーズと復号化の両方のフェーズでは、同じベクトルが必要です。IV はシークレットを保持する必要がないため、DataFormat は暗号化されたデータにインライン化し、その後の復号化フェーズで読み取り、暗号の初期化を可能にします。IV をインラインにするには、/oinline フラグを設定します。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");
byte[] initializationVector = new byte[] {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07};
SecretKey key = generator.generateKey();

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES/CBC/PKCS5Padding", key);
cryptoFormat.setInitializationVector(initializationVector);
cryptoFormat.setShouldInlineInitializationVector(true);
CryptoDataFormat decryptFormat = new CryptoDataFormat("DES/CBC/PKCS5Padding", key);
decryptFormat.setShouldInlineInitializationVector(true);

from("direct:inline")
   .marshal(cryptoFormat)
   .to("mock:encrypted")
   .unmarshal(decryptFormat)
   .to("mock:unencrypted");
```

または spring で行います。

```
<crypto id="inline" algorithm="DES/CBC/PKCS5Padding" keyRef="desKey"
initVectorRef="initializationVector"
  inline="true" />
<crypto id="inline-decrypt" algorithm="DES/CBC/PKCS5Padding" keyRef="desKey" inline="true" />
```

初期化ベクトルの使用の詳細については、 を参照してください。

- http://en.wikipedia.org/wiki/Initialization_vector
- http://www.herongyang.com/Cryptography/
- http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

73.5. ハッシュされたメッセージ認証コード(HMAC)

CryptoDataFormat の移動中に暗号化されたデータに対する攻撃を回避するために、設定可能な MAC アルゴリズムに基づいて、暗号化されたエクスチェンジコンテンツ用の Message Authentication Code を算出することもできます。計算された HMAC が、暗号化後にストリームに追加されます。これは復号化フェーズでストリームから分離されます。MAC は再計算され、送信済みのバージョンに対して検証され、送信済みバージョンに対して転送中に改ざんされませんでした。メッセージ認証コードの詳細は、 http://en.wikipedia.org/wiki/HMACを参照してください。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");
```

```
CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES", generator.generateKey());
cryptoFormat.setShouldAppendHMAC(true);

from("direct:hmac")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(cryptoFormat)
    .to("mock:unencrypted");
```

または spring で行います。

```
<crypto id="hmac" algorithm="DES" keyRef="desKey" shouldAppendHMAC="true" />
```

デフォルトでは、HmacSHA1 mac アルゴリズムを使用して HMAC が計算されますが、別のアルゴリズム名を指定することで簡単に変更できます。設定したセキュリティープロバイダーから利用可能なアルゴリズムを確認する方法については、こちらを参照してください。

```
KeyGenerator generator = KeyGenerator.getInstance("DES");

CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES", generator.generateKey());
cryptoFormat.setShouldAppendHMAC(true);
cryptoFormat.setMacAlgorithm("HmacMD5");

from("direct:hmac-algorithm")
    .marshal(cryptoFormat)
    .to("mock:encrypted")
    .unmarshal(cryptoFormat)
    .to("mock:unencrypted");
```

または spring で行います。

```
<crypto id="hmac-algorithm" algorithm="DES" keyRef="desKey" macAlgorithm="HmacMD5"
shouldAppendHMAC="true" />
```

73.6. キーの動的指定

Recipient list または同様の EIP を使用する場合、エクスチェンジの受信者は動的に異なる可能性があります。すべての受信者で同じキーを使用すると、現実的でも望ましくない場合もあります。エクスチェンジごとにキーを動的に指定できるのは便利です。その後、交換は、データ形式で処理される前に、ターゲット受信者のキーで動的に強化できます。この DataFormat を容易にするために、以下のメッセージヘッダーを使用してキーを動的に指定できます。

- CryptoDataFormat.KEY "CamelCryptoKey"

```
CryptoDataFormat cryptoFormat = new CryptoDataFormat("DES", null);
/**
 * Note: the header containing the key should be cleared after
 * marshalling to stop it from leaking by accident and
 * potentially being compromised. The processor version below is
 * arguably better as the key is left in the header when you use
 * the DSL leaks the fact that camel encryption was used.
 */
from("direct:key-in-header-encrypt")
    .marshal(cryptoFormat)
    .removeHeader(CryptoDataFormat.KEY)
    .to("mock:encrypted");

from("direct:key-in-header-decrypt").unmarshal(cryptoFormat).process(new Processor() {
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().getHeaders().remove(CryptoDataFormat.KEY);
        exchange.getOut().copyFrom(exchange.getIn());
    }
}).to("mock:unencrypted");
```

または spring で行います。

```
<crypto id="nokey" algorithm="DES" />
```

73.7. 依存関係

camel ルートで Crypto dataformat を使用するには、以下の依存関係を pom に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-crypto</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

73.8. 関連項目

- データフォーマット
- crypto(Digital Signatures)
- http://www.bouncycastle.org/java.html

第74章 CSV DATAFORMAT

Camel バージョン 1.3 で利用可能

CSV データフォーマットは Apache Commons CSV を使用して、Hof Excel によってエクスポート/インポートされる値などの CSV ペイロード(Comma Separated Values)を処理します。

74.1. オプション

CSV データフォーマットは、以下に示す 28 オプションをサポートします。

Name	デフォルト	Java タイプ	説明
formatRef		文字列	使用する参照形式です。これは他の形式のオプションで更新されます。デフォルト値は CSVFormat.DEFAULT です。
formatName		文字列	使用する形式の名前。デフォルト値は CSVFormat.DEFAULT です。
commentMarkerDisabled	false	ブール値	参照形式のコメントマーカーを無効にします。
commentMarker		文字列	参照形式のコメントマーカーを設定します。
delimiter		文字列	使用する区切り文字を設定します。デフォルト値は ,（コンマ）です。
escapeDisabled	false	ブール値	エスケープ文字の使用の無効化に使用
escape		文字列	使用するエスケープ文字を設定します。
headerDisabled	false	ブール値	ヘッダーの無効化に使用
ヘッダー		リスト	CSV ヘッダーを設定するには、以下を実行します。
allowMissingColumnNames	false	ブール値	列名がないかどうか。
ignoreEmptyLines	false	ブール値	空の行を無視するかどうか。

Name	デフォルト	Java タイプ	説明
ignoreSurroundingSpaces	**false**	**ブール値**	周りのスペースを無視するかどうか。
nullStringDisabled	**false**	**ブール値**	null 文字列の無効化に使用
nullString		**文字列**	null 文字列を設定します。
quoteDisabled	**false**	**ブール値**	引用符の無効化に使用
quote		**文字列**	デフォルトでは引用符を設定します。
recordSeparatorDisabled		**文字列**	レコードセパレーターの無効化に使用
recordSeparator		**文字列**	レコードセパレーター（別名改行）を設定します。デフォルトは新しい行文字(CRLF)です。
skipHeaderRecord	**false**	**ブール値**	出力のヘッダーレコードをスキップするかどうか。
quoteMode		**文字列**	引用符モードを設定します。
ignoreHeaderCase	**false**	**ブール値**	ヘッダー名へのアクセス時にケースを無視するかどうかを設定します。
trim	**false**	**ブール値**	先頭および末尾の空白をトリミングするかどうかを設定します。
trailingDelimiter	**false**	**ブール値**	末尾の区切り文字を追加するかどうかを設定します。
lazyLoad	**false**	**ブール値**	アンマーシャリングが、すぐに行を読み取るイテレーターを生成するか、すべての行が一度に読み取る必要があるかどうか。
useMaps	**false**	**ブール値**	リストではなく行値にアンマーシャリングがマップ(HashMap)を生成するかどうか。ヘッダーが必要です（定義または収集）。
useOrderedMaps	**false**	**ブール値**	リストの代わりに、アンマーシャリングが行値に順序付けされたマップ(LinkedHashMap)を作成するかどうか。ヘッダーが必要です（定義または収集）。

Name	デフォルト	Java タイプ	説明
recordConverterRef		文字列	使用するレジストリーからルックアップするカスタムの CsvRecordConverter を参照します。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

74.2. マップの CSV へのマーシャリング

コンポーネントを使用すると、Java マップ（またはマップで変換できる他のメッセージタイプ）を CSV ペイロードにマーシャリングすることができます。

以下のボディーを考慮する

```
Map<String, Object> body = new LinkedHashMap<>();
body.put("foo", "abc");
body.put("bar", 123);
```

そして、この Java ルート定義です。

```
from("direct:start")
    .marshal().csv()
    .to("mock:result");
```

または、この XML ルート定義

```
<route>
    <from uri="direct:start" />
    <marshal>
        <csv />
    </marshal>
    <to uri="mock:result" />
</route>
```

生成される

```
abc,123
```

74.3. CSV メッセージの JAVA リストへのアンマーシャリング

アンマーシャリングは、CSV ファイル行が含まれる CSV コンテナーが Java List に変換されます（フィールド値すべてが含まれる別の List も含まれます）。

例： CSV ファイルに persons の名前、その IQ、および現在のアクティビティーがあります。

```
Jack Dalton, 115, mad at Averell
Joe Dalton, 105, calming Joe
William Dalton, 105, keeping Joe from killing Averell
Averell Dalton, 80, playing with Rantanplan
Lucky Luke, 120, capturing the Daltons
```

CSV コンポーネントを使用して、このファイルをアンマーシャリングできるようになりました。

```
from("file:src/test/resources/?fileName=daltons.csv&noop=true")
    .unmarshal().csv()
    .to("mock:daltons");
```

作成されるメッセージには、List< List<String>> like...

```
List<List<String>> data = (List<List<String>>) exchange.getIn().getBody();
for (List<String> line : data) {
    LOG.debug(String.format("%s has an IQ of %s and is currently %s", line.get(0), line.get(1),
line.get(2)));
}
```

74.4. LIST<MAP> を CSV にマーシャリングする

Camel 2.1 から利用可能

CSV 形式にマーシャリングするデータ行が複数ある場合に、メッセージペイロードを List<Map<String, Object> オブジェクトとして保存し、 リストに各行の Map が含まれるようになりました。

74.5. CSV のファイルポーリングを行ってからアンマーシャリング

受信データを処理できる Bean の指定...

MyCsvHandler.java

```
// Some comments here
public void doHandleCsvData(List<List<String>> csvData)
{
    // do magic here
}
```

i. ルートは以下のようになります。

```
<route>
      <!-- poll every 10 seconds -->
      <from uri="file:///some/path/to/pickup/csvfiles?delete=true&amp;consumer.delay=10000" />
      <unmarshal><csv /></unmarshal>
      <to uri="bean:myCsvHandler?method=doHandleCsvData" />
</route>
```

74.6. パイプを使用して区切り文字としてマーシャルする

以下のボディーを考慮する

```
Map<String, Object> body = new LinkedHashMap<>();
body.put("foo", "abc");
body.put("bar", 123);
```

そして、この Java ルート定義です。

```
// Camel version < 2.15
CsvDataFormat oldCSV = new CsvDataFormat();
oldCSV.setDelimiter("|");
from("direct:start")
    .marshal(oldCSV)
    .to("mock:result")

// Camel version >= 2.15
from("direct:start")
    .marshal(new CsvDataFormat().setDelimiter(&#39;|&#39;))
    .to("mock:result")
```

または、この XML ルート定義

```
<route>
  <from uri="direct:start" />
```

```
  <marshal>
    <csv delimiter="|" />
  </marshal>
  <to uri="mock:result" />
</route>
```

生成される

```
abc|123
```

XML # DSL 内で autogenColumns、configRef、および strategyRef 属性を使用

Camel 2.9.2 / 2.10 で利用可能で、Camel 2.15 で削除されました。

CSV データ形式をカスタマイズして、独自の CSVConfig および/または CSVStrategy を使用できます。また、autogenColumns オプションのデフォルト値は true であることに注意してください。以下の例は、このカスタマイズを示しています。

```
<route>
  <from uri="direct:start" />
  <marshal>
    <!-- make use of a strategy other than the default one which is
'org.apache.commons.csv.CSVStrategy.DEFAULT_STRATEGY' -->
    <csv autogenColumns="false" delimiter="|" configRef="csvConfig" strategyRef="excelStrategy" />
  </marshal>
  <convertBodyTo type="java.lang.String" />
  <to uri="mock:result" />
</route>

<bean id="csvConfig" class="org.apache.commons.csv.writer.CSVConfig">
  <property name="fields">
    <list>
      <bean class="org.apache.commons.csv.writer.CSVField">
        <property name="name" value="orderId" />
      </bean>
      <bean class="org.apache.commons.csv.writer.CSVField">
        <property name="name" value="amount" />
      </bean>
    </list>
  </property>
</bean>

<bean id="excelStrategy"
class="org.springframework.beans.factory.config.FieldRetrievingFactoryBean">
  <property name="staticField" value="org.apache.commons.csv.CSVStrategy.EXCEL_STRATEGY"
/>
</bean>
```

74.7. アンマーシャリング中に SKIPFIRSTLINE オプションを使用

Camel 2.10 で利用可能で、Camel 2.15 で削除されました。

CSV データフォーマットに、CSV ヘッダーを含む最初の行を省略するように指示できます。
Spring/XML DSL の使用：

```
<route>
  <from uri="direct:start" />
  <unmarshal>
    <csv skipFirstLine="true" />
  </unmarshal>
  <to uri="bean:myCsvHandler?method=doHandleCsv" />
</route>
```

または Java DSL の場合：

```
CsvDataFormat csv = new CsvDataFormat();
csv.setSkipFirstLine(true);

from("direct:start")
  .unmarshal(csv)
.to("bean:myCsvHandler?method=doHandleCsv");
```

74.8. パイプを使用して区切り文字としてアンマーシャリング

Spring/XML DSL の使用：

```
<route>
  <from uri="direct:start" />
  <unmarshal>
    <csv delimiter="|" />
  </unmarshal>
  <to uri="bean:myCsvHandler?method=doHandleCsv" />
</route>
```

または Java DSL の場合：

```
CsvDataFormat csv = new CsvDataFormat();
CSVStrategy strategy = CSVStrategy.DEFAULT_STRATEGY;
strategy.setDelimiter('|');
csv.setStrategy(strategy);
```

```
from("direct:start")
  .unmarshal(csv)
  .to("bean:myCsvHandler?method=doHandleCsv");
```

```
CsvDataFormat csv = new CsvDataFormat();
csv.setDelimiter("|");

from("direct:start")
  .unmarshal(csv)
  .to("bean:myCsvHandler?method=doHandleCsv");
```

```
CsvDataFormat csv = new CsvDataFormat();
CSVConfig csvConfig = new CSVConfig();
csvConfig.setDelimiter(";");
csv.setConfig(csvConfig);

from("direct:start")
  .unmarshal(csv)
  .to("bean:myCsvHandler?method=doHandleCsv");
```

CSVConfig の問題

以下のようになります。

```
CSVConfig csvConfig = new CSVConfig();
csvConfig.setDelimiter(';');
```

動作しません。区切り文字を String として設定する必要があります。

74.9. 依存関係

Camel ルートで CSV を使用するには、このデータ形式を実装する camel-csv の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加できます。バージョン番号は最新の最新のリリースに置き換えてください（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-csv</artifactId>
  <version>x.x.x</version>
</dependency>
```

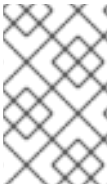
第75章 CXF

CXF コンポーネント

cxf: コンポーネントは、CXF でホストされる JAX-WS サービスに接続するための Apache CXF との統合を提供します。

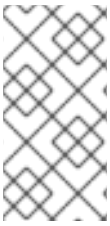
Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cxf</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```



注記

CXF の依存関係については、WHICH-JARS テキストファイルを参照してください。



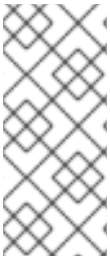
注記

ストリーミングモードで CXF を使用する場合（DataFormat オプションを参照）、Stream caching も読み取ります。

CAMEL ON EAP デプロイメント

このコンポーネントは、Red Hat JBoss Enterprise Application Platform (JBoss EAP) コンテナー上で簡素化されたデプロイメントモデルを提供する Camel on EAP (Wildfly Camel) フレームワークによってサポートされます。

CXF コンポーネントは、Apache CXF も使用する JBoss EAP webservices の susbsystem と統合します。詳細は「JAX-WS」を参照してください。



注記

現在、Camel on EAP サブシステムでは CXF または Restlet コンシューマー はサポートさ れません。ただし、CamelProxy を使用すると、CXF コンシューマーの動作を模倣することができます。

URI 形式

```
cxf:bean:cxfEndpoint[?options]
```

cxfEndpoint は Spring Bean レジストリーの Bean を参照する Bean ID を表します。この URI 形式では、ほとんどのエンドポイント詳細が Bean 定義で指定されます。

```
cxf://someAddress[?options]
```

someAddress は、CXF エンドポイントのアドレスを指定します。この URI 形式では、ほとんどのエンドポイント詳細がオプションを使用して指定されます。

上記のいずれかのスタイルで、以下のように URI にオプションを追加できます。

```
cxf:bean:cxfEndpoint?wsdlURL=wsdl/hello_world.wsdl&dataFormat=PAYLOAD
```

オプション

Name	必須	説明
wsdlURL	非対応	WSDL の場所。WSDL はデフォルトでエンドポイントアドレスから取得されます。以下に例を示します。 **file://local/wsdl/hello.wsdl** or **wsdl/hello.wsdl**

serviceClass	対応	SEI(Service Endpoint Interface)クラスの名前。このクラスは JSR181 アノテーションを持つことができますが、必須ではありません。 **2.0 以降**、このオプションは POJO モードでのみ必要です。wsdlURL オプションを指定する場合、serviceClass は PAYLOAD モードおよび MESSAGE モードでは必要ありません。serviceClass なしで wsdlURL オプションを使用すると、serviceName オプションおよび portName オプション（Spring 設定の endpointName）オプションを指定する **必要** があります。 2.0 以降、**\#** 表記を使用してレジストリーから **serviceClass** オブジェクトインスタンスを参照できます。 **参照されるオブジェクトは、Spring AOP 以外のプロキシーの Object.getClass（）.getName（）メソッドに依存するため、プロキシー（Spring AOP プロキシーは OK）** にすることはできません。 **2.8 以降、** PAYLOAD モードおよび MESSAGE モード の wsdlURL および serviceClass オプションの両方を省略することができます。省略すると、CXF の Dispatch Mode を容易にするために、任意の XML 要素を PAYLOAD モードの CxfPayload ボディーに配置できます。 例： **org.apache.camel.Hello**
serviceName	WSDL に複数の **serviceName** が存在する場合にのみ	このサービスは実装されているサービス名で、**wsdl:service@name** にマップされます。以下に例を示します。 **{http://org.apache.camel}ServiceName**

endpointName	**serviceName** の下に複数の **portName** が存在し、camel 2.2 以降 camel-cxf コンシューマーに必要です。	このサービスが実装しているポート名。**wsdl:port@name** にマッピングします。以下に例を示します。 **{http://org.apache.camel}PortName**
dataFormat	非対応	CXF エンドポイントがサポートするメッセージデータフォーマット。使用できる値は **POJO（デフォルト）**、**PAYLOAD**、**MESSAGE** です。
relayHeaders	非対応	このオプション **のrelayHeadersオプション の説明**セクションを参照してください。CXF エンドポイントがルートを通過する必要があります。現在、**dataFormat= POJO**Default:**true**Example:**true**,**false**の場合にのみ利用できます。
wrapped	非対応	CXF エンドポイントプロデューサーが呼び出す操作の種類。使用できる値は **true**、**false（デフォルト）** です。
wrappedStyle	非対応	**2.5.0 以降**、SOAP ボディーでパラメーターを表す方法を記述する WSDL スタイル。値が **false** の場合、CXF は document-literal unwraped スタイルを選択します。値が **true** の場合、CXF は document-literal ラップされたスタイルを選択します。
setDefaultBus	非対応	**deprecated:** このエンドポイントにデフォルトの CXF バスを使用するかどうかを指定します。使用できる値は **true**、**false（デフォルト）** です。このオプションは非推奨になっており、Camel 2.16 以降では **defaultBus** を使用する必要があります。

defaultBus	非対応	deprecated: このエンドポイントにデフォルトの CXF バスを使用するかどうかを指定します。使用できる値は **true**、**false（デフォルト）** です。このオプションは非推奨になっており、Camel 2.16 以降では **defaultBus** を使用する必要があります。
bus	非対応	**\#** 表記を使用して、レジストリー（例： **bus=\#busName** ）からバスオブジェクトを参照します。参照オブジェクトは **org.apache.cxf.Bus** のインスタンスである必要があります。 デフォルトでは、CXF Bus Factory によって作成されたデフォルトのバスを使用します。
cxfBinding	非対応	**\#** 表記を使用して、レジストリーで CXF バインディングオブジェクトを参照します（例： **cxfBinding=\#bindingName** ）。参照されるオブジェクトは **org.apache.camel.component.cxf.CxfBinding** のインスタンスである必要があります。
headerFilterStrategy	非対応	**\#** 表記を使用して、レジストリーでヘッダーフィルターストラテジーオブジェクトを参照します（例： **headerFilterStrategy=\#strategyName** ）。参照オブジェクトは **org.apache.camel.spi.HeaderFilterStrategy** のインスタンスである必要があります。
loggingFeatureEnabled	非対応	2.3 の新機能。このオプションは、インバウンドおよびアウトバウンド SOAP メッセージをログに書き込む CXF ロギング機能を有効にします。使用できる値は **true**、**false（デフォルト）** です。

defaultOperationName	非対応	2.4 の新機能として、このオプションはリモートサービスを呼び出す **CxfProducer** によって使用されるデフォルトの **operationName** を設定します。以下に例を示します。 **defaultOperationName=greetMe**
defaultOperationNamespace	非対応	2.4 の新機能として、このオプションはリモートサービスを呼び出す CxfProducer によって使用されるデフォルトの operationNamespace を設定します。以下に例を示します。 **defaultOperationNamespace=http://apache.org/hello_world_soap_http**
同期	非対応	2.5 の新機能、このオプションを使用すると、CXF エンドポイントが sync または async API を使用して基礎となる作業を行うことを決定できるようになりました。デフォルト値は **false** です。つまり、**camel-cxf** エンドポイントはデフォルトで async API の使用を試行します。
publishedEndpointUrl	非対応	2.5 の新機能として、このオプションはサービスアドレス URL と **?wsdl** を使用してアクセスされる公開された WSDL に表示されるエンドポイント URL を上書きします。以下に例を示します。 **publishedEndpointUrl=http://example.com/service**
properties.propName	非対応	Camel 2.8: エンドポイント URI にカスタム CXF プロパティーを設定できます。たとえば、**properties.mtom-enabled=true** を設定して MTOM を有効にします。呼び出しの開始時に CXF がスレッドを切り替えないようにするには、**properties.org.apache.cxf.interceptor.OneWayProcessorInterceptor.USE_ORIGINAL_THREAD=true** を設定します。

allowStreaming	非対応	2.8.2 の新機能このオプションは、PAYLOAD モードで実行されている場合に CXF コンポーネントを制御します（以下を参照）、受信メッセージを DOM 要素に解析するか、ペイロードを一部のケースでストリーミングできる javax.xml.transform.Source オブジェクトとして解析するかを制御します。
skipFaultLogging	非対応	2.11 の新機能。このオプションは、PhaseInterceptorChain がキャッチするフォールトのロギングをスキップするかどうかを制御します。
cxfEndpointConfigurer	非対応	Camel 2.11 の新機能。このオプションは、プログラムによる CXF エンドポイントの設定をサポートする **org.apache.camel.component.cxf.CxfEndpointConfigurer** の実装を適用する可能性があります。**Camel 2.15.0 以降、** ユーザーは configure{Server を実装して CXF サーバーおよびクライアントを設定できます。
CxfEndpointConfigurer の client} メソッド。	**username**	非対応
Camel 2.12.3 の新機能。このオプションは、CXF クライアントのユーザー名の基本認証情報を設定するために使用されます。	**password**	非対応
Camel 2.12.3 の新機能。このオプションは、CXF クライアントのパスワードの基本認証情報を設定するために使用されます。	**continuationTimeout**	非対応

serviceName および portName は QNames であるため、上記の例に示すように、それらを指定する際に {namespace} のプレフィックスが付けられるようにしてください。

データフォーマットの説明

DataFormat	説明

POJO	POJO（古い Java オブジェクト）は、ターゲットサーバーで呼び出されるメソッドに対する Java パラメーターです。プロトコルおよび論理 JAX-WS ハンドラーの両方がサポートされます。
PAYLOAD	**PAYLOAD** は、CXF エンドポイントでメッセージ設定が適用された後にメッセージペイロード（**soap:body**の内容）です。Protocol JAX-WS ハンドラーのみがサポートされます。論理 JAX-WS ハンドラーはサポートされていません。
MESSAGE	**MESSAGE** は、トランスポート層から受信される raw メッセージです。Stream と通信または変更することは推奨されません。このような DataFormat を使用している場合は、camel-cxf コンシューマーと JAX-WS ハンドラーがサポートされないため、CXF インターセプターの一部が削除されます。
CXF_MESSAGE	Camel 2.8.2 の新機能。**CXF_MESSAGE** により、トランスポート層から未加工 SOAP メッセージにメッセージを変換して CXF インターセプターの完全な機能を呼び出すことが可能になりました。

エクスチェンジプロパティー CamelCXFDataFormat を取得すると、エクスチェンジのデータ形式モードを判別できます。交換キー定数は org.apache.camel.component.cxf.CxfConstants.DATA_FORMAT_PROPERTY で定義されます。

APACHE ARIES BLUEPRINT を使用した CXF エンドポイントの設定

Camel 2.8 以降、CXF エンドポイントの Aries Blueprint 依存関係インジェクションの使用はサポートされません。スキーマは Spring スキーマと非常に似ているため、移行はかなり透過的です。

以下に例を示します。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
           xmlns:camel-cxf="http://camel.apache.org/schema/blueprint/cxf"
      xmlns:cxfcore="http://cxf.apache.org/blueprint/core"
           xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

        <camel-cxf:cxfEndpoint id="routerEndpoint"
                   address="http://localhost:9001/router"
                   serviceClass="org.apache.servicemix.examples.cxf.HelloWorld">
          <camel-cxf:properties>
```

```
            <entry key="dataFormat" value="MESSAGE"/>
        </camel-cxf:properties>
    </camel-cxf:cxfEndpoint>

    <camel-cxf:cxfEndpoint id="serviceEndpoint"
  address="http://localhost:9000/SoapContext/SoapPort"
                serviceClass="org.apache.servicemix.examples.cxf.HelloWorld">
    </camel-cxf:cxfEndpoint>

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <route>
            <from uri="routerEndpoint"/>
            <to uri="log:request"/>
        </route>
    </camelContext>

</blueprint>
```

現在、endpoint 要素は最初にサポートされている CXF namespacehandler です。

Spring と同様に Bean 参照を使用することもできます。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
           xmlns:jaxws="http://cxf.apache.org/blueprint/jaxws"
           xmlns:cxf="http://cxf.apache.org/blueprint/core"
           xmlns:camel="http://camel.apache.org/schema/blueprint"
           xmlns:camelcxf="http://camel.apache.org/schema/blueprint/cxf"
           xsi:schemaLocation="
             http://www.osgi.org/xmlns/blueprint/v1.0.0
https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
             http://cxf.apache.org/blueprint/jaxws http://cxf.apache.org/schemas/blueprint/jaxws.xsd
             http://cxf.apache.org/blueprint/core http://cxf.apache.org/schemas/blueprint/core.xsd
             ">

    <camelcxf:cxfEndpoint id="reportIncident"
                address="/camel-example-cxf-blueprint/webservices/incident"
                wsdlURL="META-INF/wsdl/report_incident.wsdl"
                serviceClass="org.apache.camel.example.reportincident.ReportIncidentEndpoint">
    </camelcxf:cxfEndpoint>

    <bean id="reportIncidentRoutes"
class="org.apache.camel.example.reportincident.ReportIncidentRoutes" />

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <routeBuilder ref="reportIncidentRoutes"/>
    </camelContext>

</blueprint>
```

MESSAGE モードで CXF の LOGGINGOUTINTERCEPTOR を有効にする方法

CXF の LoggingOutInterceptor は、回線上にあるアウトバウンドメッセージをロギングシステム (java.util.logging)に出力します。LoggingOutInterceptor は PRE_STREAM フェーズであるため（ PRE_STREAM フェーズは MESSAGE モードで削除されています）、LoggingOutInterceptor が WRITE フェーズ中に実行されるように設定する必要があります。以下に例を示します。

```
  <bean id="loggingOutInterceptor" class="org.apache.cxf.interceptor.LoggingOutInterceptor">
      <!--  it really should have been user-prestream but CXF does have such phase! -->
      <constructor-arg value="target/write"/>
  </bean>

<cxf:cxfEndpoint id="serviceEndpoint" address="http://localhost:9002/helloworld"
 serviceClass="org.apache.camel.component.cxf.HelloService">
 <cxf:outInterceptors>
    <ref bean="loggingOutInterceptor"/>
 </cxf:outInterceptors>
 <cxf:properties>
  <entry key="dataFormat" value="MESSAGE"/>
 </cxf:properties>
</cxf:cxfEndpoint>
```

RELAYHEADERS オプションの説明

JAXWS WSDL -first 開発者 の観点からは、帯域外 および帯域外ヘッダーがあります。

*アウト バウンド ヘッダーは、**SOAP** ヘッダーなどのエンドポイントの **WSDL** バインディングコントラクトの一部として明示的に定義されるヘッダーです。*

*アウトバウンド ヘッダーは、ネットワークを介してシリアライズされるヘッダーですが、**WSDL** バインディングコントラクトの一部ではありません。*

ヘッダーのリレー/フィルタリングは双方向です。

*ルートに **CXF** エンドポイントがあり、開発者が **SOAP** ヘッダーなどの有線ヘッダーを持つ必要がある場合、別の **JAXWS** エンドポイントによって消費されるルートと一緒にリレーされる必要がある場合は、**relayHeaders** がデフォルト値の **true** に設定する必要があります。*

POJO モードでのみ利用可能

***relayHeaders=true** 設定は、ヘッダーをリレーする意図を表します。特定のヘッダーがリレーされる*

かどうかの実際の決定は、MessageHeadersRelay インターフェースを実装するプラグ可能なインスタンスに委譲されます。MessageHeadersRelay の具体的な実装を参照し、ヘッダーをリレーする必要があるかどうかを判断します。すでに SoapMessageHeadersRelay の実装があり、これはそれ自体をよく知られた SOAP ネームスペースにバインドします。現在、out ofband ヘッダーのみがフィルターされ、relayHeaders=true の場合、Oband ヘッダーは常にリレーされます。ネットワーク上にヘッダーがあり、その名前空間がランタイムに不明な場合は、フォールバックする DefaultMessageHeadersRelay が使用され、すべてのヘッダーをリレーできます。

relayHeaders=false 設定は、すべてのヘッダー、帯域外および帯域外(out ofband)がドロップされることをアサートします。

独自の MessageHeadersRelay 実装をオーバーライドしたり、リレーのリストに追加したりできます。事前に読み込んだリレーインスタンスをオーバーライドするには、MessageHeadersRelay 実装サービスが上書きするものと同じ名前空間になるようにしてください。また、オーバーライドされるリレーは上書きする名前と同じスペースをすべて処理する必要があります。そうでない場合には、ルートの起動時にランタイム例外がスローされます。これは、インスタンスマッピングをリレーする名前空間に曖昧さが発生するためです。

```
<cxf:cxfEndpoint ...>
  <cxf:properties>
    <entry key="org.apache.camel.cxf.message.headers.relays">
      <list>
        <ref bean="customHeadersRelay"/>
      </list>
    </entry>
  </cxf:properties>
 </cxf:cxfEndpoint>
 <bean id="customHeadersRelay"
class="org.apache.camel.component.cxf.soap.headers.CustomHeadersRelay"/>
```

ヘッダーをリレー/ドロップする方法を示すテストをご覧ください。

link:https://svn.apache.org/repos/asf/camel/branches/camel-1.x/components/camel-cxf/src/test/java/org/apache/camel/component/cxf/soap/headers/CxfMessageHeadersRelayTest.java[https://svn.apache.org/repos/asf/camel/branches/camel-1.x/components/camel-cxf/src/test/java/org/apache/camel/component/cxf/soap/headers/CxfMessageHeadersRelayTest.java]

リリース 2.0 以降の変更

- *POJO および PAYLOAD モードがサポートされます。POJO モードでは、帯域外ヘッダーのみが処理され、CXF のヘッダーリストから削除されたため、フィルタリングにだけ利用できます。アウトバウンドヘッダーは、POJO モードの MessageContentList に組み込まれています。camel-cxf コンポーネントは、MessageContentList から In-band ヘッダーのフィルター*

が必要な場合に、PAYLOAD モードまたはプラグインを CXF インターセプター/JAXWS Handler で使用してみてください。

- **Message Header Relay メカニズムは CxfHeaderFilterStrategy に統合されました。relayHeaders オプション、そのセマンティクス、およびデフォルト値は同じですが、CxfHeaderFilterStrategy のプロパティーになります。以下は、設定例です。**

```
<bean id="dropAllMessageHeadersStrategy"
class="org.apache.camel.component.cxf.common.header.CxfHeaderFilterStrategy">

    <!--  Set relayHeaders to false to drop all SOAP headers -->
    <property name="relayHeaders" value="false"/>

</bean>
```

次に、エンドポイントは CxfHeaderFilterStrategy を参照できます。

```
<route>
    <from uri="cxf:bean:routerNoRelayEndpoint?
headerFilterStrategy=#dropAllMessageHeadersStrategy"/>
    <to uri="cxf:bean:serviceNoRelayEndpoint?
headerFilterStrategy=#dropAllMessageHeadersStrategy"/>
</route>
```

- **MessageHeadersRelay インターフェースが若干変更され、MessageHeaderFilter に変更されました。これは CxfHeaderFilterStrategy のプロパティーです。以下は、ユーザー定義の Message Header フィルターを設定する例です。**

```
<bean id="customMessageFilterStrategy"
class="org.apache.camel.component.cxf.common.header.CxfHeaderFilterStrategy">
    <property name="messageHeaderFilters">
        <list>
            <!--  SoapMessageHeaderFilter is the built in filter.  It can be removed by omitting it. -
->
            <bean
class="org.apache.camel.component.cxf.common.header.SoapMessageHeaderFilter"/>

            <!--  Add custom filter here -->
            <bean class="org.apache.camel.component.cxf.soap.headers.CustomHeaderFilter"/>
        </list>
    </property>
</bean>
```

- **relayHeaders 以外の新しいプロパティーは、CxfHeaderFilterStrategy で設定できます。**

Name	説明	type	必須?	デフォルト値
relayHeaders	すべてのメッセージヘッダーは、メッセージヘッダーフィルターによって処理されます。	**boolean**	非対応	**true** (1.6.1 behavior)
relayAllMessageHeaders	すべてのメッセージヘッダーが伝播されます（メッセージヘッダーフィルターによる処理なし）。	**boolean**	非対応	**false** （1.6.1 の動作）
allowFilterNamespaceClash	2 つのフィルターがアクティベーション namespace で重複している場合、プロパティーによって処理される方法が制御されます。値が **true** の場合、最後の値が優先されます。値が **false** の場合、例外が発生します。	**boolean**	非対応	**false** （1.6.1 の動作）

SPRING での CXF エンドポイントの設定

以下に示した Spring 設定ファイルを使用して CXF エンドポイントを設定できます。また、エンドポイントを camelContext タグに埋め込むこともできます。サービスエンドポイントを呼び出す場合は、operationName および operationNamespace ヘッダーを、呼び出す操作を明示的に設定できます。

Camel 2.x で、CXF エンドポイントのターゲット名前空間として http://camel.apache.org/schema/cxf を使用するように変更しました。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cxf="http://camel.apache.org/schema/cxf"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
       http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd
```

```
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd
">
  ...
```

注記

Apache Camel 2.x では、http://activemq.apache.org/camel/schema/cxfEndpoint 名前空間が http://camel.apache.org/schema/cxf に変更されました。

ルート beans 要素に指定された JAX-WS schemaLocation 属性が含まれるようにしてください。これにより、CXF はファイルを検証でき、必須です。また、< cxf:cxfEndpoint/ > タグの最後にある namespace 宣言にも注意してください。組み合わせた {namespace}localName 構文が存在すると、このタグの属性値に対してはサポートされないためです。

cxf:cxfEndpoint 要素は、多くの追加属性をサポートします。

Name	値
PortName	このサービスが実装しているエンドポイント名。**wsdl:port@name** にマッピングします。**ns:PORT_NAME** の形式では、**ns** は、このスコープで有効な名前空間プレフィックスです。
serviceName	このサービスは実装されているサービス名で、**wsdl:service@name** にマップされます。**ns:SERVICE_NAME** の形式では、**ns** は、このスコープで有効な namespace プレフィックスです。
wsdlURL	WSDL の場所。クラスパス、ファイルシステム、またはリモートでホストできます。
bindingId	使用するサービスモデルの **bindingId**。
address	サービスの公開アドレス。
bus	JAX-WS エンドポイントで使用されるバス名。
serviceClass	JSR181 アノテーションが含まれる SEI(Service Endpoint Interface)クラスのクラス名。

また、多くの子要素もサポートします。

Name	値
cxf:inInterceptors	このエンドポイントの受信インターセプター。< **bean> または &** lt; **ref&** gt; のリスト。
cxf:inFaultInterceptors	このエンドポイントの受信障害インターセプター。< **bean> または &** lt; **ref&** gt; のリスト。
cxf:outInterceptors	このエンドポイントの送信インターセプター。< **bean> または &** lt; **ref&** gt; のリスト。
cxf:outFaultInterceptors	このエンドポイントの送信障害インターセプター。< **bean> または &** lt; **ref&** gt; のリスト。
cxf:properties	JAX-WS エンドポイントに提供する必要があるプロパティーマップ。以下を参照してください。
cxf:handlers	JAX-WS エンドポイントに提供する JAX-WS ハンドラーリスト。以下を参照してください。
cxf:dataBinding	エンドポイントで使用する **DataBinding** を指定できます。これは、Spring < **bean class="MyDataBinding** "/> 構文を使用して指定できます。
cxf:binding	使用するエンドポイントの **BindingFactory** を指定できます。これは、Spring < **bean class="MyBindingFactory** "/> 構文を使用して指定できます。
cxf:features	このエンドポイントのインターセプターを保持する機能。< **bean> または &**lt; **ref> s**のリスト
cxf:schemaLocations	使用するエンドポイントのスキーマの場所。< **schemaLocation**> の一覧
cxf:serviceFactory	使用するこのエンドポイントのサービスファクトリー。これは、Spring < **bean class="MyServiceFactory"/> 構文を使用して指定できます。**

インターセプター、プロパティー、およびハンドラーを提供する方法を示すより高度な例は、http://cwiki.apache.org/CXF20DOC/jax-ws-configuration.htmlを参照してください。

注記

CXF:properties を使用して CXF エンドポイントの dataFormat、および Spring 設定ファイルから setDefaultBus プロパティーを以下のように設定できます。

```
<cxf:cxfEndpoint id="testEndpoint" address="http://localhost:9000/router"
    serviceClass="org.apache.camel.component.cxf.HelloService"
    endpointName="s:PortName"
    serviceName="s:ServiceName"
    xmlns:s="http://www.example.com/test">
    <cxf:properties>
      <entry key="dataFormat" value="MESSAGE"/>
      <entry key="setDefaultBus" value="true"/>
    </cxf:properties>
  </cxf:cxfEndpoint>
```

CAMEL-CXF コンポーネントを JAVA.UTIL.LOGGING ではなく LOG4J を使用する方法

CXF のデフォルトのロガーは java.util.logging です。これを log4j に変更する場合は、以下の手順を実行します。META-INF/cxf/org.apache.cxf.logger という名前のクラスパスにファイルを作成します。このファイルには、1 行にコメントがない org.apache.cxf.common.logging.Log4jLogger クラスの完全修飾名を含める必要があります。

HOW TO ALLOW CAMEL-CXF RESPONSE MESSAGE WITH XML START DOCUMENT

PHP などの一部の SOAP クライアントを使用している場合、CXF は XML 開始ドキュメント < ?xml version="1.0" encoding="utf-8"?> を追加しないため、この種のエラーが発生します。

```
Error:sendSms: SoapFault exception: [Client] looks like we got no XML document in [...]
```

この問題を解決するには、XML 開始ドキュメントを書くように StaxOutInterceptor に指示する必要があります。

```
public class WriteXmlDeclarationInterceptor extends AbstractPhaseInterceptor<SoapMessage> {
    public WriteXmlDeclarationInterceptor() {
        super(Phase.PRE_STREAM);
        addBefore(StaxOutInterceptor.class.getName());
    }

    public void handleMessage(SoapMessage message) throws Fault {
        message.put("org.apache.cxf.stax.force-start-document", Boolean.TRUE);
    }

}
```

*このようなカスタマーインターセプターを追加し、**camel-cxf** 終了ポンジーに設定することができます。*

```
<cxf:cxfEndpoint id="routerEndpoint"
address="http://localhost:${CXFTestSupport.port2}/CXFGreeterRouterTest/CamelContext/RouterPort"

  serviceClass="org.apache.hello_world_soap_http.GreeterImpl"
  skipFaultLogging="true">
    <cxf:outInterceptors>
      <!-- This interceptor will force the CXF server send the XML start document to client -->
      <bean class="org.apache.camel.component.cxf.WriteXmlDeclarationInterceptor"/>
    </cxf:outInterceptors>
    <cxf:properties>
      <!-- Set the publishedEndpointUrl which could override the service address from generated
WSDL as you want -->
      <entry key="publishedEndpointUrl" value="http://www.simple.com/services/test" />
    </cxf:properties>
  </cxf:cxfEndpoint>
```

*または、**Camel 2.4** を使用している場合は、このメッセージヘッダーを追加します。*

```
// set up the response context which force start document
Map<String, Object> map = new HashMap<String, Object>();
map.put("org.apache.cxf.stax.force-start-document", Boolean.TRUE);
exchange.getOut().setHeader(Client.RESPONSE_CONTEXT, map);
```

POJO データフォーマットで CAMEL-CXF エンドポイントからメッセージを消費する方法

camel-cxf エンドポイントコンシューマー POJO データ形式は cxf invoker をベースとしているため、メッセージヘッダーには CxfConstants.OPERATION_NAME という名前のプロパティーがあり、メッセージボディーは SEI メソッドパラメーターのリストです。

```
public class PersonProcessor implements Processor {

    private static final transient Logger LOG = LoggerFactory.getLogger(PersonProcessor.class);

    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        LOG.info("processing exchange in camel");

        BindingOperationInfo boi =
(BindingOperationInfo)exchange.getProperty(BindingOperationInfo.class.toString());
        if (boi != null) {
            LOG.info("boi.isUnwrapped" + boi.isUnwrapped());
        }
        // Get the parameters list which element is the holder.
        MessageContentsList msgList = (MessageContentsList)exchange.getIn().getBody();
        Holder<String> personId = (Holder<String>)msgList.get(0);
```

```
        Holder<String> ssn = (Holder<String>)msgList.get(1);
        Holder<String> name = (Holder<String>)msgList.get(2);

        if (personId.value == null || personId.value.length() == 0) {
            LOG.info("person id 123, so throwing exception");
            // Try to throw out the soap fault message
            org.apache.camel.wsdl_first.types.UnknownPersonFault personFault =
                new org.apache.camel.wsdl_first.types.UnknownPersonFault();
            personFault.setPersonId("");
            org.apache.camel.wsdl_first.UnknownPersonFault fault =
                new org.apache.camel.wsdl_first.UnknownPersonFault("Get the null value of person name",
personFault);
            // Since camel has its own exception handler framework, we can't throw the exception to
trigger it
            // We just set the fault message in the exchange for camel-cxf component handling and return
            exchange.getOut().setFault(true);
            exchange.getOut().setBody(fault);
            return;
        }

        name.value = "Bonjour";
        ssn.value = "123";
        LOG.info("setting Bonjour as the response");
        // Set the response message, first element is the return value of the operation,
        // the others are the holders of method parameters
        exchange.getOut().setBody(new Object[] {null, personId, ssn, name});
    }

}
```

POJO データフォーマットの CAMEL-CXF エンドポイントのメッセージを準備する方法

camel-cxf エンドポイントプロデューサーは cxf クライアント API をベースにしています。最初にメッセージヘッダーで操作名を指定し、続いてメソッドパラメーターを一覧に追加し、このパラメーター一覧でメッセージを初期化する必要があります。応答メッセージのボディーは messageContentsList で、そのリストから結果を取得できます。

メッセージヘッダーで操作名を指定しない場合、CxfProducer は CxfEndpoint から defaultOperationName の使用を試行します。CxfEndpoint に defaultOperationName が設定されていない場合、これは操作一覧から最初の操作名を選択します。

メッセージボディーからオブジェクトアレイを取得する場合は、以下のように message.getbody(Object[].class) を使用してボディーを取得できます。

```
Exchange senderExchange = new DefaultExchange(context, ExchangePattern.InOut);
final List<String> params = new ArrayList<String>();
// Prepare the request message for the camel-cxf procedure
params.add(TEST_MESSAGE);
senderExchange.getIn().setBody(params);
```

```
senderExchange.getIn().setHeader(CxfConstants.OPERATION_NAME, ECHO_OPERATION);

Exchange exchange = template.send("direct:EndpointA", senderExchange);

org.apache.camel.Message out = exchange.getOut();
// The response message's body is an MessageContentsList which first element is the return value of
the operation,
// If there are some holder parameters, the holder parameter will be filled in the reset of List.
// The result will be extract from the MessageContentsList with the String class type
MessageContentsList result = (MessageContentsList)out.getBody();
LOG.info("Received output text: " + result.get(0));
Map<String, Object> responseContext = CastUtils.cast((Map<?, ?
>)out.getHeader(Client.RESPONSE_CONTEXT));
assertNotNull(responseContext);
assertEquals("We should get the response context here", "UTF-8",
responseContext.get(org.apache.cxf.message.Message.ENCODING));
assertEquals("Reply body on Camel is wrong", "echo " + TEST_MESSAGE, result.get(0));
```

***PAYLOAD* データフォーマットの *CAMEL-CXF* エンドポイントのメッセージの処理方法**

Apache Camel 2.0: CxfMessage.getBody（） は、SOAP メッセージヘッダーおよびボディー要素の getter がある org.apache.camel.component.cxf.CxfPayload オブジェクトを返します。この変更により、Apache Camel メッセージからネイティブ CXF メッセージを分離できます。

```
protected RouteBuilder createRouteBuilder() {
    return new RouteBuilder() {
        public void configure() {
            from(SIMPLE_ENDPOINT_URI + "&dataFormat=PAYLOAD").to("log:info").process(new
Processor() {
                @SuppressWarnings("unchecked")
                public void process(final Exchange exchange) throws Exception {
                    CxfPayload<SoapHeader> requestPayload =
exchange.getIn().getBody(CxfPayload.class);
                    List<Source> inElements = requestPayload.getBodySources();
                    List<Source> outElements = new ArrayList<Source>();
                    // You can use a customer toStringConverter to turn a CxfPayLoad message into String
as you want
                    String request = exchange.getIn().getBody(String.class);
                    XmlConverter converter = new XmlConverter();
                    String documentString = ECHO_RESPONSE;

                    Element in = new XmlConverter().toDOMElement(inElements.get(0));
                    // Just check the element namespace
                    if (!in.getNamespaceURI().equals(ELEMENT_NAMESPACE)) {
                        throw new IllegalArgumentException("Wrong element namespace");
                    }
                    if (in.getLocalName().equals("echoBoolean")) {
                        documentString = ECHO_BOOLEAN_RESPONSE;
                        checkRequest("ECHO_BOOLEAN_REQUEST", request);
                    } else {
                        documentString = ECHO_RESPONSE;
                        checkRequest("ECHO_REQUEST", request);
                    }
```

```
                Document outDocument = converter.toDOMDocument(documentString);
                outElements.add(new DOMSource(outDocument.getDocumentElement()));
                // set the payload header with null
                CxfPayload<SoapHeader> responsePayload = new CxfPayload<SoapHeader>(null,
outElements, null);
                exchange.getOut().setBody(responsePayload);
            }
        });
    }
};
}
```

POJO モードでの SOAP ヘッダーの取得および設定方法

POJO は、CXF エンドポイントが Camel エクスチェンジを生成または消費するときにデータ形式が Java オブジェクトのリスト であることを意味します。Apache Camel はメッセージボディーをこのモードで POJO として公開しますが、CXF コンポーネントは引き続き SOAP ヘッダーの読み書きへのアクセスを提供します。ただし、CXF インターセプターは処理後にヘッダーリストから In-band SOAP ヘッダーを削除するため、POJO モードで使用できるのは帯域外な SOAP ヘッダーのみです。

以下の例は、SOAP ヘッダーの取得/設定方法を示しています。CXF エンドポイントから別のエンドポイントに転送するルートがあるとします。つまり、SOAP Client → Apache Camel → CXF サービスです。リクエストが CXF サービスに移動し、応答が SOAP クライアントに戻る前に 2 つのプロセッサーを添付して SOAP ヘッダーを取得/挿入できます。この例のプロセッサー(1)および(2)は InsertRequestOutHeaderProcessor および InsertResponseOutHeaderProcessor です。ルートは以下のようになります。

```
<route>
    <from uri="cxf:bean:routerRelayEndpointWithInsertion"/>
    <process ref="InsertRequestOutHeaderProcessor" />
    <to uri="cxf:bean:serviceRelayEndpointWithInsertion"/>
    <process ref="InsertResponseOutHeaderProcessor" />
</route>
```

2.x SOAP ヘッダーは、 Apache Camel Message ヘッダーとの間で伝播されます。Apache Camel メッセージヘッダー名は org.apache.cxf.headers.Header.list で、CXF で定義された定数 (org.apache.cxf.headers.Header.HEADER_LIST)です。ヘッダーの値は、CXF SoapHeader オブジェクト(org.apache.cxf.binding.soap.SoapHeader)の List <> です。以下のスニペットは InsertResponseOutHeaderProcessor です（レスポンスメッセージに新しい SOAP ヘッダーを挿入します）。InsertResponseOutHeaderProcessor と InsertRequestOutHeaderProcessor の両方で SOAP ヘッダーにアクセスする方法は実際に同じです。2 つのプロセッサーの唯一の違いは、挿入された SOAP ヘッダーの方向の設定です。

```
public static class InsertResponseOutHeaderProcessor implements Processor {

    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        // You should be able to get the header if exchange is routed from camel-cxf endpoint
```

```
        List<SoapHeader> soapHeaders = CastUtils.cast((List<?
>)exchange.getIn().getHeader(Header.HEADER_LIST));
        if (soapHeaders == null) {
            // we just create a new soap headers in case the header is null
            soapHeaders = new ArrayList<SoapHeader>();
        }

        // Insert a new header
        String xml = "<?xml version=\"1.0\" encoding=\"utf-8\"?><outofbandHeader "
            + "xmlns=\"http://cxf.apache.org/outofband/Header\" hdrAttribute=\"testHdrAttribute\" "
            + "xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\" soap:mustUnderstand=\"1\">"
            + "<name>New_testOobHeader</name><value>New_testOobHeaderValue</value>
</outofbandHeader>";
        SoapHeader newHeader = new SoapHeader(soapHeaders.get(0).getName(),
                DOMUtils.readXml(new StringReader(xml)).getDocumentElement());
        // make sure direction is OUT since it is a response message.
        newHeader.setDirection(Direction.DIRECTION_OUT);
        //newHeader.setMustUnderstand(false);
        soapHeaders.add(newHeader);

    }

}
```

PAYLOAD モードで SOAP ヘッダーを取得して設定する方法

PAYLOAD モードで SOAP メッセージにアクセスする方法を すでに紹介しています（「PAYLOAD データフォーマットの camel-cxf エンドポイントのメッセージの処理方法」を参照）。

CxfPayload オブジェクトを取得したら、DOM 要素（SOAP ヘッダー）の List を返す CxfPayload.getHeaders（） メソッドを呼び出すことができます。

```
from(getRouterEndpointURI()).process(new Processor() {
    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        CxfPayload<SoapHeader> payload = exchange.getIn().getBody(CxfPayload.class);
        List<Source> elements = payload.getBodySources();
        assertNotNull("We should get the elements here", elements);
        assertEquals("Get the wrong elements size", 1, elements.size());

        Element el = new XmlConverter().toDOMElement(elements.get(0));
        elements.set(0, new DOMSource(el));
        assertEquals("Get the wrong namespace URI", "http://camel.apache.org/pizza/types",
                el.getNamespaceURI());

        List<SoapHeader> headers = payload.getHeaders();
        assertNotNull("We should get the headers here", headers);
        assertEquals("Get the wrong headers size", headers.size(), 1);
        assertEquals("Get the wrong namespace URI",
                ((Element)(headers.get(0).getObject())).getNamespaceURI(),
                "http://camel.apache.org/pizza/types");
```

```
    }

})
.to(getServiceEndpointURI());
```

Camel 2.16.0 以降、「POJO モードでの SOAP ヘッダーの取得および設定方法」 で説明されている手法を使用して SOAP ヘッダーを設定または取得することができます。org.apache.cxf.headers.Header.list ヘッダーを使用して SOAP ヘッダーの一覧を取得および設定できるようになりました。つまり、Camel CXF エンドポイントから別のエンドポイントに転送するルートがある場合（SOAP Client → Camel → CXF サービス）、SOAP クライアントによって送信される SOAP ヘッダーも CXF サービスに転送されます。ヘッダーの転送を希望しない場合は、org.apache.cxf.headers.Header.list Camel ヘッダーから削除します。

SOAP ヘッダーが MESSAGE モードで利用できない

SOAP 処理はスキップされるため、SOAP ヘッダーは MESSAGE モードでは使用できません。

APACHE CAMEL から SOAP FAULT をスローする方法

CXF エンドポイントを使用して SOAP リクエストを消費する場合は、Camel コンテキストから SOAP Fault をスローする必要がある場合があります。基本的に、throwFault DSL を使用してこれを行うことができます。POJO、PAYLOAD、および MESSAGE データフォーマットで機能します。以下のような soap 障害を定義できます。

```
SOAP_FAULT = new SoapFault(EXCEPTION_MESSAGE, SoapFault.FAULT_CODE_CLIENT);
Element detail = SOAP_FAULT.getOrCreateDetail();
Document doc = detail.getOwnerDocument();
Text tn = doc.createTextNode(DETAIL_TEXT);
detail.appendChild(tn);
```

次に、以下のようにスローします。

```
from(routerEndpointURI).setFaultBody(constant(SOAP_FAULT));
```

CXF エンドポイントが MESSAGE データ形式で動作している場合、メッセージボディーに SOAP Fault メッセージを設定し、メッセージヘッダーにレスポンスコードを設定できます。

```
from(routerEndpointURI).process(new Processor() {

    public void process(Exchange exchange) throws Exception {
        Message out = exchange.getOut();
        // Set the message body with the
        out.setBody(this.getClass().getResourceAsStream("SoapFaultMessage.xml"));
```

```
        // Set the response code here
        out.setHeader(org.apache.cxf.message.Message.RESPONSE_CODE, new Integer(500));
    }

});
```

POJO データフォーマットでも同じです。Out ボディーに SOAP Fault を設定し、以下のように Message.setFault(true) を呼び出すことで障害を示すこともできます。

```
from("direct:start").onException(SoapFault.class).maximumRedeliveries(0).handled(true)
    .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
            SoapFault fault = exchange
                .getProperty(Exchange.EXCEPTION_CAUGHT, SoapFault.class);
            exchange.getOut().setFault(true);
            exchange.getOut().setBody(fault);
        }

    }).end().to(serviceURI);
```

CXF エンドポイントのリクエストおよび応答コンテキストを伝播する方法

CXF クライアント API は、リクエストおよび応答コンテキストで操作を呼び出す方法を提供します。CXF エンドポイントプロデューサーを使用して外部 Web サービスを呼び出す場合は、リクエストコンテキストを設定し、以下のコードで応答コンテキストを取得できます。

```
        CxfExchange exchange = (CxfExchange)template.send(getJaxwsEndpointUri(), new
Processor() {
            public void process(final Exchange exchange) {
                final List<String> params = new ArrayList<String>();
                params.add(TEST_MESSAGE);
                // Set the request context to the inMessage
                Map<String, Object> requestContext = new HashMap<String, Object>();
                requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
JAXWS_SERVER_ADDRESS);
                exchange.getIn().setBody(params);
                exchange.getIn().setHeader(Client.REQUEST_CONTEXT , requestContext);
                exchange.getIn().setHeader(CxfConstants.OPERATION_NAME,
GREET_ME_OPERATION);
            }
        });
        org.apache.camel.Message out = exchange.getOut();
        // The output is an object array, the first element of the array is the return value
        Object\[\] output = out.getBody(Object\[\].class);
        LOG.info("Received output text: " + output\[0\]);
        // Get the response context form outMessage
        Map<String, Object> responseContext =
CastUtils.cast((Map)out.getHeader(Client.RESPONSE_CONTEXT));
        assertNotNull(responseContext);
```

```
        assertEquals("Get the wrong wsdl opertion name", "
{http://apache.org/hello_world_soap_http}greetMe",
                responseContext.get("javax.xml.ws.wsdl.operation").toString());
```

添付サポート

POJO モード： SOAP with Attachment and MTOM の両方がサポートされます（MTOM を有効にするための Payload Mode の例を参照してください）。 ただし、SOAP with Attachment はテストされていません。 アタッチメントはマーシャリングされ、POJO にアンマーシャリングされるため、ユーザーは通常添付ファイルを扱う必要はありません。 アタッチメントは 2.1 以降、Camel メッセージの添付ファイルに伝播されます。 そのため、Camel Message API による再試行添付が可能です。

```
DataHandler Message.getAttachment(String id)
```

.

ペイロードモード： MTOM は 2.1 以降サポートされます。アタッチメントは、上記の Camel Message API によって取得できます。このモードでは SOAP 処理がないため、SOAP with Attachment はサポートされません。

MTOM を有効にするには、CXF エンドポイントプロパティー「mtom_enabled」を true に設定します。（Spring でのみ実行できると考えられます。）

```
<cxf:cxfEndpoint id="routerEndpoint"
address="http://localhost:${CXFTestSupport.port1}/CxfMtomRouterPayloadModeTest/jaxws-
mtom/hello"
        wsdlURL="mtom.wsdl"
        serviceName="ns:HelloService"
        endpointName="ns:HelloPort"
        xmlns:ns="http://apache.org/camel/cxf/mtom_feature">

    <cxf:properties>
        <!--  enable mtom by setting this property to true -->
        <entry key="mtom-enabled" value="true"/>

        <!--  set the camel-cxf endpoint data fromat to PAYLOAD mode -->
        <entry key="dataFormat" value="PAYLOAD"/>
    </cxf:properties>
```

Payload モードの CXF エンドポイントに送信する attachment のある Camel メッセージを生成できます。

```
Exchange exchange = context.createProducerTemplate().send("direct:testEndpoint", new
Processor() {
```

```
    public void process(Exchange exchange) throws Exception {
        exchange.setPattern(ExchangePattern.InOut);
        List&lt;Source> elements = new ArrayList&lt;Source>();
        elements.add(new DOMSource(DOMUtils.readXml(new
StringReader(MtomTestHelper.REQ_MESSAGE)).getDocumentElement()));
        CxfPayload<SoapHeader> body = new CxfPayload<SoapHeader>(new ArrayList<SoapHeader>
(),
            elements, null);
        exchange.getIn().setBody(body);
        exchange.getIn().addAttachment(MtomTestHelper.REQ_PHOTO_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.REQ_PHOTO_DATA,
"application/octet-stream")));

        exchange.getIn().addAttachment(MtomTestHelper.REQ_IMAGE_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.requestJpeg, "image/jpeg")));

    }

});

// process response

CxfPayload<SoapHeader> out = exchange.getOut().getBody(CxfPayload.class);
Assert.assertEquals(1, out.getBody().size());

Map<String, String> ns = new HashMap<String, String>();
ns.put("ns", MtomTestHelper.SERVICE_TYPES_NS);
ns.put("xop", MtomTestHelper.XOP_NS);

XPathUtils xu = new XPathUtils(ns);
Element oute = new XmlConverter().toDOMElement(out.getBody().get(0));
Element ele = (Element)xu.getValue("//ns:DetailResponse/ns:photo/xop:Include", oute,
                        XPathConstants.NODE);
String photoId = ele.getAttribute("href").substring(4); // skip "cid:"

ele = (Element)xu.getValue("//ns:DetailResponse/ns:image/xop:Include", oute,
                        XPathConstants.NODE);
String imageId = ele.getAttribute("href").substring(4); // skip "cid:"

DataHandler dr = exchange.getOut().getAttachment(photoId);
Assert.assertEquals("application/octet-stream", dr.getContentType());
MtomTestHelper.assertEquals(MtomTestHelper.RESP_PHOTO_DATA,
IOUtils.readBytesFromStream(dr.getInputStream()));

dr = exchange.getOut().getAttachment(imageId);
Assert.assertEquals("image/jpeg", dr.getContentType());

BufferedImage image = ImageIO.read(dr.getInputStream());
Assert.assertEquals(560, image.getWidth());
Assert.assertEquals(300, image.getHeight());
```

Payload モードの CXF エンドポイントから受信した Camel メッセージを消費することもできます。

```
public static class MyProcessor implements Processor {

    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        CxfPayload<SoapHeader> in = exchange.getIn().getBody(CxfPayload.class);

        // verify request
        assertEquals(1, in.getBody().size());

        Map<String, String> ns = new HashMap<String, String>();
        ns.put("ns", MtomTestHelper.SERVICE_TYPES_NS);
        ns.put("xop", MtomTestHelper.XOP_NS);

        XPathUtils xu = new XPathUtils(ns);
        Element body = new XmlConverter().toDOMElement(in.getBody().get(0));
        Element ele = (Element)xu.getValue("//ns:Detail/ns:photo/xop:Include", body,
                                   XPathConstants.NODE);
        String photoId = ele.getAttribute("href").substring(4); // skip "cid:"
        assertEquals(MtomTestHelper.REQ_PHOTO_CID, photoId);

        ele = (Element)xu.getValue("//ns:Detail/ns:image/xop:Include", body,
                                   XPathConstants.NODE);
        String imageId = ele.getAttribute("href").substring(4); // skip "cid:"
        assertEquals(MtomTestHelper.REQ_IMAGE_CID, imageId);

        DataHandler dr = exchange.getIn().getAttachment(photoId);
        assertEquals("application/octet-stream", dr.getContentType());
        MtomTestHelper.assertEquals(MtomTestHelper.REQ_PHOTO_DATA,
IOUtils.readBytesFromStream(dr.getInputStream()));

        dr = exchange.getIn().getAttachment(imageId);
        assertEquals("image/jpeg", dr.getContentType());
        MtomTestHelper.assertEquals(MtomTestHelper.requestJpeg,
IOUtils.readBytesFromStream(dr.getInputStream()));

        // create response
        List&lt;Source> elements = new ArrayList&lt;Source>();
        elements.add(new DOMSource(DOMUtils.readXml(new
StringReader(MtomTestHelper.RESP_MESSAGE)).getDocumentElement()));
        CxfPayload&lt;SoapHeader> sbody = new CxfPayload&lt;SoapHeader>(new
ArrayList&lt;SoapHeader>(),
            elements, null);
        exchange.getOut().setBody(sbody);
        exchange.getOut().addAttachment(MtomTestHelper.RESP_PHOTO_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.RESP_PHOTO_DATA,
"application/octet-stream")));

        exchange.getOut().addAttachment(MtomTestHelper.RESP_IMAGE_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.responseJpeg, "image/jpeg")));

    }
}
```

***Message Mode:** 接続は、メッセージをまったく処理しないため、サポートされていません。*

CXF_MESSAGE Mode: MTOM はサポートされ、上記の Camel Message API によって接続を取得できます。マルチパート(MTOM)メッセージを受信すると、デフォルトの SOAPMessage to String コンバーターはボディーに完全なマルチパートペイロードを提供することに注意してください。SOAP XML を String としてのみ必要な場合は、メッセージボディーを message.getSOAPPart（） で設定でき、Camel convert は残りの作業を行うことができます。

スタックトレース情報を伝播させる方法

Java の例外がサーバー側で発生したときに、例外のスタックトレースがフォールトメッセージにマーシャリングされてクライアントに返されるように、CXF エンドポイントを設定することができます。この機能を有効にするには、以下のように cxfEndpoint 要素で、dataFormat を PAYLOAD に設定し、faultStackTraceEnabled プロパティーを true に設定します。

```
<cxf:cxfEndpoint id="router" address="http://localhost:9002/TestMessage"
    wsdlURL="ship.wsdl"
    endpointName="s:TestSoapEndpoint"
    serviceName="s:TestService"
    xmlns:s="http://test">
  <cxf:properties>
    <!-- enable sending the stack trace back to client; the default value is false-->
    <entry key="faultStackTraceEnabled" value="true" /> <entry key="dataFormat" value="PAYLOAD"
/>
  </cxf:properties>
</cxf:cxfEndpoint>
```

セキュリティー上の理由から、スタックトレースには原因となる例外 (つまりスタックトレースの Caused by 以降の部分) は含まれません。スタックトレースに原因となる例外を含めたい場合は、以下のように cxfEndpoint 要素の exceptionMessageCauseEnabled プロパティーを true に設定します。

```
<cxf:cxfEndpoint id="router" address="http://localhost:9002/TestMessage"
    wsdlURL="ship.wsdl"
    endpointName="s:TestSoapEndpoint"
    serviceName="s:TestService"
    xmlns:s="http://test">
  <cxf:properties>
    <!-- enable to show the cause exception message and the default value is false -->
    <entry key="exceptionMessageCauseEnabled" value="true" />
    <!-- enable to send the stack trace back to client,  the default value is false-->
    <entry key="faultStackTraceEnabled" value="true" />
    <entry key="dataFormat" value="PAYLOAD" />
  </cxf:properties>
</cxf:cxfEndpoint>
```

警告

exceptionMessageCauseEnabled フラグは、テストおよび診断目的でのみ有効にしてください。サーバにおいて例外の元の原因を隠すことで、敵対的なユーザーがサーバーを調査しにくくするのが、通常の実践的なやり方です。

PAYLOAD モードでのサポートのストリーミング

2.8.2 では、camel-cxf コンポーネントは PAYLOAD モードを使用する場合に受信メッセージのストリーミングをサポートするようになりました。以前のバージョンでは、受信メッセージは完全に DOM 解析されました。メッセージが大きい場合は、時間がかかるため、大量のメモリーが使用されます。2.8.2 以降では、受信メッセージはルーティング中に javax.xml.transform.Source として残ることができます。ペイロードが変更されていない場合は、ターゲットの宛先に直接ストリーミングすることができます。一般的な「simple proxy」のユースケース（例： from("cxf:...").to("cxf:...")については、パフォーマンスが大幅に向上し、メモリーの要件が大幅に低下する可能性があります。

ただし、ストリーミングが適切でない場合もあります。ストリーミングの性質上、無効な受信 XML が処理チェーンの後半までキャッチされないことがあります。また、特定のアクションでは、メッセージを DOM に解析（WS-Security やメッセージトレースなど）する必要があります。この場合、ストリーミングの利点は制限されます。この時点で、ストリーミングを制御する 2 つの方法があります。

- *エンドポイントプロパティー： "allowStreaming=false" をエンドポイントプロパティーとして追加し、ストリーミングのオン/オフをオン/オフにできます。*
- *コンポーネントプロパティー： CxfComponent オブジェクトには、コンポーネントから作成されるエンドポイントのデフォルトを設定できる allowStreaming プロパティーもあります。*
- *グローバルシステムプロパティー： オフの場合は、"org.apache.camel.component.cxf.streaming" のシステムプロパティーを "false" に追加して有効にすることができます。これはグローバルデフォルトを設定しますが、上記のエンドポイントプロパティーを設定すると、そのエンドポイントのこの値が上書きされます。*

汎用 CXF DISPATCH モードの使用

2.8.0 以降、camel-cxf コンポーネントは、任意の構造のメッセージを転送できる汎用 CXF ディスパッチモード をサポートします（つまり、特定の XML スキーマにバインドされません）。このモード

を使用するには、CXF エンドポイントの wsdlURL および serviceClass 属性を指定するだけです。

```
<cxf:cxfEndpoint id="testEndpoint" address="http://localhost:9000/SoapContext/SoapAnyPort">
  <cxf:properties>
    <entry key="dataFormat" value="PAYLOAD"/>
  </cxf:properties>
</cxf:cxfEndpoint>
```

デフォルトの CXF ディスパッチクライアントは特定の SOAPAction ヘッダーを送信しないことに注意してください。したがって、ターゲットサービスに特定の SOAPAction 値が必要な場合は、SOAPAction(case-insensitive)キーを使用して Camel ヘッダーに提供されます。

75.1. JBOSS ENTERPRISE APPLICATION PLATFORM での CXF コンシューマー

JBoss Enterprise Application Platform での camel-cxf コンシューマーの設定は、スタンドアロン Camel とは異なります。プロデューサーエンドポイントは通常どおり機能します。

JBoss Enterprise Application Platform では、camel-cxf コンシューマーはコンテナーによって提供されるデフォルトの Undertow HTTP サーバーを使用します。サーバーは undertow サブシステム設定内に定義されます。以下は、standalone.xml のデフォルト設定の抜粋です。

```
<subsystem xmlns="urn:jboss:domain:undertow:4.0">
    <buffer-cache name="default" />
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true" />
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true" />
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content" />
            <filter-ref name="server-header" />
            <filter-ref name="x-powered-by-header" />
            <http-invoker security-realm="ApplicationRealm" />
        </host>
    </server>
</subsystem>
```

この場合、Undertow は http および https socket-binding によって指定されたインターフェース/ポートをリッスンするように設定されます。デフォルトでは、これは http のポート 8080、https の場合は 8443 です。

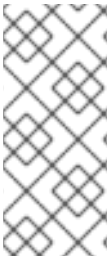
たとえば、異なるホストまたはポートの組み合わせを使用してエンドポイントコンシューマーを設定すると、警告がサーバーログファイルに表示されます。たとえば、以下のホストおよびポート設定は無視されます。

```
<cxf:rsServer id="cxfRsConsumer"
              address="http://somehost:1234/path/to/resource"
              serviceClass="org.example.ServiceClass" />
```

```
<cxf:cxfEndpoint id="cxfWsConsumer"
                 address="http://somehost:1234/path/to/resource"
                 serviceClass="org.example.ServiceClass" />
```

```
[org.wildfly.extension.camel] (pool-2-thread-1) Ignoring configured host:
http://somehost:1234/path/to/resource
```

ただし、コンシューマーはデフォルトのホストおよびポート **localhost:8080** または **localhost:8443** で引き続き利用できます。



注記

camel-cxf コンシューマーを使用するアプリケーションは **WAR** としてパッケージ化する 必要があり ます。以前の **Fuse on JBoss EAP** リリースでは、**JAR** などの他のタイプのアーカイブの使用は許可されますが、 サポート対象外になりました。

75.1.1. 代替ポートの設定

代替ポートが許可される場合は、**JBoss Enterprise Application Platform** サブシステム設定を使用して設定する必要があります。詳細は、 サーバーのドキュメントで説明されています。

https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/7.1/html/configuration_guide/configuring_the_web_server_undertow

75.1.2. SSL の設定

SSL を設定するには、**JBoss Enterprise Application Platform** の『**SSL** 設定ガイド』を参照してください。

https://access.redhat.com/documentation/ja-jp/red_hat_jboss_enterprise_application_platform/7.1/html-single/how_to_configure_server_security/#configure_one_way_and_two_way_ssl_tls_for_application

75.1.3. Elytron を使用したセキュリティー設定

Fuse on JBoss EAP は、Elytron セキュリティーフレームワークでの camel-cxf コンシューマーエンドポイントのセキュア化をサポートします。

75.1.3.1. セキュリティードメインの設定

Elytron で Fuse on JBoss EAP アプリケーションをセキュアにするには、WAR デプロイメントの WEB-INF/jboss-web.xml 内でアプリケーションセキュリティードメインを参照する必要があります。

```
<jboss-web>
  <security-domain>my-application-security-domain</security-domain>
</jboss-web>
```

&lt ;security-domain> 設定は、Undertow サブシステムによって定義される < application-security-domain& gt; の名前を参照します。たとえば、Undertow サブシステム < application-security-domain > は、以下のように JBoss Enterprise Application Platform サーバーの standalone.xml 設定ファイル内に設定されます。

```
<subsystem xmlns="urn:jboss:domain:undertow:6.0">
    ...
    <application-security-domains>
        <application-security-domain name="my-application-security-domain" http-authentication-factory="application-http-authentication"/>
    </application-security-domains>
</subsystem>
```

&lt ;http-authentication-factory> application-http-authentication は Elytron サブシステム内に定義されます。application-http-authentication は、standalone.xml および standalone-full.xml サーバー設定ファイルの両方でデフォルトで利用できます。以下に例を示します。

```
<subsystem xmlns="urn:wildfly:elytron:1.2">
    ...
    <http>
        ...
        <http-authentication-factory name="application-http-authentication" http-server-mechanism-factory="global" security-domain="ApplicationDomain">
            <mechanism-configuration>
                <mechanism mechanism-name="BASIC">
                    <mechanism-realm realm-name="Application Realm" />
                </mechanism>
                <mechanism mechanism-name="FORM" />
            </mechanism-configuration>
        </http-authentication-factory>
        <provider-http-server-mechanism-factory name="global" />
    </http>
    ...
</subsystem>
```

application -http-authentication という名前の <http-authentication-factory > は、ApplicationDomain という Elytron セキュリティードメインへの参照を保持します。

Elytron サブシステムの設定方法の詳細は、Elytron の ドキュメントを参照してください。

75.1.3.2. セキュリティー制約、認証方法、およびセキュリティーロールの設定

camel-cxf コンシューマーエンドポイントのセキュリティー制約、認証方法、およびセキュリティーロールは、WAR デプロイメント WEB-INF/web.xml 内で設定できます。たとえば、BASIC 認証を設定するには、次のコマンドを実行します。

```
<web-app>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>secure</web-resource-name>
      <url-pattern>/webservices/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>my-role</role-name>
    </auth-constraint>
  </security-constraint>
  <security-role>
    <description>The role that is required to log in to /webservices/*</description>
    <role-name>my-role</role-name>
  </security-role>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>my-realm</realm-name>
  </login-config>
</web-app>
```

Servlet Specification で定義された < url-pattern > は Web アプリケーションのコンテキストパスに相対することに注意してください。アプリケーションが my-app.war としてパッケージされている場合、JBoss Enterprise Application Platform はコンテキストパス /my-app でアクセスできるようになり、< url-patternpattern > /webservices/* は /my-app への相対パスに適用されます。

たとえば、http://my-server/my-app/webservices/my-endpoint に対するリクエストは /webservices/* パターンと一致しますが、http://my-server/webservices/my-endpoint は一致しません。

これは、Fuse on JBoss EAP ではベースパスがホスト Web アプリケーションコンテキストパス外にある camel-cxf エンドポイントコンシューマーを作成することができるため重要です。たとえば、my-app.war 内に http://my-server/webservices/my-endpoint の camel-cxf コンシューマーを作成できます。

このようなコンテキストエンドポイントのセキュリティー制約を定義するために、Fuse on JBoss EAP はカスタム 標準以外 の <url-pattern & gt; 規則をサポートします。たとえば、my-app.war 内で http://my-server/webservices/my-endpoint をセキュアにするには、以下の設定を web.xml に追加します。

```
<web-app>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>secure</web-resource-name>
      <url-pattern>///webservices/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>my-role</role-name>
    </auth-constraint>
  </security-constraint>
  <security-role>
    <description>The role that is required to log in to /webservices/*</description>
    <role-name>my-role</role-name>
  </security-role>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>my-realm</realm-name>
  </login-config>
</web-app>
```

第76章 CXF-RS コンポーネント

Camel バージョン 2.0 で利用可能

cxfrs: コンポーネントは、CXF でホストされる JAX-RS 1.1 および 2.0 サービスに接続するための Apache CXF との統合を提供します。

CXF をコンシューマーとして使用する場合、CXF Bean コンポーネントを 使用すると、処理からメッセージペイロードを RESTful または SOAP Web サービスとして受信する方法をファクトアウトできます。これは、多数トランスポートを使用して Web サービスを消費する可能性があります。Bean コンポーネントの設定はシンプルで、Camel および CXF を使用して Web サービスを実装する最も高速な方法を提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cxf</artifactId>
  <version>x.x.x</version>  <!-- use the same version as your Camel core version -->
</dependency>
```

76.1. URI 形式

```
cxfrs://address?options
```

ここでの address は CXF エンドポイントのアドレスを表します。

```
cxfrs:bean:rsEndpoint
```

rsEndpoint は、CXFRS クライアントまたはサーバーを提供する Spring Bean の名前を表します。

上記のいずれかのスタイルで、以下のように URI にオプションを追加できます。

```
cxfrs:bean:cxfEndpoint?resourceClasses=org.apache.camel.rs.Example
```

76.2. オプション

CXF-RS コンポーネントは以下の 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

CXF-RS エンドポイントは URI 構文を使用します。

```
cxfrs:beanId:address
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

76.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
beanId	既存の設定済みの CxfRsEndpoint を検索する。Bean: をプレフィックスとして使用する必要があります。		文字列
address	サービスの公開アドレス。		文字列

76.2.2. クエリーパラメーター（29 パラメーター）：

Name	説明	デフォルト	Type
機能 (common)	機能一覧を CxfRs エンドポイントに設定します。		リスト

Name	説明	デフォルト	Type
loggingFeatureEnabled (common)	このオプションは、インバウンドおよびアウトバウンドの REST メッセージをログに記録する CXF ロギング機能を有効にします。	false	boolean
loggingSizeLimit (common)	ロギング機能が有効になっているときにロガーが出力されるバイト数の合計サイズを制限します。		int
modelRef (common)	このオプションは、アノテーションなしでリソースクラスに役立つモデルファイルを指定するために使用されます。このオプションを使用する場合、サービスクラスを省略してドキュメントのみのエンドポイントをエミュレートできます。		文字列
providers (common)	カスタム JAX-RS プロバイダーを CxfRs エンドポイントに設定します。コンマで区切られた、検索するプロバイダーの一覧が含まれる文字列を指定できます。		文字列
resourceClasses (common)	REST サービスとしてエクスポートするリソースクラス。複数のクラスをコンマで区切ることができます。		リスト
schemaLocations (common)	受信 XML または JAXB 駆動 JSON の検証に使用できるスキーマの場所を設定します。		リスト
skipFaultLogging (common)	このオプションは、PhaseInterceptorChain がキャッチするフォールトのロギングをスキップするかどうかを制御します。	false	boolean

Name	説明	デフォルト	Type
bindingStyle (consumer)	Camel からリクエストおよび応答をマッピングする方法を設定します。SimpleConsumer の 2 つの値を使用できます。このバインディングスタイルはリクエストパラメーター、マルチパートなどを処理し、それらを IN ヘッダー、IN アタッチ、およびメッセージボディーにマッピングします。これは、org.apache.cxf.message.MessageContentsList の低レベルの処理を排除することを目指しています。また、レスポンスマッピングにより、より柔軟で簡素化されます。コンシューマーでのみ利用可能です。デフォルト： デフォルトのスタイル。コンシューマーの場合、MessageContentsList でルートに渡され、ルートに低レベルの処理が必要です。これは従来のバインディングスタイルで、CXF スタックから送信される org.apache.cxf.message.MessageContentsList を IN メッセージボディーにダンプします。その後、ユーザーは JAX-RS メソッド署名によって定義されたコントラクトに従って処理されます。custom: バインディングオプションを使用してカスタムバインディングを指定できます。	デフォルト	BindingStyle
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
publishedEndpointUrl (consumer)	このオプションは、WADL から公開される endpointUrl を上書きできます。これは、リソースアドレス url および _wadl でアクセスできます。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cookieHandler (producer)	HTTP セッションを維持するためのクッキーハンドラーの設定		CookieHandler

Name	説明	デフォルト	Type
hostnameVerifier (producer)	使用するホスト名検証子。表記を使用して、レジストリーから HostnameVerifier を参照します。		HostnameVerifier
sslContextParameters (producer)	Camel SSL 設定のリファレンス。表記を使用して SSL コンテキストを参照します。		SSLContextParameters
throwExceptionOnFailure (producer)	このオプションは、CxfRsProducer に対して戻りコードを検査するように指示し、戻りコードが 207 を超える場合は例外を生成します。	true	boolean
httpClientAPI (producer)	true の場合、CxfRsProducer は HttpClientAPI を使用してサービスを呼び出します。false の場合、CxfRsProducer は ProxyClientAPI を使用してサービスを呼び出します。	true	boolean
ignoreDeleteMethodMessageBody (producer)	このオプションは、HTTP API を使用する際に DELETE メソッドのメッセージボディーを無視するように CxfRsProducer に指示するために使用されます。	false	boolean
maxClientCacheSize (producer)	このオプションを使用すると、キャッシュの最大サイズを設定できます。この実装は、CxfProvider および CxfRsProvider で CXF クライアントまたは ClientFactoryBean をキャッシュします。	10	int
バインディング（詳細）	カスタム CxfBinding を使用して Camel Message と CXF Message 間のバインディングを制御する場合。		CxfRsBinding
bus (advanced)	カスタムが設定された CXF バスを使用するには、以下を行います。		バス
continuationTimeout (advanced)	このオプションは、CXF サーバーが Jetty または Servlet トランスポートを使用している場合にデフォルトで CxfConsumer で使用できる CXF 継続タイムアウトを設定するために使用されます。	30000	Long
cxfRsEndpointConfigurer (advanced)	このオプションは、プログラムによる CXF エンドポイントの設定をサポートする org.apache.camel.component.cxf.jaxrs.CxfRsEndpointConfigurer の実装を適用する可能性があります。ユーザーは、CxfEndpointConfigurer の configureServer/Client メソッドを実装して、CXX サーバーおよびクライアントを設定できます。		CxfRsEndpointConfigurer
defaultBus (advanced)	CXF エンドポイント自体がバスを作成すると、デフォルトのバスを設定します。	false	boolean

Name	説明	デフォルト	Type
headerFilterStrategy (advanced)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
performInvocation (advanced)	オプションが true の場合、Camel はリソースクラスインスタンスの呼び出しを実行し、さらに処理するために応答オブジェクトをエクスチェンジに配置します。	false	boolean
propagateContexts (advanced)	オプションが true の場合、JAXRS UriInfo、HttpHeaders、Request、および SecurityContext コンテキストは、型化された Camel エクスチェンジプロパティーとしてカスタム CXFRS プロセッサーで利用できます。これらのコンテキストを使用して、JAX-RS API を使用して現在のリクエストを分析できます。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

spring 設定を介して CXF REST エンドポイントを設定することもできます。CXF REST クライアントと CXF REST Server には多くの違いがあるため、それらに異なる設定を提供します。詳細は、スキーマファイル と CXF JAX-RS ドキュメント を参照してください。

76.3. CAMEL での REST エンドポイントの設定方法

camel-cxf スキーマファイル では、REST エンドポイント定義に 2 つの要素があります。CXF :rsServer for REST consumer, cxf:rsClient for REST producer.
ここでは、Camel REST サービスルート設定の例 があります。

76.4. メッセージヘッダーからの CXF プロデューサーアドレスを上書きする方法

camel-cxfrs プロデューサーは、「CamelDestinationOverrideUrl」のキーでメッセージを設定することにより、サービスアドレスをオーバーライドすることをサポートします。

```
// set up the service address from the message header to override the setting of CXF endpoint
 exchange.getIn().setHeader(Exchange.DESTINATION_OVERRIDE_URL,
constant(getServiceAddress()));
```

76.5. REST 要求の消費 - シンプルバインディングスタイル

Camel 2.11 から利用可能

Default バインディングスタイルは低レベルではなく、ユーザーはルートに送信される MessageContentsList オブジェクトを手動で処理する必要があります。そのため、ルートロジックをメソッド signature および JAX-RS 操作のパラメーターインデックスと密接にまとめます。さまざまで、困難なエラーとエラーの妥当性があります。

一方、SimpleConsumer バインディングスタイルは、Camel Message 内で リクエストデータがよりアクセスできるように するために、以下のマッピングを実行します。

- *JAX-RS パラメーター（@HeaderParam、@QueryParam など）は IN メッセージヘッダーとしてインジェクトされます。ヘッダー名はアノテーションの値に一致します。*
- *リクエストエンティティー（POJO またはその他のタイプ）は IN メッセージボディーになります。単一のエンティティーを JAX-RS メソッド署名で特定できない場合、元の MessageContentsList にフォールバックします。*
- *バイナリー @Multipart ボディーの一部は IN メッセージの添付、DataHandler、InputStream、DataSource、および CXF の Attachment クラスをサポートします。*
- *バイナリー以外の @Multipart ボディーの部分は IN メッセージヘッダーとしてマッピングされます。ヘッダー名は Body Part 名と一致します。*

また、レスポンスマッピング には、以下のルールが適用されます。

- *メッセージボディータイプが javax.ws.rs.core.Response （ユーザービルド応答）と異なる場合は、新しい Response が作成され、メッセージボディーがエンティティーとして設定されます（つまり、null でない場合）。応答ステータスコードは Exchange.HTTP_RESPONSE_CODE ヘッダーから取得されます。*
- *メッセージボディータイプが javax.ws.rs.core.Response と等しい場合は、ユーザーがカスタム応答を構築しているため、最終応答になります。*
- *いずれの場合も、カスタムまたはデフォルトの HeaderFilterStrategy によって許可される Camel ヘッダーが HTTP 応答に追加されます。*

76.5.1. シンプルバインディングスタイルの有効化

このバインディングスタイルは、コンシューマーエンドポイントの bindingStyle パラメーターを SimpleConsumer の値に設定することでアクティベートできます。

```
from("cxfrs:bean:rsServer?bindingStyle=SimpleConsumer")
  .to("log:TEST?showAll=true");
```

76.5.2. 異なるメソッド署名を使用した要求バインディングの例

以下は、Simple バインディングからの予想される結果と共にメソッドシグネチャーの一覧です。

public Response doAction(BusinessObject request);
Request payload is placed in IN message body, replace the original MessageContentsList.

public Response doAction（BusinessObject request, @HeaderParam("abcd")String abcd, @QueryParam("defg")String defg）; Request payload placed in IN message body, replace the original MessageContentsList.両方のリクエストパラメーターが IN メッセージヘッダーとしてマッピングされ、名前が abcd および defg になります。

public Response doAction（@HeaderParam("abcd")String abcd, @QueryParam("defg")String defg）; Both request params mapped as IN message headers with names abcd and defg.元の MessageContentsList には 2 つのパラメーターのみが含まれている場合でも保持されます。

public Response doAction（@Multipart(value="body1")BusinessObject request, @Multipart(value="body2")BusinessObject request2）; 最初のパラメーターは、body1 という名前のヘッダーとして転送され、2 つ目のパラメーターはヘッダー本文 2 としてマッピングされます。元の MessageContentsList は IN メッセージのボディーとして保持されます。

public Response doAction(InputStream abcd); The InputStream is unwrapped from the MessageContentsList and preserved as the IN message body.

public Response doAction(DataHandler abcd); DataHandler は MessageContentsList からラップされず、IN メッセージボディーとして保持されます。

76.5.3. シンプルバインディングスタイルの他の例

JAX-RS リソースクラスにこのメソッドがある場合：

```
@POST @Path("/customers/{type}")
public Response newCustomer(Customer customer, @PathParam("type") String type,
@QueryParam("active") @DefaultValue("true") boolean active) {
    return null;
}
```

以下のルートで提供されます。

```
from("cxfrs:bean:rsServer?bindingStyle=SimpleConsumer")
    .recipientList(simple("direct:${header.operationName}"));

from("direct:newCustomer")
    .log("Request: type=${header.type}, active=${header.active}, customerData=${body}");
```

XML ペイロードを使用する以下の HTTP リクエスト（カスタマー DTO が JAXB にアノテーションが付けられていることを前提とします）。

```
POST /customers/gold?active=true

Payload:
<Customer>
  <fullName>Raul Kripalani</fullName>
  <country>Spain</country>
  <project>Apache Camel</project>
</Customer>
```

メッセージを出力します。

```
Request: type=gold, active=true, customerData=<Customer.toString() representation>
```

リクエストおよび書き込み応答の処理方法の例は、 を参照して ください。

76.6. REST 要求の消費： デフォルトのバインディングスタイル

CXF JAXRS フロントエンド は JAX-RS(JSR-311)API を実装するため、リソースクラスを REST サービスとしてエクスポートできます。そして、CXF Invoker API を利用し て REST リクエストを通常の Java オブジェクトメソッド呼び出しに変換します。
Camel Restlet コンポーネントとは異なり、エンドポイント内で URI テンプレートを指定する必要はありません。CXF は JSR-311 仕様に従って REST リクエスト URI をリソースクラスメソッドへのマッピ

ングに扱います。Camel で必要となるのは、このメソッドリクエストを正しいプロセッサーまたはエンドポイントへ委譲することです。

CXFRS route...

エンドポイントの設定に使用される対応するリソースクラス

INFO:*Note about resource classes*

デフォルトでは、JAX-RS リソースクラスは JAX-RS プロパティーを設定するために のみ* 使用されます。メッセージをエンドポイントにルーティングする際に、メソッドが *実行 されません。代わりに、ルートが全処理を行う役割を担います。

Camel 2.15 以降では、デフォルトモードの no-op サービス実装クラスではなく、インターフェースのみを提供するだけで十分です。

Camel 2.15 以降、performInvocation オプションが有効になっていると、サービス実装が最初に呼び出されると、レスポンスが Camel エクスチェンジに設定され、ルートの実行は通常通り続行されます。これは、既存の JAX-RS 実装を Camel ルートに統合したり、カスタムプロセッサーの JAX-RS 応答を処理する場合に便利です。

76.7. CAMEL-CXFRS プロデューサーを介して REST サービスを呼び出す方法

CXF JAXRS フロントエンド は プロキシーベースのクライアント API を実装し、この API を使用してリモート REST サービスをプロキシー経由で呼び出すことができます。camel-cxfrs プロデューサーはこの プロキシー API をベースにしています。
メッセージヘッダーで操作名を指定し、メッセージボディーでパラメーターを準備するだけで、camel-cxfrs プロデューサーは適切な REST リクエストを生成します。

以下は例です。

CXF JAXRS フロントエンド は、http 中心クライアント API も提供し ます。 この API を camel-cxfrs プロデューサーから呼び出すこともできます。HTTP_PATH と HTTP_METHOD を指定し、URI オプション httpClientAPI を使用するか、メッセージヘッダー CxfConstants.CAMEL_CXF_RS_USING_HTTP_API を設定して、プロデューサーが http 中心のクライアント API を使用できるようにします。応答オブジェクトを、メッセージヘッ

ダー CxfConstants.CAMEL_CXF_RS_RESPONSE_CLASS で指定されたタイプクラスに変換できます。

Camel 2.1 以降では、CXFRS http 中心クライアントの cxfrs URI からクエリーパラメーターを指定するサポートもサポートしています。

Error formatting macro: snippet: java.lang.IndexOutOfBoundsException: Index: 20, Size: 20

Dynamical ルーティングをサポートするには、CxfConstants.CAMEL_CXF_RS_QUERY_MAP ヘッダーを使用して URI のクエリーパラメーターを上書きして、そのパラメーターマップを設定できます。

76.8. WHAT'S THE CAMEL TRANSPORT FOR CXF（CXF の CAMEL トランスポート）

CXF では、アドレスを定義して Web サービスを提供または消費します。アドレスの最初の部分は、使用するプロトコルを指定します。エンドポイント設定の address="http://localhost:9000" は、ローカルホストのポート 9000 で http プロトコルを使用してサービスを提供することを意味します。Camel Tranport を CXF に統合すると、新しいトランスポート「camel」が取得されます。そのため、address="camel://direct:MyEndpointName" を指定して CXF サービスアドレスを camel ダイレクトエンドポイントにバインドできます。

技術的には、CXF トランスポートの Camel トランスポート を Camel コアライブラリーと実装するコンポーネントです。これにより、Camel のルーティングエンジンや統合パターンのサポートと CXF サービスを簡単に使用できます。

76.9. CAMEL の CXF トランスポート層との統合

Camel Tranport を CXF バスに含めるには、CamelTransportFactory を使用します。これは、Java および Spring で実行できます。

76.9.1. Spring での Camel トランスポートの設定

特別な設定を行う場合は、applicationcontext で以下のスニペットを使用できます。camel トランスポートのみをアクティベートする場合は、アプリケーションコンテキストで何もする必要はありません。アプリに camel-cxf-transport jar（または camel バージョンが 2.7.x 未満の場合は camel-cxf.jar）が含まれる場合すぐに、cxf は jar をスキャンし、CamelTransportFactory を読み込みます。

```
<!-- you don't need to specify the CamelTransportFactory configuration as it is auto load by
CXF bus -->
```

```
<bean class="org.apache.camel.component.cxf.transport.CamelTransportFactory">
  <property name="bus" ref="cxf" />
  <property name="camelContext" ref="camelContext" />
  <!-- checkException new added in Camel 2.1 and Camel 1.6.2 -->
  <!-- If checkException is true , CamelDestination will check the outMessage's
     exception and set it into camel exchange. You can also override this value
     in CamelDestination's configuration. The default value is false.
     This option should be set true when you want to leverage the camel's error
     handler to deal with fault message -->
  <property name="checkException" value="true" />
  <property name="transportIds">
    <list>
      <value>http://cxf.apache.org/transports/camel</value>
    </list>
  </property>
</bean>
```

76.9.2. Camel トランスポートのプログラムによる統合

Camel トランスポートは、Camel コンテキストをトランスポートファクトリーに設定するのに使用できる setContext メソッドを提供します。このファクトリーを有効にしたい場合は、ファクトリーを CXF バスに登録する必要があります。以下は、完全な例です。

```
import org.apache.cxf.Bus;
import org.apache.cxf.BusFactory;
import org.apache.cxf.transport.ConduitInitiatorManager;
import org.apache.cxf.transport.DestinationFactoryManager;
...

BusFactory bf = BusFactory.newInstance();
Bus bus = bf.createBus();
CamelTransportFactory camelTransportFactory = new CamelTransportFactory();
// set up the CamelContext which will be use by the CamelTransportFactory
camelTransportFactory.setCamelContext(context)
// if you are using CXF higher then 2.4.x the
camelTransportFactory.setBus(bus);

// if you are lower CXF, you need to register the ConduitInitiatorManager and
DestinationFactoryManager like below
// register the conduit initiator
ConduitInitiatorManager cim = bus.getExtension(ConduitInitiatorManager.class);
cim.registerConduitInitiator(CamelTransportFactory.TRANSPORT_ID,
camelTransportFactory);
// register the destination factory
DestinationFactoryManager dfm = bus.getExtension(DestinationFactoryManager.class);
dfm.registerDestinationFactory(CamelTransportFactory.TRANSPORT_ID,
camelTransportFactory);
// set or bus as the default bus for cxf
BusFactory.setDefaultBus(bus);
```

76.10. SPRING で宛先とコンジットの設定

76.10.1. Namespace

Camel トランスポートエンドポイントを設定するために使用される要素は、名前空間 http://cxf.apache.org/transports/camel で定義されます。通常、接頭辞 camel の使用が参照されます。Camel トランスポート設定要素を使用するには、以下を示す行をエンドポイントの設定ファイルの beans 要素に追加する必要があります。さらに、設定要素の namespace を xsi:schemaLocation 属性に追加する必要があります。

設定名前空間の追加

```
<beans ...
      xmlns:camel="http://cxf.apache.org/transports/camel
      ...
      xsi:schemaLocation="...
                          http://cxf.apache.org/transports/camel
                          http://cxf.apache.org/transports/camel.xsd
                         ...>
```

76.10.2. destination 要素

camel:destination 要素およびその子を使用して Camel トランスポートサーバーのエンドポイントを設定します。camel:destination 要素は、エンドポイントに対応する WSDL port 要素を指定する単一の属性 名 を取ります。name 属性の値は portQName'.camel-destination' の形式を取ります。以下の例は、WSDL フラグメント < port binding="widgetSOAPBinding" name="widgetSOAPPort "> が http://widgets.widgetvendor.net の場合のエンドポイントの設定を追加するために使用される camel:destination 要素を示しています。

camel:destination 要素

```
...
  <camel:destination name="{http://widgets/widgetvendor.net}widgetSOAPPort.http-
destination>
    <camelContext id="context" xmlns="http://activemq.apache.org/camel/schema/spring">
        <route>
          <from uri="direct:EndpointC" />
          <to uri="direct:EndpointD" />
        </route>
    </camelContext>
  </camel:destination>

  <!-- new added feature since Camel 2.11.x
  <camel:destination name="{http://widgets/widgetvendor.net}widgetSOAPPort.camel-
destination" camelContextId="context" />

...
```

Spring の camel:destination 要素には、設定情報を指定する子要素が多数あります。これらは以下で説明します。

要素	説明
camel-spring:camelContext	camel 宛先で Camel コンテキストを指定できます。
camel:camelContextRef	Camel 宛先に注入する Camel コンテキスト ID

76.10.3. conduit 要素

camel:conduit 要素とその子を使用して Camel トランスポートクライアントを設定します。camel:conduit 要素は、エンドポイントに対応する WSDL port 要素を指定する単一の属性 名 を取ります。name 属性の値は portQName'.camel-conduit' の形式を取ります。たとえば、以下のコードは、WSDL フラグメント < port binding="widget SOAPBinding" name="widgetSOAPPort "> が http://widgets.widgetvendor.net の場合のエンドポイントの設定を追加するために使用される camel:conduit 要素を示しています。

http-conf:conduit Element

```
...
  <camelContext id="conduit_context" xmlns="http://activemq.apache.org/camel/schema/spring">
      <route>
          <from uri="direct:EndpointA" />
          <to uri="direct:EndpointB" />
      </route>
   </camelContext>

  <camel:conduit name="{http://widgets/widgetvendor.net}widgetSOAPPort.camel-conduit">
    <camel:camelContextRef>conduit_context</camel:camelContextRef>
  </camel:conduit>

  <!-- new added feature since Camel 2.11.x
```

```
  <camel:conduit name="{http://widgets/widgetvendor.net}widgetSOAPPort.camel-conduit"
camelContextId="conduit_context" />


  <camel:conduit name="*.camel-conduit">
  <!-- you can also using the wild card to specify the camel-conduit that you want to configure
-->
    ...
  </camel:conduit>
...
```

camel:conduit 要素には、設定情報を指定する子要素が多数あります。これらは以下で説明します。

要素

説明

camel-spring:camelContext

camel コンジットで camel コンテキストを指定できます。

camel:camelContextRef

camel コンダクトにインジェクトする Camel コンテキスト ID

76.11. BLUEPRINT で宛先とコンダミットの設定

Camel 2.11.x 以降、Camel Transport は Blueprint での設定をサポートします。

Blueprint を使用している場合は、名前空間 http://cxf.apache.org/transports/camel/blueprint を使用して、blow などのスキーマをインポートする必要があります。

Blueprint の設定名前空間の追加

```
<beans ...
```

```
xmlns:camel="http://cxf.apache.org/transports/camel/blueprint"
...
xsi:schemaLocation="...
                    http://cxf.apache.org/transports/camel/blueprint
                    http://cxf.apache.org/schmemas/blueprint/camel.xsd
                   ...>
```

Blueprint camel:conduit camel:destination には camelContextId 属性が 1 つだけあり、Camel 宛先で camel コンテキストを指定することはサポートされません。

```
<camel:conduit id="*.camel-conduit" camelContextId="camel1" />
<camel:destination id="*.camel-destination" camelContextId="camel1" />
```

76.12. CAMEL を CXF のロードバランサーとして使用する例

この例は、CXF で camel 負荷分散機能を使用する方法を示しています。CXF で設定ファイルを読み込み、"camel://direct:EndpointA" および "camel://direct:EndpointB" アドレスでエンドポイントを公開する必要があります。

76.13. CAMEL を CXF に割り当てる方法および例

Apache Camel を使用した CXF Web サービスの改善 JMS トランスポート

第77章 データフォーマットのコンポーネント

Camel バージョン 2.12 から利用可能

dataformat: コンポーネントは Data Format を Camel コンポーネントとして使用できます。

77.1. URI 形式

```
dataformat:name:(marshal|unmarshal)[?options]
```

name は、Data Format の名前です。その後、次に マーシャリング または アンマーシャリング 操作が必要です。このオプションは、使用中の データフォーマット の設定に使用されます。サポートされるオプションについては、Data Format のドキュメント を参照してください。

77.2. DATAFORMAT オプション

Data Format コンポーネントにはオプションがありません。

Data Format エンドポイントは、URI 構文を使用して設定します。

```
dataformat:name:operation
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

77.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
name	**必要な** データ形式の名前		文字列
operation	マーシャリングまたはアンマーシャリングのいずれかを使用するのに **必要な** 操作		文字列

77.2.2. クエリーパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

77.3. サンプル

たとえば、JAXB Data Format を使用するには、以下のように実行できます。

```
from("activemq:My.Queue").
  to("dataformat:jaxb:unmarshal?contextPath=com.acme.model").
  to("mqseries:Another.Queue");
```

XML DSL では、以下を行います。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="activemq:My.Queue"/>
    <to uri="dataformat:jaxb:unmarshal?contextPath=com.acme.model"/>
    <to uri="mqseries:Another.Queue"/>
  </route>
</camelContext>
```

第78章 データセットコンポーネント

Camel バージョン 1.3 で利用可能

分散および非同期処理のテストは非常に困難です。Mock、Test、DataSet エンドポイントは、Camel テストフレームワークにより優れた機能を提供し、エンタープライズ統合パターンと Camel の多くのコンポーネントと強力な Bean 統合 とのさまざまなコンポーネントを使用して、ユニットテストと統合テストを簡素化します。

DataSet コンポーネントは、システムの負荷およびウェイクテストを簡単に実行するメカニズムを提供します。これは、DataSet インスタンス をメッセージのソースとして作成できるようにし、データセットを受け取ることをアサートすることで機能します。

Camel は、データセットを送信するときに スループットロガー を使用します。

78.1. URI 形式

```
dataset:name[?options]
```

ここでの name は、レジストリーで DataSet インスタンス の検索に使用されます。

Camel には、独自の DataSet を実装するためのベースとして使用する org.apache.camel.component.dataset.DataSet クラスである org.apache.camel.component.dataset.DataSetSupport クラスのサポート実装が含まれています。Camel には、テストに使用できる実装（ org.apache.camel.component.dataset.SimpleDataSet、 org.apache.camel.component.dataset.ListDataSet および org.apache.camel.component.dataset.FileDataSet、すべて DataSetSupport を拡張する）も同梱されます。

78.2. オプション

Dataset コンポーネントにはオプションがありません。

Dataset エンドポイントは、URI 構文を使用して設定します。

```
dataset:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

78.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	レジストリーで検索するデータセットの名前		DataSet

78.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
dataSetIndex (common)	CamelDataSetIndex ヘッダーの動作を制御します。コンシューマーの場合： - off = the header will not be set - strict/lenient = the header will be set For Producers: - off = the header value will not be verified, and will not present = strict = the header value must be present and will be verified = lenient = the header value will be verified = lenient = the header value will be verified if it is present, and will be set if it is not strict = the header value will be verified = lenient = the header value will be	lenient	文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
initialDelay（コンシューマー）	メッセージの送信を開始するまでの待機時間（ミリ秒単位）。	1000	Long
minRate (consumer)	DataSet に最低でもこの数のメッセージが含まれるまで待ちます。	0	int
preloadSize (consumer)	ルートが初期化を完了する前に事前にロード（送信）されるメッセージの数を設定します。	0	Long
produceDelay (consumer)	遅延を指定でき、コンシューマーによってメッセージが送信されると遅延が生じます（処理速度をシミュレートするために）	3	Long

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
assertPeriod (producer)	モックエンドポイントが再アサートされる猶予期間を設定し、事前のアサーションが有効であることを確認します。これは、たとえば多数のメッセージが到着することをアサートするために使用されます。たとえば、expectMessageCount(int)が 5 に設定されている場合、5 つ以上のメッセージが到達するとアサーションが満たされます。5 つのメッセージが到着するようにするには、さらにメッセージが到達しないように、少し待機する必要があります。これは、このリンク setAssertPeriod(long)メソッドを使用できます。デフォルトでは、この期間は無効です。	0	Long
consumeDelay (producer)	遅延を指定できるようにします。これにより、メッセージがプロデューサーによって消費されると遅延が生じます（処理速度をシミュレートするために）	0	Long
expectedCount (producer)	このエンドポイントによって受信されるべきメッセージエクスチェンジの数を指定します。注意： 0 のメッセージを想定したい場合は、テストの開始時に 0 がマッチするので、アサート期間を設定して、メッセージが到達しないようにする必要があります。そのため、リンク setAssertPeriod(long)を使用します。別の方法として、NotifyBuilder を使用して、モックでリンク assertIsSatisfied（）メソッドを呼び出す前に、Camel がメッセージをルーティングするタイミングを把握することができます。これにより、テスト時間を短縮するために、固定されたアサート期間を使用できません。n 番目のメッセージがこのモックエンドポイントに到達することをアサートする場合は、詳細については、リンク setAssertPeriod(long)メソッドも参照してください。	-1	int
reportGroup (producer)	サイズのグループに基づいてスループットロギングを有効にするために使用される数。		int
resultMinimumWaitTime (producer)	リンクアサートIsSatisfied（）が満たされるまで、リンクアサートが待機する最小時間（ミリ秒単位）を設定します。	0	Long

Name	説明	デフォルト	Type
resultWaitTime (producer)	リンクアサート IsSatisfied（）が満たされるまでリンクアサートが待機する最大時間（ミリ秒単位）を設定します。	0	Long
retainFirst (producer)	受信されたエクスチェンジの最初の数だけを保持するように指定します。これは、このモックエンドポイントを受け取るすべての Exchange のコピーを保存せず、大量のデータでテストする際に使用され、メモリー消費を削減します。重要： この制限を使用すると、リンク getReceivedCounter（）は受信したエクスチェンジの実際の数を返します。たとえば、5000 Exchange を受け取ったため、最初の 10 Exchange のみを保持するように設定している場合は、リンク getReceivedCounter（）は引き続き 5000 を返しますが、リンク getExchanges（）とリンク getReceivedExchanges（）メソッドの最初の 10 エクスチェンジのみがあります。この方法を使用する場合は、想定される他の方法の一部はサポートされません。たとえば、リンク expectBodiesReceived(Object...)は受信される最初の本文数に期待を設定します。リンク setRetainFirst(int)メソッドおよびリンク setRetainLast(int)メソッドの両方を設定して、最初に受信した最初と最後に受信した両方を制限できます。	-1	int
retainLast (producer)	最後に受信したエクスチェンジの数のみを保持するように指定します。これは、このモックエンドポイントを受け取るすべての Exchange のコピーを保存せず、大量のデータでテストする際に使用され、メモリー消費を削減します。重要： この制限を使用すると、リンク getReceivedCounter（）は受信したエクスチェンジの実際の数を返します。たとえば、5000 Exchange を受け取ったときに最後の 20 Exchange のみを保持するよう設定されている場合、リンク getReceivedCounter（）は引き続き 5000 を返しますが、リンク getExchanges（）とリンク getReceivedExchanges（）メソッドの最後の 20 Exchange のみが存在します。この方法を使用する場合は、想定される他の方法の一部はサポートされません。たとえば、リンク expectBodiesReceived(Object...)は受信される最初の本文数に期待を設定します。リンク setRetainFirst(int)メソッドおよびリンク setRetainLast(int)メソッドの両方を設定して、最初に受信した最初と最後に受信した両方を制限できます。	-1	int

Name	説明	デフォルト	Type
sleepForEmptyTest (producer)	expectedMessageCount(int)がゼロで呼び出されると、このエンドポイントが実際に空であることを確認できるようスリープを指定できます。	0	Long
copyOnExchange (producer)	このモックエンドポイントで受信されたときに受信エクスチェンジのディープコピーを作成するかどうかを設定します。デフォルトは true です。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

URI にクエリーオプションを追加するには、 ?option=value&option=value&...

78.3. データセットの設定

Camel は DataSet インターフェースを実装する Bean のレジストリーでルックアップします。そのため、独自の DataSet を以下のように登録できます。

```
<bean id="myDataSet" class="com.mycompany.MyDataSet">
  <property name="size" value="100"/>
</bean>
```

78.4. 例

たとえば、一連のメッセージがキューに送信され、メッセージを失うことなくキューから消費されることをテストするには、以下を実行します。

```
// send the dataset to a queue
from("dataset:foo").to("activemq:SomeQueue");

// now lets test that the messages are consumed correctly
from("activemq:SomeQueue").to("dataset:foo");
```

上記は レジストリー を参照して、メッセージの作成に使用される foo DataSet インスタンスを検索します。

次に、以下のように SimpleDataSet を使用するなど、DataSet 実装を作成し、データセットの大きさや、メッセージの内容などを設定します。

78.5. DATASETSUPPORT（ABSTRACT クラス）

DataSetSupport の抽象クラスは、新しい DataSets のスタート地点であり、派生クラスに便利な機能を提供します。

78.5.1. DataSetSupport のプロパティー

プロパティー	Type	デフォルト	説明
defaultHeaders	**Map<String,Object>**	**null**	デフォルトのメッセージボディーを指定します。SimpleDataSet の場合、これは定数ペイロードです。ただし、メッセージごとにカスタムペイロードを作成する場合は、**DataSetSupport** の独自の派生を作成します。
outputTransformer	**org.apache.camel.Processor**	null	
size	**Long**	**10**	送信/消費するメッセージの数を指定します。
reportCount	**Long**	**-1**	レポートの進捗前に受信されるメッセージの数を指定します。大規模な負荷テストの進捗状況を表示する際に役立ちます。0 の場合、**サイズ** / 5 の場合、サイズが 0 の場合、**size** は **reportCount** 値に設定されます。

78.6. SIMPLEDATASET

SimpleDataSet は DataSetSupport を拡張し、デフォルトの本文を追加します。

78.6.1. SimpleDataSet の追加プロパティー

プロパティー	Type	デフォルト	説明
defaultBody	**オブジェクト**	**<hello>world!</hello>**	デフォルトのメッセージボディーを指定します。デフォルトでは、**SimpleDataSet** はエクスチェンジごとに同じ定数ペイロードを生成します。各エクスチェンジのペイロードをカスタマイズする場合は、Camel **Processor** を作成し、**outputTransformer** プロパティーを設定して **SimpleDataSet** がこれを使用するように設定します。

78.7. LISTDATASET

Camel 2.17 以降利用可能

List'DataSet' extends DataSetSupport は、デフォルトの本文のリストを追加します。

78.7.1. ListDataSet の追加プロパティー

プロパティー	Type	デフォルト	説明
defaultBodies	**List<Object>**	**emptyLinkedList<Object>**	デフォルトのメッセージボディーを指定します。デフォルトでは、**ListDataSet** は **CamelDataSetIndex** を使用して **defaultBodies** の一覧から定数ペイロードを選択します。ペイロードをカスタマイズする場合は、Camel **プロセッサー** を作成し、**outputTransformer** プロパティーを設定して **ListDataSet** がこれを使用するように設定します。
size	**Long**	defaultBodies リストのサイズ	送信/消費するメッセージの数を指定します。この値は、**defaultBodies** リストのサイズと異なる場合があります。値が **defaultBodies** リストのサイズよりも小さい場合、リスト要素の一部は使用されません。値が **defaultBodies** リストのサイズよりも大きい場合、CamelDataSetIndex のモジュラーと **defaultBodies** リストのサイズ（例：**CamelDataSetIndex % defaultBodies.size（）**）を使用してエクスチェンジのペイロードが選択されます。

78.8. FILEDATASET

Camel 2.17 以降利用可能

FileDataSet は ListDataSet を拡張し、ファイルから本文をロードするサポートを追加します。

78.8.1. FileDataSet の追加プロパティー

プロパティー	Type	デフォルト	説明
sourceFile	**ファイル**	null	ペイロードのソースファイルを指定します。
delimiter	**文字列**	\z	**java.util.Scanner** によって使用される区切り文字パターンを指定し、ファイルを複数のペイロードに分割します。

第79章 DIGITALOCEAN コンポーネント

Camel バージョン 2.19 から利用可能

DigitalOcean コンポーネントを使用すると、[digitalocean-api-java](https://www.digitalocean.com/community/projects/api-client-in-java)をカプセル化することで、Camel で DigitalOcean クラウド内の Droplets とリソースを管理できます。DigitalOcean コントロールパネルに精通している機能はすべて、この Camel コンポーネントで利用できます。

79.1. 前提条件

有効な DigitalOcean アカウントと有効な OAuth トークンが必要です。アカウントの DigitalOcean コントロールパネルの [Apps & API](https://cloud.digitalocean.com/settings/applications)セクションに移動すると、OAuth トークンを生成できます。

79.2. URI 形式

DigitalOcean コンポーネント は以下の URI 形式を使用します。

```
digitalocean://endpoint?[options]
```

ここで、endpoint は DigitalOcean リソースタイプです。

例： ドロップダウンを一覧表示するには、以下を実行します。

```
digitalocean://droplets?operation=list&oAuthToken=XXXXXX&page=1&perPage=10
```

DigitalOcean コンポーネントはプロデューサーエンドポイントのみをサポートするため、ルートの開始時にこのコンポーネントを使用してチャネル内のメッセージをリッスンできません。

79.3. オプション

DigitalOcean コンポーネントにはオプションがありません。

DigitalOcean エンドポイントは、URI 構文を使用して設定します。

digitalocean:operation

以下の path パラメーターおよびクエリーパラメーターを使用します。

79.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
operation	指定のリソースに対して実行する操作。		DigitalOceanOperations

79.3.2. クエリーパラメーター（10 パラメーター）：

Name	説明	デフォルト	Type
ページ（プロデューサー）	ページネーションに使用します。ページ番号を強制的に実行します。	1	整数
perPage (producer)	ページネーションに使用します。リクエストごとに項目数を設定します。ページごとの結果の最大数は 200 です。	25	整数
resource (producer)	操作を実行する DigitalOcean リソースタイプが **必要です**。		DigitalOceanResources
digitalOceanClient (advanced)	既存の設定済みの DigitalOceanClient をクライアントとして使用します。		DigitalOceanClient
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
httpProxyHost (proxy)	必要に応じてプロキシーホストを設定します。		文字列
httpProxyPassword (proxy)	必要に応じてプロキシーパスワードを設定します。		文字列
httpProxyPort (proxy)	必要に応じてプロキシーポートを設定します。		整数
httpProxyUser (proxy)	必要に応じてプロキシーホストを設定します。		文字列

Name	説明	デフォルト	Type
oAuthToken (security)	DigitalOcean OAuth トークン		文字列

オペレーション URI オプションまたは CamelDigitalOceanOperation メッセージヘッダーを使用して、各エンドポイントに 操作 値を提供する必要があります。

操作 値はすべて DigitalOceanOperations 列挙で定義されます。

コンポーネントが使用する ヘッダー 名はすべて DigitalOceanHeaders 列挙で定義されます。

79.4. メッセージボディーの結果

返されるすべてのメッセージ本文は、digitalocean-api-java ライブラリーによって提供されるオブジェクトを使用します。

79.5. API レート制限

camel-digitalocean コンポーネントでカプセル化された DigitalOcean REST API は、API Rate Limiting の対象となります。メソッドごとの制限の詳細は、[API Rate Limits documentation] (https://developers.digitalocean.com/documentation/v2/#rate-limit)を参照してください。

79.6. アカウントエンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | 00:00:0- | | get | get account info | | com.myjeeva.digitalocean.pojo.Account |

79.7. BLOCKSTORAGES エンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | 00:00:0- | | list | list all of the Block Storage volumes available on your account | | List<com.myjeeva.digitalocean.pojo.Volume > | | get | show information about a Block Storage volume| CamelDigitalOceanId Integer*| com.myjeeva.digitalocean.pojo.Volume | | get | show information by name| CamelDigitalOceanName* String *
'CamelDigitalOceanRegion'* string*| com.myjeeva.digitalocean.pojo.Volume | | listSnapshots | volume から作成されたスナップショット*

| CamelDigitalOceanId Integer*| List<com.myjeeva.digitalocean. POJO.Snapshot > | | create | create a new volume | CamelDigitalOceanVolumeSizeGigabytes* Integer*
'CamelDigitalOceanName'* String*
'CamelDigitalOceanDescription'* String*
' camelDigitalOceanRegion'* String*| com.myjeeva.digitalocean.pojo.Volume | | delete | delete a Block Storage volume, すべてのデータを破棄し、アカウントからデータを削除します| CamelDigitalOceanId* Integer*| com.myjeeva.digitalocean.pojo.Delete| | delete | delete a Block Storage volume by name| CamelDigitalOceanName* String
'CamelDigitalOceanRegion' String| com.myjeeva.digitalocean.pojo.Delete | attach | attach a Block Storage volume to a Droplet| CamelDigitalOceanId Integer
'CamelDigitalOceanRegion' String | com.myjeeva.digitalocean.pojo.Delete | attach | attach a Block Storage volume to a Droplet| CamelDigitalOcean DigitalOceanDropletId'* Integer*
'CamelDigitalOceanDropletRegion'* String*| com.myjeeva.digitalocean.pojo.Action | attach | attach a Block Storage volume to a Droplet by name| Camel DigitalOceanName* String*
'CamelDigitalOceanDropletId'* Integer*
'CamelDigitalOceanDropletRegion'* String*| com.myjeeva.digitalocean.pojo.Action | detach | Block Storage ボリュームを Droplet| CamelDigitalOceanId* Integer *
'CamelDigitalOceanDropletId'* Integer*
'CamelDigitalOceanDropletRegion'* String*| com.myjeeva.digital ocean.pojo.Action | attach | detach a Block Storage volume from a Droplet by name| CamelDigitalOceanName* String*
'CamelDigitalOceanDropletId'* Integer*
'CamelDigitalOceanDropletRegion ' string | com.myjeeva.digitalocean.pojo.Action | resize | resize a Block Storage volume | CamelDigitalOceanVolumeSizeGigabytes* Integer*
'CamelDigitalOceanRegion'* String*| com.myjeeva.digitalocean.pojo.Action | | listActions | retrieve all actions that been executed on a volume | CamelDigitalOceanId* Integer*| List<com.myjeeva.digitalocean.pojo. action> |*

79.8. DROPLETS エンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | 00:00:0- | | list | list all Droplets in your account | | List<com.myjeeva.digitalocean.pojo.Droplet > | | get | show an individual droplet | CamelDigital OceanId Integer*| com.myjeeva.digitalocean.pojo.Droplet | | create | create a new Droplet | CamelDigitalOceanName* String *
'CamelDigitalOceanDropletImage'* String *
'CamelDigitalOceanRegion'* String *
'CamelDigitalOceanDropletSize'* String *
'CamelDigitalOceanDropletSSHKeys'* List\<String\ >
'CamelDigitalOceanDropletEnableBackups '* boolean *
'CamelDigitalOceanDropletEnableIpv6'* Boolean *
'CamelDigitalOceanDropletEnablePrivateNetworking'* Boolean *
'CamelDigitalOceanDropletUser Data'* String *
'CamelDigitalOceanDropletVolumes'* List\ <String\ >
'CamelDigitalOceanDropletTags'* List\<String\> *| com.myjeeva.digitalocean.pojo. Droplet | | create | create multiple Droplets | CamelDigitalOceanNames* List\<String\ > *
'CamelDigitalOceanDropletImage' String
'CamelDigitalOceanRegion'* String *
'CamelDigitalOceanRegion'* String *
'Camel DigitalOceanDropletSize'* String *
'CamelDigitalOceanDropletSSHKeys'* List\<String\ >
'CamelDigitalOceanDropletEnableBackups'* Boolean *
'CamelDigitalOce anDropletEnableIpv6'* Boolean *
'CamelDigitalOceanDropletEnablePrivateNetworking'* Boolean *
'CamelDigitalOceanDropletUserData'* String *
'CamelDigitalOceanDroplet volumes'* List\<String\ >
'CamelDigitalOceanDropletTags'* List\<String\> *| com.myjeeva.digitalocean.pojo.Droplet | | delete | delete a Droplet, | CamelDigitalOceanId* Integer*| com.myjeeva.digitalocean.pojo.Delete | | enableBackups | enable backups on an existing Droplet | CamelDigitalOceanId* Integer*| com.my jeeva.digitalocean.pojo.Action | | disableBackups | disable backups on an existing Droplet | CamelDigitalOceanId* Integer*| com.myjeeva.digitalocean.pojo.Action | | enableI pv6 | enable IPv6 networking on an existing Droplet | CamelDigitalOceanId* Integer*| com.myjeeva.digitalocean.pojo.Action | | enablePrivateNetworking | enable private networking on an existing Droplet | CamelDigital OceanId* Integer*| com.myjeeva.digitalocean.pojo.Action | | reboot | reboot a Droplet |*

CamelDigitalOceanId Integer| *com.myjeeva.digitalocean.poj O.Action | | powerCycle | power cycle a Droplet | CamelDigitalOceanId* Integer| *com.myjeeva.digitalocean.pojo.Action | | shutdown | shutdown a Droplet | CamelDigitalOce anId* Integer| *com.myjeeva.digitalocean.pojo.Action | | powerOff | power off a Droplet | CamelDigitalOceanId* Integer| *com.myjeeva.digitalocean.poj O.Action | | powerOn | power on a Droplet | CamelDigitalOceanId* Integer| *com.myjeeva.digitalocean.pojo.Action | | restore | shutdown a Droplet | CamelDigitalOce anId* Integer *
'CamelDigitalOceanImageId'* Integer| *com.myjeeva.digitalocean.pojo.Action | | passwordReset | reset the password for a Droplet | CamelDigitalOceanId* Integer| *com.myjeeva.digitalocean.pojo.Action | | resize | resize a Droplet | CamelDigitalOceanId* Integer *
'CamelDigitalOceanDropletSize'* String| *com .myjeeva.digitalocean.pojo.Action | | rebuild | rebuild a Droplet | CamelDigitalOceanId* Integer *
'CamelDigitalOceanImageId'* Integer| *com.myjeeva.digital ocean.pojo.Action | | rename | rename a Droplet | CamelDigitalOceanId* Integer *
'CamelDigitalOceanName'* String| *com.myjeeva.digitalocean.pojo.Action | | changeKernel | change the kernel of a Droplet | CamelDigitalOceanId* Integer *
'CamelDigitalOceanKernelId'* Integer| *com.myjeeva.digitalocean.pojo.Action | | takeSnapshot | snapshot a Droplet | CamelDigitalOceanId* Integer *
'CamelDigitalOceanName'** String| *com.myjeeva.digitalocean.pojo.Action | | tag | tag a Droplet | CamelDigitalOceanId* Integer *
'CamelDigitalOceanName'* String| *com.myjeeva.digitalocean.pojo.Response | | untag | untag a Droplet | CamelDigitalO ceanId* Integer *
'CamelDigitalOceanName'* String| *com.myjeeva.digitalocean.pojo.Response | | listKernels | retrieve a listKernels | Droplet | CamelDigitalO ceanId* Integer *| List<com.myjeeva.digitalocean.pojo.Kernel > | | listSnapshots | Droplet | CamelDigitalOceanId* Integer *| List<com. myjeeva.digitalocean.pojo.Snapshot > | | listBackups | retrieve any backup associated with a Droplet | CamelDigitalOceanId* Integer *| List<com.myjeeva.digitalocean.pojo.Backup> | | listActions | retrieve all actions that are executed on a Droplet | CamelDigitalOceanId* Integer *| List<com.myjeeva.digitalocean.pojo.Action > | | listNeighbors | retrieve a listNeighbors | retrieve a list of droplets that 同じ物理サーバー | CamelDigitalOceanId* Integer *| List<com.myjeeva.digitalocean.pojo.Droplet > | | listAllNeighbors | | listAllNeighbors | | list<com.myjeeva.digitalocean.pojo.Droplet> |*

79.9. IMAGES エンドポイント

*| operation | Description | Headers | Result | | ------ | ---- | ------- | 00:00:0- | | list | list images available on your account | CamelDigitalOceanType** DigitalOceanImageTypes *| List<com.myjeeva.digitalocean.pojo.Image & gt; | | ownList | retrieve the private images of a user | | List<com.myjeeva.digitalocean.pojo.Image > | | listActions | retrieve all actions that have been executed on a Image | CamelDigitalOceanId* Integer *| List <com.myjeeva.digitalocean.pojo.Action > | | get | retrieve information about an image(public or private)by id| CamelDigitalOceanId* Integer| *com.myjeeva.digitalocean.pojo.Image | | get | retrieve information about an public image by slug| CamelDigitalOceanDropletImage* String| *com.myjeeva.digitalocean.pojo.Image | | update | update an image| CamelDigitalOcean id* Integer *
'CamelDigitalOceanName'* String| *com.myjeeva.digitalocean.pojo.Image | | delete | delete an image| CamelDigitalOceanId* Integer *| com.my jeeva.digitalocean.pojo.Delete | | transfer | transfer an image to another region| CamelDigitalOceanId* Integer *
'CamelDigitalOceanRegion'* String| *com.myjeeva.digitalocean . POJO.Action | | convert | convert an image, たとえば、スナップショットへのバックアップ| CamelDigitalOceanId* Integer *| com.myjeeva.digitalocean.pojo.Action |*

79.10. SNAPSHOTS エンドポイント

*| operation | Description | Headers | Result | | ------ | ---- | ------- | 00:00:0- | | list | list all of the snapshots available on your account | CamelDigitalOceanType** DigitalOceanSnapshotTypes *|*

List<com.myjeeva.digitalocean.poj O.Snapshot& gt; | | get | retrieve | retrieve information about a snapshot| CamelDigitalOceanId Integer*| com.myjeeva.digitalocean.pojo.Snapshot | | delete | delete an snapshot| CamelDigitalOceanId* Integer *| com.myjeeva.digitalocean.pojo.Delete |*

79.11. KEYS エンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | 00:00:0- | | list | list all of the keys in your account | | List<com.myjeeva.digitalocean.pojo.Key > | | get | retrieve information about a key by id| CamelDigit alOceanId Integer*| com.myjeeva.digitalocean.pojo.Key | | get | retrieve information about a key by fingerprint| CamelDigitalOceanKeyFingerprint* String*| com.myjeeva.digitaloceanan . POJO.Key | | update | update a key by id| CamelDigitalOceanId* Integer *
'CamelDigitalOceanName'* String*| com.myjeeva.digitalocean.pojo.Key | | Update | update a key by fingerprint| CamelDigitalOceanKeyFingerprint* String *
'CamelDigitalOceanName'* String*| com.myjeeva.digitalocean.pojo.Key | | delete | delete A key by id| CamelDigitalOceanId* Integer *| com.myjeeva.digitalocean.pojo.Delete | | delete | delete a key by fingerprint| CamelDigitalOceanKeyFingerprint* String *| com.myjeeva.digitalocean.pojo.Delete |*

79.12. リージョンエンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | 00:00:0- | | list | list all of the regions that are available | | List<com.myjeeva.digitalocean.pojo.Region> |

79.13. SIZE のエンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | 00:00:0- | | list | list all of the size that are available | | List<com.myjeeva.digitalocean.pojo.Size> |

79.14. FLOATING IP エンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | 00:00:0- | | list | list all of the floating IPs available on your account | | List<com.myjeeva.digitalocean.pojo.FloatingIP > | | create | create a new Floating IP assigned to a Droplet | CamelDigitalOceanId Integer *| List<com.myjeeva.digitalocean.pojo.FloatingIP > | | create | create a new Floating IP assigned to a Region | CamelDigitalOceanRegion* String *| List<com. myjeeva.digitalocean.pojo.FloatingIP > | | get | retrieve information about a floating IP| CamelDigitalOceanFloatingIPAddress* String*| com.myjeeva.digitalocean.pojo.Key | | delete | Floating IP を削除して、アカウントからその IP を削除します。CamelDigitalOceanFloatingIPAddress* String*| com.myjeeva.digitalocean.pojo.Delete | | assign | assign a floating IP to a Droplet| CamelDigitalOce anFloatingIPAddress* String *
'CamelDigitalOceanDropletId'* Integer*| com.myjeeva.digitalocean.pojo.Action | | unassign | unassign a Floating IP | CamelDigitalOceanFloatingIPAddress string | com.myjeeva.digitalocean.pojo.Action | | listActions | retrieve all actions that been executed on a floating IP | CamelDigitalOceanFloatingIPAddress* String *| List<com.myjeeva.digital ocean.pojo.Action> |*

79.15. TAGS エンドポイント

| operation | Description | Headers | Result | | ------ | ---- | ------- | 00:00:0- | | list | list all of your tags | | List<com.myjeeva.digitalocean.pojo.Tag > | | create | create a Tag | CamelDigitalOceanName string | com.myjeeva.digitalocean.pojo.Tag | | get | retrieve an individual tag | CamelDigitalOceanName String | com.myjeeva.digitalocean.pojo.Tag | | delete | delete タグ | CamelDigitalOceanName String | com.myjeeva.digitalocean.pojo.Delete | | update | update a tag | CamelDigitalOceanName String
'CamelDigitalOcean newname' String| com.myjeeva.digitalocean.pojo.Tag |

79.16. 例

アカウント情報の取得

```
from("direct:getAccountInfo")
    .setHeader(DigitalOceanConstants.OPERATION, constant(DigitalOceanOperations.get))
    .to("digitalocean:account?oAuthToken=XXXXXX")
```

ドロップレットの作成

```
from("direct:createDroplet")
    .setHeader(DigitalOceanConstants.OPERATION, constant("create"))
    .setHeader(DigitalOceanHeaders.NAME, constant("myDroplet"))
    .setHeader(DigitalOceanHeaders.REGION, constant("fra1"))
    .setHeader(DigitalOceanHeaders.DROPLET_IMAGE, constant("ubuntu-14-04-x64"))
    .setHeader(DigitalOceanHeaders.DROPLET_SIZE, constant("512mb"))
    .to("digitalocean:droplet?oAuthToken=XXXXXX")
```

ドロップレットの一覧を表示します。

```
from("direct:getDroplets")
    .setHeader(DigitalOceanConstants.OPERATION, constant("list"))
    .to("digitalocean:droplets?oAuthToken=XXXXXX")
```

Droplet に関する情報を取得します(dropletId = 34772987)。

```
from("direct:getDroplet")
    .setHeader(DigitalOceanConstants.OPERATION, constant("get"))
    .setHeader(DigitalOceanConstants.ID, 34772987)
    .to("digitalocean:droplet?oAuthToken=XXXXXX")
```

Droplet のシャットダウン情報(dropletId = 34772987)

```
from("direct:shutdown")
    .setHeader(DigitalOceanConstants.ID, 34772987)
    .to("digitalocean:droplet?operation=shutdown&oAuthToken=XXXXXX")
```

第80章 DIRECT コンポーネント

Camel バージョン 1.0 で利用可能

direct: コンポーネントは、プロデューサーがメッセージエクスチェンジを送信するときに、コンシューマーの直接同期呼び出しを提供します。
このエンドポイントは、同じ Camel コンテキストの既存のルートを接続するために使用できます。

ヒント

非同期 The SEDA コンポーネントは、プロデューサーがメッセージエクスチェンジを送信するときにコンシューマーの非同期呼び出しを提供します。

ヒント

他の Camel コンテキストへ の接続。VM コンポーネントは、同じ JVM で実行される限り Camel コンテキスト間の接続を提供します。

80.1. URI 形式

```
direct:someName[?options]
```

someName には、エンドポイントを一意に識別する文字列を使用できます。

80.2. オプション

Direct コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
ブロック（プロデューサー）	アクティブなコンシューマーのないダイレクトエンドポイントにメッセージを送信する場合、プロデューサーに対して、コンシューマーがアクティブになるのをブロックし、待機することができます。	true	boolean
タイムアウト（プロデューサー）	block が有効な場合に使用するタイムアウト値。	30000	Long

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Direct エンドポイントは、URI 構文を使用して設定されます。

```
direct:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

80.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	直接エンドポイントに **必要な** 名前		文字列

80.2.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
ブロック （プロデューサー）	アクティブなコンシューマーのないダイレクトエンドポイントにメッセージを送信する場合、プロデューサーに対して、コンシューマーがアクティブになるのをブロックし、待機することができます。	true	boolean
failIfNoConsumers (producer)	アクティブなコンシューマーのない DIRECT エンドポイントに送信するときに、プロデューサーが例外を発生させて失敗するかどうか。	false	boolean
タイムアウト （プロデューサー）	block が有効な場合に使用するタイムアウト値。	30000	Long
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

80.3. サンプル

以下のルートでは direct コンポーネントを使用して、2 つのルートをリンクします。

```
from("activemq:queue:order.in")
    .to("bean:orderServer?method=validate")
    .to("direct:processOrder");

from("direct:processOrder")
    .to("bean:orderService?method=process")
    .to("activemq:queue:order.out");
```

Spring DSL を使用した例を以下に示します。

```
<route>
 <from uri="activemq:queue:order.in"/>
 <to uri="bean:orderService?method=validate"/>
 <to uri="direct:processOrder"/>
</route>

<route>
 <from uri="direct:processOrder"/>
```

```
 <to uri="bean:orderService?method=process"/>
 <to uri="activemq:queue:order.out"/>
</route>
```

SEDA コンポーネントの例も併せて参照してください。

80.4. 関連項目

- *SEDA*
- *VM*

第81章 DIRECT VM コンポーネント

Camel バージョン 2.10 で利用可能

direct-vm: コンポーネントは、プロデューサーがメッセージエクスチェンジを送信する際に、JVM 内のコンシューマーの直接同期呼び出しを提供します。
このエンドポイントは、同じ Camel コンテキストの既存のルートや、同じ JVM にある他の Camel コンテキストから接続するために使用できます。

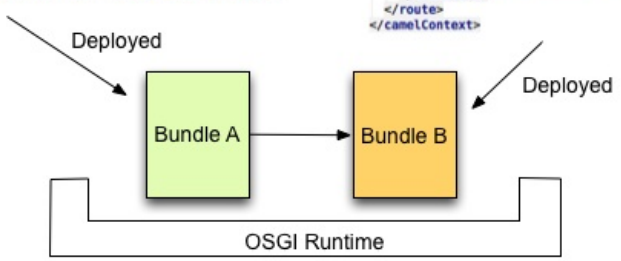
このコンポーネントは、 Direct -VM が CamelContext インスタンス全体の通信をサポートするという点で Direct コンポーネントとは異なり、このメカニズムを使用して Web アプリケーション全体で通信できます（camel-core.jar はシステム/起動クラスパス上にある場合）。

実行時に、既存のコンシューマーを停止して新しいコンシューマーを開始することで、新しいコンシューマーをスワップできます。
ただし、任意の時点では、指定のエンドポイントに対して最大でアクティブなコンシューマーを 1 つだけ指定できます。

このコンポーネントでは、後に確認できるように、異なる OSGI Bundles にデプロイされたルートを接続することもできます。異なるバンドルで動作している場合でも、Camel ルートは同じスレッドを使用します。
これは、トランザクション - Tx を使用してアプリケーションの開発を自動的に行います。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="timer:foo?period=5000"/>
    <log message=">> Here is a message from direct-vm (bundle1)"/>
    <log message=">> Using ${threadName} of bundle 1"/>
    <to uri="direct-vm:hello"/>
    <log message=">> Response received from bundle 2 : ${body}"/>
  </route>
</camelContext>
```

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct-vm:hello"/>
    <log message=">> Prepare response from bundle 2"/>
    <log message=">> Using ${threadName} of bundle 2"/>
    <transform>
        <method ref="helloBean" method="hello"/>
    </transform>
  </route>
</camelContext>
```




```
INFO  | 14 - timer://foo | route7  >> Here is a message from direct-vm (bundle1)
INFO  | 14 - timer://foo | route7  >> Using Camel (89-camel-23) thread #14 - timer://foo of bundle 1
INFO  | 14 - timer://foo | route8  >> Prepare response from bundle 2
INFO  | 14 - timer://foo | route8  >> Using Camel (89-camel-23) thread #14 - timer://foo of bundle 2
INFO  | 14 - timer://foo | route7  >> Response received from bundle 2 : Hi from Camel direct-vm at 2012-06-21 15:21:22
```

81.1. URI 形式

```
direct-vm:someName
```

someName には、エンドポイントを一意に識別する文字列を使用できます。

81.2. オプション

Direct VM コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
ブロック（プロデューサー）	アクティブなコンシューマーのないダイレクトエンドポイントにメッセージを送信する場合、プロデューサーに対して、コンシューマーがアクティブになるのをブロックし、待機することができます。	true	boolean
タイムアウト（プロデューサー）	block が有効な場合に使用するタイムアウト値。	30000	Long
headerFilterStrategy (advanced)	プロデューサーエンドポイント（要求と応答の両方）にのみ適用される HeaderFilterStrategy を設定します。デフォルト値： none		HeaderFilterStrategy
propagateProperties (advanced)	プロデューサー側からコンシューマー側にプロパティーを伝播するかどうか、またはコンシューマー側にプロパティーを伝播するかどうか。デフォルト値: true	true	boolean
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Direct VM エンドポイントは、URI 構文を使用して設定します。

```
direct-vm:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

81.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	direct-vm エンドポイントに **必要な** 名前		文字列

81.2.2. クエリーパラメーター（9 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
ブロック （プロデューサー）	アクティブなコンシューマーのないダイレクトエンドポイントにメッセージを送信する場合、プロデューサーに対して、コンシューマーがアクティブになるのをブロックし、待機することができます。	true	boolean
failIfNoConsumers (producer)	アクティブなコンシューマーのない Direct-VM エンドポイントに送信するときに、プロデューサーが例外を発生させて失敗するかどうか。	false	boolean
タイムアウト （プロデューサー）	block が有効な場合に使用するタイムアウト値。	30000	Long
headerFilterStrategy (producer)	プロデューサーエンドポイント（要求と応答の両方）にのみ適用される HeaderFilterStrategy を設定します。デフォルト値： none		HeaderFilterStrategy
propagateProperties (advanced)	プロデューサー側からコンシューマー側にプロパティーを伝播するかどうか、またはコンシューマー側にプロパティーを伝播するかどうか。デフォルト値: true	true	boolean

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

81.3. サンプル

以下のルートでは direct コンポーネントを使用して、2 つのルートをリンクします。

```
from("activemq:queue:order.in")
    .to("bean:orderServer?method=validate")
    .to("direct-vm:processOrder");
```

別の OSGi バンドルなど、別の CamelContext に

```
from("direct-vm:processOrder")
    .to("bean:orderService?method=process")
    .to("activemq:queue:order.out");
```

Spring DSL を使用した例を以下に示します。

```
<route>
 <from uri="activemq:queue:order.in"/>
 <to uri="bean:orderService?method=validate"/>
 <to uri="direct-vm:processOrder"/>
</route>

<route>
 <from uri="direct-vm:processOrder"/>
 <to uri="bean:orderService?method=process"/>
 <to uri="activemq:queue:order.out"/>
</route>
```

81.4. 関連項目

- *Direct*
- *SEDA*

- *VM*

第82章 DISRUPTOR コンポーネント

Camel バージョン 2.12 から利用可能

中断者： コンポーネントは標準の SEDA コンポーネントよりもはるかに非同期 SEDA の動作を提供しますが、標準の SEDA で使用される ブロッキングキュー の代わりに Disruptor を使用します。または、以下を実行します。

Disruptor-vm: エンドポイントは、このコンポーネントによってサポートされ、標準の 仮想マシン の代替を提供します。SEDA コンポーネントと同様に、中断 者 のバッファーです。エンドポイントは 単一 の CamelContext 内でのみ表示され、永続性やリカバリーにはサポートは提供されません。break or-vm: エンドポイントのバッファーは CamelContext インスタンス全体の通信もサポートします。そのため、このメカニズムを使用して Web アプリケーション全体で通信できます（ camel-disruptor.jar は system/boot クラスパスにあります）。

SEDA または VM コンポーネントで Disruptor コンポーネントを使用することを選択する主な利点は、プロデューサーとマルチキャストされたコンシューマー間またはマルチキャストされたコンシューマー間または同時コンシューマーの間で競合が高いユースケースのパフォーマンスです。このような場合、スループットが大幅に増大し、レイテンシーが削減されました。競合のないシナリオでのパフォーマンスは、SEDA および仮想マシンのコンポーネントと類似しています。

Disruptor は、可能な限り SEDA および仮想マシンコンポーネントの動作とオプションを模倣する意図と共に実装されます。それらの主な相違点は以下のとおりです。

- 使用されるバッファーは常にサイズでバインドされます（デフォルトは 1024 エクスチェンジ）。
- バッファーは常に書き込みされるため、Diruptor のデフォルト動作は例外をスローする代わりにバッファーが満杯になっている間ブロックされます。このデフォルトの動作はコンポーネントに設定できます（オプションを参照）。
- Disruptor enpoints は BrowsableEndpoint インターフェースを実装しません。そのため、現在 Disruptor のエクスチェンジを取得できず、エクスチェンジの量のみを取得することはできません。
- Disruptor では、コンシューマー（マルチキャストその他）を静的に設定する必要があります。コンシューマーの追加または削除には、Diruptor のすべての保留中のエクスチェンジを完全にフラッシュする必要があります。

- 再設定の結果として、Disruptor を介して送信されたデータは直接処理され、コンシューマーが 1 つ以上ある場合は 'gone' になり、結合後に公開される新しいエクスチェンジのみを取得するようになりました。

- pollTimeout オプションは Disruptor コンポーネントではサポートされません。

- プロデューサーが完全な Disruptor でブロックされると、スレッド割り込みに応答しません。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-disruptor</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

82.1. URI 形式

```
disruptor:someName[?options]
```

または

```
disruptor-vm:someName[?options]
```

someName には、現在の CamelContext 内のエンドポイントを一意に識別する文字列を指定できます（または
disruptor-vm:の場合のコンテキスト全体で）。
以下の形式で URI にクエリーオプションを追加できます。

```
?option=value&option=value&...
```

82.2. オプション

以下のオプションは、中断者： および disruptor-vm: コンポーネントの両方に対して有効です。

Disruptor コンポーネントは、以下に示す 8 個のオプションをサポートします。

Name	説明	デフォルト	Type
defaultConcurrent Consumers (consumer)	同時コンシューマーのデフォルト数を設定します。	1	int
DefaultMultiple Consumers （コンシューマー）	複数のコンシューマーのデフォルト値の設定	false	boolean
defaultProducerType (producer)	DisruptorProducerType のデフォルト値を設定するには、デフォルト値は Multi です。	マルチ	DisruptorProducerType
defaultWaitStrategy (consumer)	DisruptorWaitStrategy のデフォルト値を設定するには、デフォルト値は Blocking です。	Blocking	DisruptorWaitStrategy
defaultBlockWhenFull (producer)	完全な場合に block のデフォルト値を設定するには、デフォルト値は true です。	true	boolean
queueSize (common)	リングバッファーサイズの設定を **非推奨** 化		int
bufferSize (common)	リングバッファーサイズの設定	1024	int
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Disruptor エンドポイントは URI 構文を使用して設定されます。

```
disruptor:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

82.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	**必要な** キューの名前		文字列

82.2.2. クエリーパラメーター（12 パラメーター）：

Name	説明	デフォルト	Type
size (common)	Disruptors リングバッファーの最大容量が、実質的に 2 の最も近い累乗に増えます。注記： このオプションを使用する場合は、キュー名で最初のエンドポイントが作成され、サイズが決定されます。すべてのエンドポイントが同じサイズを使用するようにするには、それらすべてに size オプション、または作成される最初のエンドポイントを設定します。	1024	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	エクスチェンジを処理する同時スレッドの数。	1	int
multipleConsumers (consumer)	複数のコンシューマーを許可するかどうかを指定します。有効にすると、Publish-Subscribe メッセージングに Disruptor を使用できます。つまり、キューにメッセージを送信し、各コンシューマーがメッセージのコピーを受け取ることができます。このオプションを有効にすると、すべてのコンシューマーエンドポイントでこのオプションを指定する必要があります。	false	boolean
waitStrategy (consumer)	新しいエクスチェンジが公開されるまで待機するためにコンシューマースレッドによって使用されるストラテジーを定義します。許可されるオプションは、Blocking、Sleeping、BusySpin、および Yielding です。	Blocking	DisruptorWaitStrategy

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
blockWhenFull (producer)	メッセージを完全な Disruptor に送信するスレッドが、リングバッファーの容量が使い切られるまでブロックするかどうか。デフォルトでは、呼び出しスレッドはブロックされ、メッセージが受け入れられるまで待機します。このオプションを無効にすると、キューが満杯であることを示す例外が発生します。	false	boolean
producerType (producer)	Disruptor で許可されるプロデューサーを定義します。指定できるオプションは、複数のプロデューサーと 1 つのプロデューサーが 1 つのスレッドで（または同期されている）アクティブである場合に限り、特定の最適化を可能にする multi です。	マルチ	DisruptorProducerType
タイムアウト （プロデューサー）	プロデューサーが非同期タスクの完了の待機を停止するまでのタイムアウト（ミリ秒単位）。タイムアウトを無効にするには、0 または負の値を使用します。	30000	Long
waitForTaskToComplete (producer)	非同期タスクが完了するまで待機すべきかどうかを指定するオプション。続行します。Always、Never、または IfReplyExpected の 3 つのオプションがサポートされます。最初の 2 つの値は self-explany です。IfReplyExpected の最後の値は、メッセージが Request Reply based の場合にのみ待機します。	IfReplyExpected	WaitForTaskToComplete
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

82.3. 待機ストラテジー

wait ストラテジーは、現在次のエクスチェンジの公開を待機しているコンシューマースレッドによって実行される待機タイプに影響します。以下のストラテジーを選択できます。

Name	説明	アドバイス
Blocking	バリアを待機中のコンシューマーにロック変数と条件変数を使用するブロッキングストラテジー。	このストラテジーは、スループットと低レイテンシーが CPU リソースほど重要ではない場合に使用できます。
スリープ状態	最初にスピンングストラテジーを使用してから、Thread.yield（）を使用し、最終的に OS と JVM がバリアを待機している間に OS と JVM が許容するスペース設定が可能です。	このストラテジーは、パフォーマンスと CPU リソース間の適切な危険性があります。レイテンシーの急増は、不規則な期間の後に発生する可能性があります。
BusySpin	バリアを待機中のコンシューマーにビジースピンループを使用するビジー Spin ストラテジー。	このストラテジーは CPU リソースを使用してシステムコールを使用し、レイテンシージッターが発生する可能性があります。スレッドを特定の CPU コアにバインドできる場合は、最適です。
生成	最初のスピンの後にバリアを待機しているコンシューマーに Thread.yield（）を使用するストラテジーを生成します。	このストラテジーは、非常にレイテンシーが急増することなく、パフォーマンスと CPU リソース間の耐性があります。

82.4. 要求応答の使用

Disruptor コンポーネントは Request Reply の使用をサポートします。この場合、呼び出し元は Async ルートが完了するまで待機します。たとえば、以下のようになります。

```
from("mina:tcp://0.0.0.0:9876?textline=true&sync=true").to("disruptor:input");
from("disruptor:input").to("bean:processInput").to("bean:createResponse");
```

上記のルートには、受信要求を許可するポート 9876 に TCP リスナーがあります。リクエストは disruptor:input バッファーにルーティングされます。Request Reply メッセージであるため、レスポンスを待機します。disruptor:input バッファー上のコンシューマーが完了すると、応答を元のメッセージの応答にコピーします。

82.5. 同時コンシューマー

デフォルトでは、Disruptor エンドポイントは単一のコンシューマースレッドを使用しますが、同時実行コンシューマースレッドを使用するように設定できます。したがって、スレッドプールの代わりに以下を使用できます。

```
from("disruptor:stageName?concurrentConsumers=5").process(...)
```

この 2 つの差に応じて、スレッドプールは起動時に動的に増減する可能性があることに注意してください。ただし、同時にコンシューマーの数は常に、Diruptor が内部で固定、サポートされています。そ

のため、パフォーマンスが向上します。

82.6. スレッドプール

以下のように Disruptor エンドポイントへスレッドプールを追加することに注意してください。

```
from("disruptor:stageName").thread(5).process(...)
```

Disruptor と併用するために通常の BlockingQueue を追加することができ、Disruptor を使用してパフォーマンスが大幅に低下します。代わりに、concurrentConsumers オプションを使用して Disruptor エンドポイントでメッセージを処理するスレッドの数を直接設定するアドバイスがあります。

82.7. 例

以下のルートでは Disruptor を使用してリクエストをこの非同期キューに送信し、別のスレッドでさらに処理するために fire-and-forget メッセージを送信し、このスレッド内の定数応答を元の呼び出し元に返します。

```
public void configure() throws Exception {
    from("direct:start")
        // send it to the disruptor that is async
        .to("disruptor:next")
        // return a constant response
        .transform(constant("OK"));

    from("disruptor:next").to("mock:result");
}
```

ここでは Hello World のメッセージを送信し、応答が OK であることを想定します。

```
Object out = template.requestBody("direct:start", "Hello World");
assertEquals("OK", out);
```

「Hello World」メッセージは、さらなる処理のために、別のスレッドの Disruptor から消費されます。これはユニットテストからのものであるため、ユニットテストでアサーションを実行できるモックエンドポイントに送信されます。

82.8. MULTIPLECONSUMERS の使用

この例では、2 つのコンシューマーを定義し、それらを Spring Bean として登録しました。

```
<!-- define the consumers as spring beans -->
<bean id="consumer1" class="org.apache.camel.spring.example.FooEventConsumer"/>

<bean id="consumer2"
class="org.apache.camel.spring.example.AnotherFooEventConsumer"/>

<camelContext xmlns="http://camel.apache.org/schema/spring">
    <!-- define a shared endpoint which the consumers can refer to instead of using url -->
    <endpoint id="foo" uri="disruptor:foo?multipleConsumers=true"/>
</camelContext>
```

Disruptor foo エンドポイントで multipleConsumers=true を指定しているため、これらの 2 つ以上のコンシューマーが pub-sub スタイルのメッセージングの種類としてメッセージの独自のコピーを受け取ることができます。Bean はユニットテストの一部として、メッセージをモックエンドポイントに送信しますが、@Consume を使用して Disruptor から消費する方法に注目してください。

```
public class FooEventConsumer {

    @EndpointInject(uri = "mock:result")
    private ProducerTemplate destination;

    @Consume(ref = "foo")
    public void doSomething(String body) {
        destination.sendBody("foo" + body);
    }

}
```

82.9. 中断した情報の抽出

必要な場合は、この方法では JMX を使用せずにバッファーサイズなどの情報を取得できます。

```
DisruptorEndpoint disruptor = context.getEndpoint("disruptor:xxxx");
int size = disruptor.getBufferSize();
```

第83章 DNS コンポーネント

Camel バージョン 2.7 で利用可能

これは、DNSJava を使用して DNS クエリーを実行するための Camel の追加コンポーネントです。
コンポーネントは DNSJava 上のシン層です。
コンポーネントは、以下の操作を提供します。

- ip - IP でドメインを解決する
- ルックアップ（ドメインに関する情報を検索する）
- dig - DNS クエリーを実行します。

INFO:*Requires SUN JVM* DNSJava ライブラリーでは、SUN JVM で実行する必要があります。Apache ServiceMix または Apache Karaf を使用する場合は、etc/jre.properties ファイルを調整して sun.net.spi.nameservice をエクスポートされた Java プラットフォームパッケージの一覧に追加する必要があります。この変更を有効にするには、サーバーを再起動する必要があります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-dns</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

83.1. URI 形式

DNS コンポーネントの URI スキームは以下のとおりです。

```
dns://operation[?options]
```

このコンポーネントはプロデューサーのみをサポートします。

83.2. オプション

DNS コンポーネントにはオプションがありません。

DNS エンドポイントは、URI 構文を使用して設定します。

```
dns:dnsType
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

83.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
dnsType	**必要な** 検索のタイプ。		DnsType

83.2.2. クエリーパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

83.3. HEADERS

ヘッダー	Type	操作	説明
dns.domain	文字列	ip	ドメイン名。必須。
dns.name	文字列	lookup	検索する名前。必須。
dns.type		lookup、dig	ルックアップのタイプ。**org.xbill.dns.Type** の値と一致する必要があります。オプション。

ヘッダー	Type	操作	説明
dns.class		lookup、dig	ルックアップの DNS クラス。**org.xbill.dns.DClass** の値と一致する必要があります。オプション。
dns.query	文字列	dig	クエリー自体。必須。
dns.server	文字列	dig	クエリーに固有のサーバー。指定がない場合は、OS で指定されたデフォルトのものが使用されます。オプション。

83.4. 例

83.4.1. IP ルックアップ

```
<route id="IPCheck">
    <from uri="direct:start"/>
    <to uri="dns:ip"/>
</route>
```

これは、ドメインの IP を検索します。たとえば、www.example.com は 192.0.32.10 に解決されます。
ルックアップする IP アドレスは、「dns.domain」 キーのあるヘッダーで指定する必要があります。

83.4.2. DNS ルックアップ

```
<route id="IPCheck">
    <from uri="direct:start"/>
    <to uri="dns:lookup"/>
</route>
```

ドメインに関連付けられた DNS レコードのセットを返します。
ルックアップする名前は、キーが 「dns.name」 のヘッダーで指定する必要があります。

83.4.3. DNS Dig

dig は、DNS クエリーを実行するための Unix コマンドラインユーティリティーです。

```
<route id="IPCheck">
    <from uri="direct:start"/>
    <to uri="dns:dig"/>
</route>
```

クエリーはキー "dns.query" のヘッダーに指定する必要があります。

83.5. DNS アクティベーションキー

DnsActivationPolicy は、dns 状態に基づいてルートを動的に起動および停止するために使用できます。

同じコンポーネントのインスタンスが異なるリージョンで実行されている場合は、dns がリージョンを参照している場合にのみ、各リージョンにルートを設定してアクティブにすることができます。

つまり、NYC にインスタンスがあり、SFO にインスタンスがある可能性もあります。サービス CNAME service.example.com が nyc-service.example.com を参照するように設定し、NYC インスタンスを起動し、SFO インスタンスを停止します。CNAME service.example.com を変更して sfo-service.example.com autoMember-gitopsnyc インスタンスを参照すると、そのルートが停止し、sfo によりルートが起動します。これにより、実際のコンポーネントを再起動せずにリージョンを切り換えることができます。

```
 <bean id="dnsActivationPolicy"
class="org.apache.camel.component.dns.policy.DnsActivationPolicy">
  <property name="hostname" value="service.example.com" />
  <property name="resolvesTo" value="nyc-service.example.com" />
  <property name="ttl" value="60000" />
 </bean>

 <route id="routeId" autoStartup="false" routePolicyRef="dnsActivationPolicy">
 </route>
```

第84章 DOCKER コンポーネント

Camel バージョン 2.15 から利用可能

Docker と通信するための Camel コンポーネント。

Docker Camel コンポーネントは、Docker Remote API 経由で docker-java を使用します。

84.1. URI 形式

```
docker://[operation]?[options]
```

ここでの operation は Docker で実行する固有のアクションになります。

84.2. 一般的なオプション

Docker コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有 docker 設定を使用するには、以下を実行します。		DockerConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Docker エンドポイントは URI 構文を使用して設定します。

```
docker:operation
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

84.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
operation	使用する操作が **必要**		DockerOperation

84.2.2. クエリーパラメーター（20 パラメーター）：

Name	説明	デフォルト	Type
Email (common)	ユーザーに関連付けられたメールアドレス		文字列
ホスト （共通）	**必要な** Docker ホスト	localhost	文字列
ポート （共通）	**必要な** Docker ポート	2375	整数
requestTimeout (common)	応答の要求タイムアウト（秒単位）		整数
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cmdExecFactory (advanced)	使用する DockerCmdExecFactory 実装の完全修飾クラス名	com.github.dockerjava.netty.NettyDockerCmdExecFactory	文字列

Name	説明	デフォルト	Type
followRedirectFilter (advanced)	リダイレクトフィルターに従うかどうか。	false	boolean
loggingFilter (advanced)	ロギングフィルターを使用するかどうか。	false	boolean
maxPerRouteConnections (advanced)	最大ルート接続	100	整数
maxTotalConnections (advanced)	最大接続数	100	整数
serverAddress (advanced)	Docker レジストリーのサーバーアドレス。	https://index.docker.io/v1/	文字列
ソケット （詳細）	ソケット接続モード	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
certPath (security)	SSL 証明書チェーンを含む場所		文字列
パスワード （セキュリティー）	認証に使用するパスワード		文字列
セキュア （セキュリティー）	HTTPS 通信の使用	false	boolean
tlsVerify (security)	TLS の確認	false	boolean
ユーザー名 （セキュリティー）	認証するユーザー名		文字列

84.3. ヘッダーストラテジー

すべての URI オプションは Header プロパティーとして渡すことができます。メッセージヘッダーにある値は、URI パラメーターよりも優先されます。ヘッダープロパティーは、以下のように CamelDocker で始まる URI オプションの形式を取ります。

URI オプション	ヘッダープロパティー
containerId	CamelDockerContainerId

84.4. 例

以下の例では、Docker からのイベントを使用します。

```
from("docker://events?host=192.168.59.103&port=2375").to("log:event");
```

以下の例は、システム全体の情報について Docker をクエリーします。

```
from("docker://info?host=192.168.59.103&port=2375").to("log:info");
```

84.5. 依存関係

Camel ルートで Docker を使用するには、コンポーネントを実装する camel-docker の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加できます。バージョン番号は最新の最新のリリースに置き換えてください（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-docker</artifactId>
  <version>x.x.x</version>
</dependency>
```

第85章 DOZER コンポーネント

Camel バージョン 2.15 から利用可能

dozer: コンポーネントは、Camel 2.15.0 以降の Dozer マッピングフレームワークを使用して Java Bean 間でマッピングする機能を提供します。 Camel は、型コンバーターと して Dozer マッピングをトリガーする機能もサポートします。 Dozer エンドポイントと Dozer コンバーターの使用の主な相違点は以下のとおりです。

- コンバーターレジストリーを介して、エンドポイントごとの Dozer マッピング設定を管理する機能。
- Dozer エンドポイントを設定して、Camel データフォーマットを使用して入出力をマーシャリング/アンマーシャリングし、単一の任意の変換エンドポイントをサポートするように設定できます。
- Dozer コンポーネントを使用すると、Dzer の粒度の細かい統合や拡張により、追加機能をサポートできます（例： マッピングリテラル値のマッピング、マッピングの式の使用など）。

Dozer コンポーネントを使用するには、Maven ユーザーは以下の依存関係を pom.xml に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-dozer</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

85.1. URI 形式

Dozer コンポーネントはプロデューサーエンドポイントのみをサポートします。

```
dozer:endpointId[?options]
```

ここで、 endpointId は Dozer エンドポイント設定を一意に識別するために使用される名前です。

Dozer エンドポイント URI の例：

```
from("direct:orderInput").
  to("dozer:transformOrder?
mappingFile=orderMapping.xml&targetModel=example.XYZOrder").
  to("direct:orderOutput");
```

85.2. オプション

Dozer コンポーネントにはオプションがありません。

Dozer エンドポイントは URI 構文を使用して設定します。

```
dozer:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

85.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	人間が判読できるマッピングの名前が **必要です**。		文字列

85.2.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
mappingConfiguration (producer)	Dozer マッピングの設定に使用する Camel レジストリーの DozerBeanMapperConfiguration Bean の名前。これは、Dozer の設定方法を細かく制御するために使用可能な mappingFile オプションの代替です。値に接頭辞を使用して、Bean が Camel レジストリー（例： myDozerConfig）にあることを示しているのを忘れないようにしてください。		DozerBeanMapperConfiguration
mappingFile (producer)	Dozer 設定ファイルの場所。デフォルトでは、このファイルはクラスパスから読み込まれますが、file:、classpath:、または http: を使用して、特定の場所から設定を読み込むことができます。	dozerBeanMapping.xml	文字列

Name	説明	デフォルト	Type
marshalId (producer)	マッピング出力を Java 以外の型にマーシャリングするために使用する Camel Context 内で定義される dataFormat の ID。		文字列
sourceModel (producer)	マッピングで使用されるソースタイプの完全修飾クラス名。指定された場合は、マッピングへの入力は、Dzer でマッピングされる前に指定された型に変換されます。		文字列
targetModel (producer)	マッピングで使用されるターゲットタイプの **必須** 完全修飾クラス名。		文字列
unmarshalId (producer)	非 Java タイプからマッピング入力のアンマーシャリングに使用する Camel Context 内で定義される dataFormat の ID。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

85.3. DOZER でのデータフォーマットの使用

Dozer は、マッピング用の Java 以外のソースおよびターゲットをサポートしないため、XML ドキュメントを独自の Java オブジェクトにマッピングすることはできません。 Camel は主に、データ形式を使用して Java とさまざまな種類の形式のマーシャリングを幅広くサポートします。 Dozer コンポーネントは、Dzer で処理する前に入力データおよび出力データをデータ形式で渡すように指定することで、このサポートを活用します。 Dozer への呼び出し外でこれを常に行うことができますが、Dozer コンポーネントで直接サポートすることで、1 つのエンドポイントを使用して Camel 内の任意の変換を設定できます。

たとえば、Dzer コンポーネントを使用して XML データ構造と JSON データ構造をマッピングしたいとします。 Camel Context に以下のデータ形式が定義されている場合：

```
<dataFormats>
  <json library="Jackson" id="myjson"/>
  <jaxb contextPath="org.example" id="myjaxb"/>
</dataFormats>
```

その後、JAXB データフォーマットを使用して入力 XML をアンマーシャリングし、Jackson を使用してマッピング出力をマーシャリングするように Dozer エンドポイントを設定できます。

```
<endpoint uri="dozer:xml2json?
marshalId=myjson&amp;unmarshalId=myjaxb&amp;targetModel=org.example.Order"/>
```

85.4. DOZER の設定

Dozer エンドポイントには、ソースオブジェクトとターゲットオブジェクト間のマッピングを定義する Dozer マッピング設定ファイルが必要です。 mappingFile または mappingConfiguration オプションがエンドポイントで指定されていない場合、コンポーネントはデフォルトで META-INF/dozerBeanMapping.xml の場所に設定されます。 1 つのエンドポイントに複数のマッピング設定ファイルを指定する必要がある場合や、追加の設定オプション（イベントリスナーやカスタムコンバーターなど）を指定する場合、 org.apache.camel.converter.dozer.DozerBeanMapperConfiguration のインスタンスを使用できます。

```
<bean id="mapper" class="org.apache.camel.converter.dozer.DozerBeanMapperConfiguration">
  <property name="mappingFiles">
    <list>
      <value>mapping1.xml</value>
      <value>mapping2.xml</value>
    </list>
  </property>
</bean>
```

85.5. エクステンションのマッピング

Dozer コンポーネントは、多数のエクステンションを Dozer マッピングフレームワークにカスタムコンバーターとして実装します。 これらのコンバーターは、Dozer 自体で直接サポートされないマッピング機能を実装します。

85.5.1. 変数マッピング

変数マッピングを使用すると、Dozer 設定内の変数定義の値を、ソースフィールドの値を使用する代わりにターゲットフィールドにマップできます。 これは、他のマッピングフレームワークの定数マッピングと同等で、リテラル値をターゲットフィールドに割り当てることができます。 変数マッピングを使用するには、マッピング設定内で変数を定義してから、VariableMapper クラスから任意のターゲットフィールドにマップするだけです。

```
<mappings xmlns="http://dozermapper.github.io/schema/bean-mapping"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://dozermapper.github.io/schema/bean-mapping
http://dozermapper.github.io/schema/bean-mapping.xsd">
  <configuration>
    <variables>
      <variable name="CUST_ID">ACME-SALES</variable>
    </variables>
  </configuration>
  <mapping>
```

```
    <class-a>org.apache.camel.component.dozer.VariableMapper</class-a>
    <class-b>org.example.Order</class-b>
    <field custom-converter-id="_variableMapping" custom-converter-param="${CUST_ID}">
      <a>literal</a>
      <b>custId</b>
    </field>
  </mapping>
</mappings>
```

85.5.2. カスタムマッピング

カスタムマッピングにより、ソースフィールドをターゲットフィールドにマップするために独自のロジックを定義できます。 これらは Dozer の顧客コンバーターと機能的に似ており、以下の 2 つの大きな違いがあります。

- カスタムマッピングを使用すると、1 つのクラスに複数のコンバーターメソッドを含めることができます。
- カスタムマッピングを持つ Dozer 固有のインターフェースを実装する必要はありません。

カスタムマッピングは、マッピング設定で組み込みの '_customMapping' コンバーターを使用して宣言されます。 このコンバーターのパラメーターの構文は以下のとおりです。

```
[class-name][,method-name]
```

メソッド名はオプションです。Dozer コンポーネントは、マッピングに必要な入出力タイプに一致するメソッドを検索します。 カスタムマッピングと設定の例を以下に示します。

```
public class CustomMapper {
    // All customer ids must be wrapped in "[ ]"
    public Object mapCustomer(String customerId) {
        return "[" + customerId + "]";
    }
}
```

```
<mappings xmlns="http://dozermapper.github.io/schema/bean-mapping"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://dozermapper.github.io/schema/bean-mapping
http://dozermapper.github.io/schema/bean-mapping.xsd">
  <mapping>
    <class-a>org.example.A</class-a>
    <class-b>org.example.B</class-b>
    <field custom-converter-id="_customMapping"
        custom-converter-param="org.example.CustomMapper,mapCustomer">
```

```
      <a>header.customerNum</a>
      <b>custId</b>
    </field>
  </mapping>
</mappings>
```

85.5.3. 式マッピング

式マッピングを使用すると、Camel の強力な 言語 機能を使用して式を評価し、結果をマッピングのターゲットフィールドに割り当てることができます。 Camel がサポートする言語は、式マッピングで使用できます。 式の基本的な例には、Camel メッセージヘッダーまたはエクスチェンジプロパティーをターゲットフィールドにマップする機能、または複数のソースフィールドをターゲットフィールドに連結する機能が含まれます。 マッピング式の構文は以下のとおりです。

```
[language]:[expression]
```

メッセージヘッダーをターゲットフィールドにマップする例：

```
<mappings xmlns="http://dozermapper.github.io/schema/bean-mapping"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://dozermapper.github.io/schema/bean-mapping
http://dozermapper.github.io/schema/bean-mapping.xsd">
  <mapping>
    <class-a>org.apache.camel.component.dozer.ExpressionMapper</class-a>
    <class-b>org.example.B</class-b>
    <field custom-converter-id="_expressionMapping" custom-converter-
param="simple:\${header.customerNumber}">
      <a>expression</a>
      <b>custId</b>
    </field>
  </mapping>
</mappings>
```

Dozer が EL を使用して定義された変数値の解決を試行する際にエラーを防ぐために、式内のプロパティーは "\" でエスケープする必要があります。

第86章 ドリルコンポーネント

Camel バージョン 2.19 から利用可能

drill : コンポーネント を使用すると、Apache Drill Clusterへのクエリーが可能になります。

Drill は、Big Data exploration の Apache オープンソース SQL クエリーエンジンです。ドリルは、最新の Big Data アプリケーションから来る半構造化および迅速な進化データ上で高性能な分析をサポートするために開発され、業界標準のクエリー言語である ANSI SQL の知識とエコシステムを引き続き提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-drill</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

86.1. URI 形式

```
drill://host[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

86.2. ドリルプロデューサー

プロデューサーは CamelDrillQuery ヘッダーを使用してクエリーを実行し、結果をボディーに配置します。

86.3. オプション

Drill コンポーネントにはオプションがありません。

Drill エンドポイントは URI 構文を使用して設定します。

-

drill:host

以下の path パラメーターおよびクエリーパラメーターを使用します。

86.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
host	**必要な** ZooKeeper ホスト名または IP アドレス。ホスト名または IP アドレスの代わりに local を使用して、ローカルの Drillbit に接続します。		文字列

86.3.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
clusterId (producer)	Cluster ID https://drill.apache.org/docs/using-the-jdbc-driver/determining-the-cluster-id		文字列
ディレクトリー（プロデューサー）	ZooKeeper のドリルディレクトリー		文字列
モード（プロデューサー）	接続モード： zk: Zookeeper ドリルビット： Drillbit の直接接続 https://drill.apache.org/docs/using-the-jdbc-driver/	ZK	DrillConnectionMode
ポート（プロデューサー）	ZooKeeper port number		整数
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

86.4. 関連項目

- ***Configuring Camel (Camel の設定)***
- ***コンポーネント***

- *エンドポイント*
- *はじめに*

第87章 DROPBOX コンポーネント

Camel バージョン 2.14 から利用可能

dropbox: コンポーネントを使用すると、Dropbox リモートフォルダーをメッセージのプロデューサーまたはコンシューマーとして処理できます。Dropbox Java Core API（このコンポーネントのリファレンスバージョンは 1.7.x）を使用すると、この Camel コンポーネントには以下の機能があります。

- コンシューマーとして、クエリー別にファイルをダウンロードし、ファイルを検索します。
- プロデューサーとしてファイルをダウンロード、リモートディレクトリー間でファイルを移動、ファイル/ディレクトリーの削除、ファイル/ディレクトリーの削除、ファイルのアップロード、クエリーによるファイルの検索

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-dropbox</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

87.1. URI 形式

```
dropbox://[operation]?[options]
```

ここでの operation は Dropbox リモートフォルダーで実行する特定のアクション（通常は CRUD アクション）です。

87.2. 操作

操作	説明
del	Dropbox 上のファイルまたはディレクトリーを削除します。
get	Dropbox からファイルをダウンロード

操作	説明
move	Dropbox のフォルダーからファイルを移動
put	Dropbox でのファイルのアップロード
search	文字列クエリーに基づく Dropbox の検索

操作 が機能するには追加のオプションが必要です。特定の操作にはいくつかの必須となります。

87.3. オプション

Dropbox API と連携するには、 accessToken および clientIdentifier を取得する必要があります。取得方法を説明する Dropbox ドキュメント を参照してください。

Dropbox コンポーネントにはオプションがありません。

Dropbox エンドポイントは URI 構文を使用して設定します。

```
dropbox:operation
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

87.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
operation	Dropbox リモートフォルダーで実行する **必要** がある特定のアクション（通常は CRUD アクション）。		DropboxOperation

87.3.2. クエリーパラメーター（12 パラメーター）：

Name	説明	デフォルト	Type
accessToken (common)	特定の Dropbox ユーザーの API リクエストを行うためにアクセストークンが **必要**		文字列

Name	説明	デフォルト	Type
クライアント （共通）	既存の DbxClient インスタンスを DropBox クライアントとして使用します。		DbxClientV2
clientIdentifier (common)	API 要求を行うために登録されたアプリケーションの名前		文字列
localPath (common)	ローカルのファイルシステムから Dropbox にアップロードするオプションのフォルダーまたはファイル。このオプションが設定されていない場合、メッセージボディーはアップロードするコンテンツとして使用されます。		文字列
newRemotePath (common)	宛先ファイルまたはフォルダー		文字列
クエリー （共通）	検索するサブ文字列のスペース区切りリスト。ファイルは、サブ文字列がすべて含まれている場合にのみ一致します。このオプションが設定されていない場合、すべてのファイルが一致します。		文字列
remotePath (common)	移動する元のファイルまたはフォルダー		文字列
uploadMode (common)	アップロードするモード。同じ名前のファイルが dropbox に存在する場合は、新しいファイルの名前が変更されます。同じ名前のファイルがドロップボックスにすでに存在する場合は、これは上書きされます。		DropboxUploadMode
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

87.4. DEL 操作

Dropbox 上のファイルを削除します。

Camel プロデューサーとしてのみ動作します。

この操作のオプションを以下に示します。

プロパティー	必須	説明
remotePath	**true**	Dropbox で削除するフォルダーまたはファイル

87.4.1. サンプル

```
from("direct:start")
  .to("dropbox://del?accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1")
  .to("mock:result");

from("direct:start")
  .to("dropbox://del?
accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1/file1.tar.gz")
  .to("mock:result");
```

87.4.2. 結果メッセージヘッダー

メッセージの結果には、以下のヘッダーが設定されます。

プロパティー	値
DELETED_PATH	dropbox で削除されたパスの名前

87.4.3. 結果メッセージのボディー

以下のオブジェクトはメッセージボディーの結果に設定されます。

オブジェクトの種類	説明
文字列	dropbox で削除されたパスの名前

87.5. GET(DOWNLOAD)操作

Dropbox からファイルをダウンロードします。

Camel プロデューサーまたは Camel コンシューマーとして機能します。

この操作のオプションを以下に示します。

プロパティー	必須	説明
remotePath	**true**	Dropbox からダウンロードするフォルダーまたはファイル

87.5.1. サンプル

```
from("direct:start")
  .to("dropbox://get?
accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1/file1.tar.gz")
  .to("file:///home/kermit/?fileName=file1.tar.gz");

from("direct:start")
  .to("dropbox://get?accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1")
  .to("mock:result");

from("dropbox://get?accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1")
  .to("file:///home/kermit/");
```

87.5.2. 結果メッセージヘッダー

メッセージの結果には、以下のヘッダーが設定されます。

プロパティー	値

プロパティー	値
DOWNLOADED_FILE	ファイルのダウンロードが 1 つある場合は、ダウンロードしたリモートファイルのパス。
DOWNLOADED_FILES	複数のファイルのダウンロードの場合は、ダウンロードしたリモートファイルのパス。

87.5.3. 結果メッセージのボディー

以下のオブジェクトはメッセージボディーの結果に設定されます。

オブジェクトの種類	説明
ByteArrayOutputStream	単一ファイルダウンロードの場合、ダウンロードしたファイルを表すストリーム。
Map<String, ByteArrayOutputStream>	複数のファイルのダウンロードの場合、ダウンロードしたリモートファイルのパスとして、ダウンロードしたファイルを表すストリームとしてキーを持つマップ。

87.6. MOVE 操作

Dropbox 上のファイルをあるフォルダーから別のフォルダーに移動します。

Camel プロデューサーとしてのみ動作します。

この操作のオプションを以下に示します。

プロパティー	必須	説明
remotePath	**true**	移動する元のファイルまたはフォルダー
newRemotePath	**true**	宛先ファイルまたはフォルダー

87.6.1. サンプル

```
from("direct:start")
  .to("dropbox://move?
```

```
accessToken=XXX&clientIdentifier=XXX&remotePath=/root/folder1&newRemotePath=/root/folder2")
  .to("mock:result");
```

87.6.2. 結果メッセージヘッダー

メッセージの結果には、以下のヘッダーが設定されます。

プロパティー	値
MOVED_PATH	dropbox で移動されたパスの名前

87.6.3. 結果メッセージのボディー

以下のオブジェクトはメッセージボディーの結果に設定されます。

オブジェクトの種類	説明
文字列	dropbox で移動されたパスの名前

87.7. PUT（アップロード）操作

Dropbox でファイルをアップロードします。

Camel プロデューサーとして動作します。

この操作のオプションを以下に示します。

プロパティー	必須	説明
uploadMode	**true**	このオプションを追加または強制的に指定すると、ドロップボックスにファイルを保存する方法を指定します。「add」の場合は、同じ名前のファイルがドロップボックスに存在する場合、新しいファイルの名前が変更されます。同じ名前のファイルがドロップボックスに存在する場合は「force」の場合、これは上書きされます。

プロパティー	必須	説明
localPath	**false**	ローカルファイルシステムから Dropbox にアップロードするフォルダーまたはファイル。このオプションを設定すると、Camel メッセージボディーからのコンテンツが含まれる単一ファイルとしてアップロードよりも優先されます（メッセージ本文はバイト配列に変換されます）。
remotePath	**false**	Dropbox のフォルダー宛先。プロパティーが設定されていない場合、コンポーネントはローカルパスと同等のリモートパス上のファイルをアップロードします。Windows または絶対 localPath がない場合は、以下のように例外で実行できます。 原因： java.lang.IllegalArgumentException: 'path': bad path: must start with "/": "C:/My/File" OR Caused by: java.lang.IllegalArgumentException: 'path': bad path: must start with "/": "MyFile"

87.7.1. サンプル

```
from("direct:start").to("dropbox://put?
accessToken=XXX&clientIdentifier=XXX&uploadMode=add&localPath=/root/folder1")
  .to("mock:result");

from("direct:start").to("dropbox://put?
accessToken=XXX&clientIdentifier=XXX&uploadMode=add&localPath=/root/folder1&remotePa
th=/root/folder2")
  .to("mock:result");
```

そして、メッセージボディーからコンテンツを含む単一のファイルをアップロードする場合

```
from("direct:start")
   .setHeader(DropboxConstants.HEADER_PUT_FILE_NAME, constant("myfile.txt"))
   .to("dropbox://put?
accessToken=XXX&clientIdentifier=XXX&uploadMode=add&remotePath=/root/folder2")
   .to("mock:result");
```

ファイルの名前は、その順序で DropboxConstants.HEADER_PUT_FILE_NAME または Exchange.FILE_NAME ヘッダーで指定できます。ヘッダーが指定されていない場合、ファイル名としてメッセージ ID(uuid)が使用されます。

87.7.2. 結果メッセージヘッダー

メッセージの結果には、以下のヘッダーが設定されます。

プロパティー	値
UPLOADED_FILE	単一ファイルのアップロードの場合、アップロードしたリモートパスのパス。
UPLOADED_FILES	複数のファイルをアップロードする場合、リモートパスがアップロードされた文字列。

87.7.3. 結果メッセージのボディー

以下のオブジェクトはメッセージボディーの結果に設定されます。

オブジェクトの種類	説明
文字列	単一ファイルのアップロードの場合は、アップロード操作、OK、または KO が発生します。
Map<String, DropboxResultCode>	複数のファイルをアップロードする場合、アップロードするリモートファイルのパスと、アップロード操作の結果、OK または KO キーを持つマップ。

87.8. 検索操作

サブディレクトリーを含むリモート Dropbox フォルダー内で検索します。

Camel プロデューサーおよび Camel コンシューマーとして機能します。

この操作のオプションを以下に示します。

プロパティー	必須	説明
remotePath	**true**	検索先の Dropbox のフォルダー。
query	**true**	検索するサブ文字列のスペース区切りリスト。ファイルは、サブ文字列がすべて含まれている場合にのみ一致します。このオプションが設定されていない場合、すべてのファイルが一致します。クエリーはエンドポイント設定または Camel メッセージのヘッダー **CamelDropboxQuery** で提供される必要があります。

87.8.1. サンプル

```
from("dropbox://search?
accessToken=XXX&clientIdentifier=XXX&remotePath=/XXX&query=XXX")
  .to("mock:result");

from("direct:start")
  .setHeader("CamelDropboxQuery", constant("XXX"))
  .to("dropbox://search?accessToken=XXX&clientIdentifier=XXX&remotePath=/XXX")
  .to("mock:result");
```

87.8.2. 結果メッセージヘッダー

メッセージの結果には、以下のヘッダーが設定されます。

プロパティー	値
FOUNDED_FILES	見つかったファイルパスの一覧

87.8.3. 結果メッセージのボディー

以下のオブジェクトはメッセージボディーの結果に設定されます。

オブジェクトの種類	説明
List<DbxEntry>	見つかったファイルパスの一覧。このオブジェクトの詳細については、Dropbox のドキュメントを参照してください。

第88章 EHCACHE コンポーネント

Camel バージョン 2.18 から利用可能

ehcache コンポーネントを使用すると、キャッシュ実装として Ehcache 3 を使用してキャッシュ操作を実行できます。

このコンポーネントは、プロデューサーおよびイベントベースのコンシューマーエンドポイントをサポートします。

キャッシュコンシューマーはイベントベースのコンシューマーで、特定のキャッシュアクティビティーをリッスンして応答するために使用できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ehcache</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

88.1. URI 形式

```
ehcache://cacheName[?options]
```

URI には、 ?option=value&option=#beanRef&... という形式でクエリーオプションを追加できます。

88.2. オプション

Ehcache コンポーネントは、以下に示す 7 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	グローバルコンポーネントの設定を設定します。		EhcacheConfiguration

Name	説明	デフォルト	Type
cacheManager (common)	キャッシュマネージャー		CacheManager
CacheManager 設定 （共通）	キャッシュマネージャーの設定		設定
cacheConfiguration (common)	キャッシュの作成に使用されるデフォルトのキャッシュ設定。		CacheConfiguration<?,?>
cachesConfigurations (common)	キャッシュの作成に使用されるキャッシュ設定のマップ。		マップ
cacheConfigurationUri (common)	Ehcache XML 設定ファイルの場所を参照する URI		文字列
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ehcache エンドポイントは URI 構文を使用します。

```
ehcache:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

88.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	キャッシュ名が **必要です**。		文字列

88.2.2. クエリーパラメーター（17 パラメーター）：

Name	説明	デフォルト	Type
cacheManager (common)	キャッシュマネージャー		CacheManager

Name	説明	デフォルト	Type
cacheManagerConfiguration (common)	キャッシュマネージャーの設定		設定
configurationUri (common)	Ehcache XML 設定ファイルの場所を参照する URI		文字列
createCacheIfNotExist (common)	キャッシュが存在する場合や事前設定できない場合は、キャッシュを作成する必要がある場合を設定します。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
eventFiring (consumer)	配信モードの設定（同期、非同期）	非同期	EventFiring
eventOrdering (consumer)	配信モードの設定（順序なし、順序なし）	ORDERED	EventOrdering
eventTypes (consumer)	リッスンするイベントのタイプを設定します。	EVICTED、EXPIRED、REMOVED、CREATED、UPDATED	Set
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
アクション （プロデューサー）	デフォルトのキャッシュアクションを設定します。アクションがメッセージヘッダーに設定されている場合は、ヘッダーからの操作が優先されます。		文字列
キー （プロデューサー）	デフォルトのアクションキーを設定します。キーがメッセージヘッダーに設定されている場合は、ヘッダーのキーが優先されます。		オブジェクト
Configuration (advanced)	キャッシュの作成に使用されるデフォルトのキャッシュ設定。		CacheConfiguration<?,?>
設定 （詳細）	キャッシュの作成に使用されるキャッシュ設定のマップ。		マップ
keyType (advanced)	キャッシュキータイプ（デフォルトは java.lang.Object）	java.lang.Object	文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
valueType (advanced)	キャッシュ値タイプ（デフォルトは java.lang.Object）	java.lang.Object	文字列

88.2.3. Message Headers Camel

ヘッダー	Type	説明
CamelEhcacheAction	**文字列**	キャッシュ上で実施される操作。有効なオプションは以下のとおりです。 * PUT * PUT_ALL * PUT_IF_ABSENT * GET * GET_ALL * REMOVE * REMOVE_ALL * REPLACE
CamelEhcacheActionHasResult	ブール値	アクションの結果がある場合は true に設定します。

ヘッダー	Type	説明
CamelEhcacheActionSucceeded	**ブール値**	アクションを生じさせる場合は true に設定します。
CamelEhcacheKey	オブジェクト	アクションに使用するキャッシュキー
CamelEhcacheKeys	set<Object>	キーの一覧（「」で使用） * PUT_ALL * GET_ALL * REMOVE_ALL
CamelEhcacheValue	オブジェクト	キャッシュに追加する値または操作の結果
CamelEhcacheOldValue	オブジェクト	PUT_IF_ABSENT などのアクションのキーに関連付けられた以前の値、または REPLACE などのアクションの比較に使用するオブジェクト
CamelEhcacheEventType	EventType	受信するイベントのタイプ

88.3. EHCACHE ベースのべき等リポジトリーの例：

```
CacheManager manager = CacheManagerBuilder.newCacheManager(new
XmlConfiguration("ehcache.xml"));
EhcacheIdempotentRepository repo = new EhcacheIdempotentRepository(manager,
"idempotent-cache");

from("direct:in")
    .idempotentConsumer(header("messageId"), idempotentRepo)
    .to("mock:out");
```

88.4. EHCACHE ベースの集計リポジトリーの例：

```
public class EhcacheAggregationRepositoryRoutesTest extends CamelTestSupport {
    private static final String ENDPOINT_MOCK = "mock:result";
```

```
    private static final String ENDPOINT_DIRECT = "direct:one";
    private static final int[] VALUES = generateRandomArrayOfInt(10, 0, 30);
    private static final int SUM = IntStream.of(VALUES).reduce(0, (a, b) -> a + b);
    private static final String CORRELATOR = "CORRELATOR";

    @EndpointInject(uri = ENDPOINT_MOCK)
    private MockEndpoint mock;

    @Produce(uri = ENDPOINT_DIRECT)
    private ProducerTemplate producer;

    @Test
    public void checkAggregationFromOneRoute() throws Exception {
        mock.expectedMessageCount(VALUES.length);
        mock.expectedBodiesReceived(SUM);

        IntStream.of(VALUES).forEach(
            i -> producer.sendBodyAndHeader(i, CORRELATOR, CORRELATOR)
        );

        mock.assertIsSatisfied();
    }

    private Exchange aggregate(Exchange oldExchange, Exchange newExchange) {
        if (oldExchange == null) {
            return newExchange;
        } else {
            Integer n = newExchange.getIn().getBody(Integer.class);
            Integer o = oldExchange.getIn().getBody(Integer.class);
            Integer v = (o == null ? 0 : o) + (n == null ? 0 : n);

            oldExchange.getIn().setBody(v, Integer.class);

            return oldExchange;
        }
    }

    @Override
    protected RoutesBuilder createRouteBuilder() throws Exception {
        return new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                from(ENDPOINT_DIRECT)
                    .routeId("AggregatingRouteOne")
                    .aggregate(header(CORRELATOR))
                    .aggregationRepository(createAggregateRepository())
                    .aggregationStrategy(EhcacheAggregationRepositoryRoutesTest.this::aggregate)
                    .completionSize(VALUES.length)

.to("log:org.apache.camel.component.ehcache.processor.aggregate.level=INFO&showAll=tru
e&mulltiline=true")
                    .to(ENDPOINT_MOCK);
            }
        };
    }
```

```
    protected EhcacheAggregationRepository createAggregateRepository() throws Exception {
        CacheManager cacheManager = CacheManagerBuilder.newCacheManager(new
XmlConfiguration("ehcache.xml"));
        cacheManager.init();

        EhcacheAggregationRepository repository = new EhcacheAggregationRepository();
        repository.setCacheManager(cacheManager);
        repository.setCacheName("aggregate");

        return repository;
    }
}
```

第89章 EJB コンポーネント

Camel バージョン 2.4 で利用可能

ejb: コンポーネントは EJB を Camel メッセージエクスチェンジにバインドします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ejb</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

89.1. URI 形式

```
ejb:ejbName[?options]
```

ejbName は、アプリケーションサーバーの JNDI レジストリーで EJB を検索するために使用される任意の文字列になります。

89.2. オプション

EJB コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
context (producer)	EJB の検索に使用するコンテキスト		コンテキスト
プロパティー（プロデューサー）	コンテキストが設定されていない場合に javax.naming.Context を作成するプロパティー。		プロパティー
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

EJB エンドポイントは、URI 構文を使用して設定します。

```
ejb:beanName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

89.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
beanName	**Required**: 呼び出す Bean の名前を設定します。		文字列

89.2.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
メソッド（プロデューサー）	Bean で呼び出すメソッドの名前を設定します。		文字列
キャッシュ（詳細）	有効にすると、Camel は最初のレジストリールックアップの結果をキャッシュします。レジストリーの Bean がシングルトンスコープとして定義されている場合は、キャッシュを有効にすることができます。	false	boolean
multiParameterArray (advanced)	**非推奨**: メッセージボディーから渡されるパラメーターを処理する方法。true は、メッセージボディーがパラメーターの配列である必要があることを意味します。非推奨注記： このオプションは Camel によって内部的に使用され、エンドユーザー向けの予定はありません。非推奨注記： このオプションは Camel によって内部的に使用され、エンドユーザー向けの予定はありません。	false	boolean
パラメーター（詳細）	Bean の追加プロパティーの設定に使用されます。		マップ
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

89.3. BEAN バインディング

呼び出される Bean メソッドを選択する方法（メソッド パラメーターで明示的に指定されていない場合）と、メッセージからパラメーター値が、Camel のさまざまな Bean 統合メカニズムすべてで使用される Bean バインディングメカニズムによって定義される方法。

89.4. 例

以下の例では、以下のように定義される Greater EJB を使用します。

GreaterLocal.java

```
public interface GreaterLocal {

    String hello(String name);

    String bye(String name);

}
```

実装方法

GreaterImpl.java

```
@Stateless
public class GreaterImpl implements GreaterLocal {

    public String hello(String name) {
        return "Hello " + name;
    }

    public String bye(String name) {
        return "Bye " + name;
    }

}
```

89.4.1. Java DSL の使用

この例では、EJB で hello メソッドを呼び出します。この例は Apache OpenEJB を使用したユニットテストを使用するため、OpenEJB 設定で EJB コンポーネントに JndiContext を設定する必要があります。

```
@Override
protected CamelContext createCamelContext() throws Exception {
```

```
    CamelContext answer = new DefaultCamelContext();

    // enlist EJB component using the JndiContext
    EjbComponent ejb = answer.getComponent("ejb", EjbComponent.class);
    ejb.setContext(createEjbContext());

    return answer;
}

private static Context createEjbContext() throws NamingException {
    // here we need to define our context factory to use OpenEJB for our testing
    Properties properties = new Properties();
    properties.setProperty(Context.INITIAL_CONTEXT_FACTORY,
"org.apache.openejb.client.LocalInitialContextFactory");

    return new InitialContext(properties);
}
```

その後、Camel ルートで EJB を使用する準備が整いました。

```
from("direct:start")
    // invoke the greeter EJB using the local interface and invoke the hello method
    .to("ejb:GreaterImplLocal?method=hello")
    .to("mock:result");
```

実際のアプリケーションサーバー

実際のアプリケーションサーバーでは、EJB コンポーネントで JndiContext を設定する必要はありません。これはアプリケーションサーバーと同じ JVM でデフォルトの JndiContext を作成するため、通常は JNDI レジストリーにアクセスして EJBをルックアップできるためです。ただし、リモート JVM またはこのような方法でアプリケーションサーバーにアクセスする必要がある場合は、事前にプロパティーを準備する必要があります。

89.4.2. Spring XML の使用

これは、代わりに Spring XML を使用した例と同じです。

これはユニットテストに基づくため、EJB コンポーネントを設定する必要があります。

```
<!-- setup Camel EJB component -->
<bean id="ejb" class="org.apache.camel.component.ejb.EjbComponent">
    <property name="properties" ref="jndiProperties"/>
</bean>

<!-- use OpenEJB context factory -->
```

```
<p:properties id="jndiProperties">
    <prop
key="java.naming.factory.initial">org.apache.openejb.client.LocalInitialContextFactory</prop>
</p:properties>
```

Camel ルートで EJB を使用する準備ができる前に、以下を行います。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:start"/>
        <to uri="ejb:GreaterImplLocal?method=hello"/>
        <to uri="mock:result"/>
    </route>
</camelContext>
```

89.5. 関連項目

- **Configuring Camel (Camel の設定)**
- コンポーネント
- エンドポイント
- はじめに
- **Bean**
- **Bean バインディング**
- **Bean インテグレーション**

第90章 ELASTICSEARCH コンポーネント（非推奨）

Camel バージョン 2.11 で利用可能

ElasticSearch コンポーネントを使用すると、ElasticSearch サーバーとのインターフェースを実行できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-elasticsearch</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

90.1. URI 形式

```
elasticsearch://clusterName[?options]
```

90.2. エンドポイントオプション

Elasticsearch コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
クライアント（詳細）	エンドポイントごとにクライアントを作成する代わりに、既存の設定された Elasticsearch クライアントを使用します。		クライアント
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Elasticsearch エンドポイントは URI 構文を使用して設定します。

```
elasticsearch:clusterName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

90.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
clusterName	**必要な** クラスター名、またはローカルモードに local を使用します。		文字列

90.2.2. クエリーパラメーター（11 パラメーター）：

Name	説明	デフォルト	Type
clientTransportSniff (producer)	クライアントでは、クラスターの他の部分をスニッフィングできるクライアントを指します（デフォルトは true）。この設定は client.transport.sniff 設定にマッピングします。	true	ブール値
consistencyLevel (producer)	INDEX 操作および BULK 操作で使用する書き込み整合性レベル（ONE、QUORUM、ALL、または DEFAULT のいずれか）。	DEFAULT	WriteConsistencyLevel
データ （プロデューサー）	これは、データ（シャード）をそのノードに割り当てることを許可するノードを指します。この設定は、node.data の設定にマップします。		ブール値
indexName (producer)	動作させるインデックスの名前		文字列
indexType (producer)	動作させるインデックスのタイプ		文字列
IP （プロデューサー）	使用する TransportClient リモートホスト ip		文字列
操作 （プロデューサー）	実行する操作		文字列
pathHome (producer)	ElasticSearch 設定の path.home プロパティー。有効なパスを指定する必要があります。指定しない場合は、デフォルトの $user.home/.elasticsearch が使用されます。	${user.home}/.elasticsearch	文字列

Name	説明	デフォルト	Type
ポート （プロデューサー）	使用する TransportClient リモートポート（デフォルトは 9300）。	9300	int
transportAddresses (producer)	使用する ip:port 形式のリモートトランスポートアドレスのコンマ区切りリスト。transportAddresses を代わりに考慮するには、ip オプションおよび port オプションを空白のままにする必要があります。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

90.3. ローカルテスト

ローカル(JVM/classloader)ElasticSearch サーバーに対して実行する場合は、clusterName の値を URI の「local」に設定するだけです。詳細は、『クライアントガイド』 を参照してください。

90.4. メッセージ操作

現在、以下の ElasticSearch 操作がサポートされています。エンドポイント URI オプションまたはエクスチェンジヘッダーを「operation」のキーと、以下の値のいずれかに設定されるだけです。一部の操作では、他のパラメーターやメッセージボディーも設定する必要があります。

operation	メッセージボディー	説明
INDEX	Map、String、byte[] または XContentBuilder コンテンツを index にします。	インデックスにコンテンツを追加し、本文にコンテンツの indexId を返します。**Camel 2.15 では、** メッセージヘッダーをキー "indexId" に設定することで indexId を設定できます。

operation	メッセージボディー	説明
GET_BY_ID	取得するコンテンツのインデックス ID	指定されたインデックスを取得し、本文の GetResult オブジェクトを返します。
DELETE	削除するコンテンツのインデックス ID	指定した indexId を削除し、本文に DeleteResult オブジェクトを返します。
BULK_INDEX	許可されるすべてのタイプの **List** または **Collection**（XContentBuilder、Map、byte[]、String）	*Camel 2.14,* コンテンツをインデックスに追加し、本文で正常にインデックス化されたドキュメントの ID の一覧を返す

operation	メッセージボディー	説明
一括	許可されるすべてのタイプの **List** または **Collection**（XContentBuilder、Map、byte[]、String）	**camel 2.15:** コンテンツをインデックスに追加し、ボディーに BulkResponse オブジェクトを返します。
SEARCH	マップまたは SearchRequest オブジェクト	**Camel 2.15:** クエリー文字列のマップでコンテンツを検索します。
MULTIGET	MultigetRequest.Item オブジェクトの一覧	**Camel 2.17:** 指定されたインデックス、型などを MultigetRequest で取得し、ボディーに MultigetResponse オブジェクトを返します。
MULTISEARCH	SearchRequest オブジェクトの一覧	**Camel 2.17:** MultiSearchRequest で指定されたパラメーターを検索し、ボディーに MultiSearchResponse オブジェクトを返します。
EXISTS	ヘッダーとしてインデックス名	**Camel 2.17: ボディーのブール値オブジェクトを返します。**

operation	メッセージボディー	説明
UPDATE	更新する Map、String、byte[] または XContentBuilder コンテンツ	Camel 2.17: コンテンツをインデックスに更新し、本文の内容の indexId を返します。

90.5. インデックスの例

以下は簡単な INDEX の例です。

```
from("direct:index")
.to("elasticsearch://local?operation=INDEX&indexName=twitter&indexType=tweet");
```

```
<route>
  <from uri="direct:index" />
  <to uri="elasticsearch://local?operation=INDEX&indexName=twitter&indexType=tweet"/>
</route>
```

クライアントは、マップを含むボディーメッセージをルートに渡すだけです。結果のボディーには、作成された indexId が含まれます。

```
Map<String, String> map = new HashMap<String, String>();
map.put("content", "test");
String indexId = template.requestBody("direct:index", map, String.class);
```

90.6. 詳細情報は、これらのリソースを参照してください。

Elasticsearch Main Site

ElasticSearch Java API

90.7. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第91章 ELASTICSEARCH5 コンポーネント（非推奨）

Camel バージョン 2.19 から利用可能

ElasticSearch コンポーネントを使用すると、ElasticSearch 5.x API とのインターフェースを実行できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-elasticsearch5</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

91.1. URI 形式

```
elasticsearch5://clusterName[?options]
```

91.2. エンドポイントオプション

Elasticsearch5 コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
クライアント（詳細）	エンドポイントごとにクライアントを作成する代わりに、既存の設定された Elasticsearch クライアントを使用します。これにより、特定の設定でクライアントをカスタマイズできます。		TransportClient
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Elasticsearch5 エンドポイントは URI 構文を使用して設定されます。

```
elasticsearch5:clusterName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

91.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
clusterName	クラスターの **必須** 名		文字列

91.2.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
clientTransportSniff (producer)	クライアントは残りのクラスターをスニスさせることができるかどうかです。この設定は client.transport.sniff 設定にマッピングします。	false	boolean
indexName (producer)	動作させるインデックスの名前		文字列
indexType (producer)	動作させるインデックスのタイプ		文字列
IP （プロデューサー）	使用する TransportClient リモートホスト ip		文字列
操作 （プロデューサー）	実行する操作		ElasticsearchOperation
pingSchedule (producer)	クライアントがクラスターに ping する時間（単位）。	5s	文字列
pingTimeout (producer)	ノードからの ping 応答を待機する時間（単位）。	5s	文字列
ポート （プロデューサー）	使用する TransportClient リモートポート（デフォルトは 9300）。	9300	int
tcpCompress (producer)	圧縮(LZF)が全ノード間で有効にする場合は True。	false	boolean
tcpConnectTimeout (producer)	接続タイムアウトを待つ時間（単位）。	30s	文字列

Name	説明	デフォルト	Type
transportAddresses (producer)	使用する ip:port 形式のリモートトランスポートアドレスのコンマ区切りリスト。transportAddresses を代わりに考慮するには、ip オプションおよび port オプションを空白のままにする必要があります。		文字列
waitForActiveShards (producer)	インデックス作成は、シャードの書き込みの整合性数が利用可能になるまで待機します。	1	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
enableSSL (security)	SSL を有効にします。クラスパスに XPack クライアント jar が必要	false	boolean
パスワード （認証）	クラスターに対して認証を行うためのパスワード。クラスパスに XPack クライアント jar が必要		文字列
ユーザー （認証）	クラスターに対して認証するためのユーザー。クラスターにアクセスするために transport_client ロールが必要です。クラスパスに XPack クライアント jar が必要		文字列

91.3. メッセージ操作

現在、以下の ElasticSearch 操作がサポートされています。エンドポイント URI オプションまたはエクスチェンジヘッダーを「operation」のキーと、以下の値のいずれかに設定されるだけです。一部の操作では、他のパラメーターやメッセージボディーも設定する必要があります。

operation	メッセージボディー	説明

operation	メッセージボディー	説明
INDEX	Map、String、byte[] または XContentBuilder コンテンツを index にします。	インデックスにコンテンツを追加し、本文にコンテンツの indexId を返します。キー "indexId" でメッセージヘッダーを設定して indexId を設定できます。
GET_BY_ID	取得するコンテンツのインデックス ID	指定されたインデックスを取得し、本文の GetResult オブジェクトを返します。
DELETE	削除するコンテンツのインデックス名およびタイプ	指定した indexName および indexType を削除し、本文に DeleteResponse オブジェクトを返します。
DELETE_INDEX	削除するコンテンツのインデックス名	指定した indexName を削除し、本文に DeleteIndexResponse オブジェクトを返します。

operation	メッセージボディー	説明
BULK_INDEX	許可されるすべてのタイプの **List** または **Collection**（XContentBuilder、Map、byte[]、String）	インデックスにコンテンツを追加し、本文で正常にインデックス化されたドキュメントの ID の一覧を返します。
一括	許可されるすべてのタイプの **List** または **Collection**（XContentBuilder、Map、byte[]、String）	インデックスにコンテンツを追加し、ボディーに BulkResponse オブジェクトを返します。
SEARCH	Map、String、または SearchRequest オブジェクト	クエリー文字列のマップでコンテンツを検索します。

operation	メッセージボディー	説明
MULTIGET	MultigetRequest.Item オブジェクトの一覧	MultigetRequest で指定のインデックス、タイプなどを取得し、ボディーに MultigetResponse オブジェクトを返します。
MULTISEARCH	SearchRequest オブジェクトの一覧	MultiSearchRequest に指定されたパラメーターを検索し、ボディーに MultiSearchResponse オブジェクトを返します。
EXISTS	ヘッダーとしてインデックス名	インデックスが存在するかチェックし、本文にブール値フラグを返します。
UPDATE	更新する Map、String、byte[] または XContentBuilder コンテンツ	インデックスにコンテンツを更新し、本文に含まれるコンテンツの indexId を返します。

91.4. インデックスの例

以下は簡単な INDEX の例です。

```
from("direct:index")
.to("elasticsearch5://elasticsearch?operation=INDEX&indexName=twitter&indexType=tweet");
```

```
<route>
  <from uri="direct:index" />
  <to uri="elasticsearch5://elasticsearch?operation=INDEX&indexName=twitter&indexType=tweet"/>
</route>
```

クライアントは、マップを含むボディーメッセージをルートに渡すだけです。結果のボディーには、作成された indexId が含まれます。

```
Map<String, String> map = new HashMap<String, String>();
map.put("content", "test");
String indexId = template.requestBody("direct:index", map, String.class);
```

91.5. 詳細情報は、これらのリソースを参照してください。

Elastic Main Site（Elastic メインサイト）

ElasticSearch Java API

91.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第92章 ELASTICHSEARCH REST COMPONENT

Camel バージョン 2.21 で利用可能

ElasticSearch コンポーネントを使用すると、REST クライアントライブラリーを使用して ElasticSearch 6.x API をインターフェースできます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-elasticsearch-rest</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

92.1. URI 形式

```
elasticsearch-rest://clusterName[?options]
```

92.2. エンドポイントオプション

Elastichsearch Rest コンポーネントは 12 個のオプションをサポートします。

Name	説明	デフォルト	Type
クライアント（詳細）	エンドポイントごとにクライアントを作成する代わりに、既存の設定された Elasticsearch クライアントを使用します。これにより、特定の設定でクライアントをカスタマイズできます。		RestClient
hostAddresses (advanced)	使用する ip:port 形式のリモートトランスポートアドレスのコンマ区切りリスト。hostAddresses を代わりに考慮するには、ip オプションおよび port オプションを空白のままにする必要があります。		文字列
socketTimeout (advanced)	ソケットがタイムアウトするまで待機するタイムアウト（ミリ秒単位）。	30000	int
connectionTimeout (advanced)	接続がタイムアウトするまでの待機時間（ミリ秒単位）。	30000	int

Name	説明	デフォルト	Type
user (advance)	基本的な認証ユーザー		文字列
password (producer)	認証のパスワード		文字列
enableSSL (advanced)	SSL の有効化	false	ブール値
maxRetryTimeout (advanced)	再試行までの時間（ミリ秒単位）。	30000	int
enableSniffer (advanced)	実行中の Elasticsearch クラスターからノードを自動的に検出します。	false	ブール値
snifferInterval (advanced)	連続した通常のスニファ実行の間隔（ミリ秒単位）。sniffOnFailure が無効になったとき、または連続してスニファーの実行の間に失敗がない場合に非表示になります。	300000	int
sniffAfterFailureDelay (advanced)	失敗後にスケジュールされるスニファ実行の遅延（ミリ秒単位）	60000	int
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Elastichsearch Rest エンドポイントは、URI 構文を使用して設定します。

```
elasticsearch-rest:clusterName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

92.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
clusterName	クラスターの **必須** 名		文字列

92.2.2. クエリーパラメーター（11 パラメーター）：

Name	説明	デフォルト	Type
connectionTimeout (producer)	接続がタイムアウトするまでの待機時間（ミリ秒単位）。	30000	int
disconnect (producer)	プロデューサーの呼び出しが完了した後に切断します。	false	boolean
enableSSL (producer)	SSL の有効化	false	boolean
hostAddresses (producer)	使用する ip:port 形式のリモートトランスポートアドレスのある **必須** のコンマ区切りリスト。hostAddresses を代わりに考慮するには、ip オプションおよび port オプションを空白のままにする必要があります。		文字列
indexName (producer)	動作させるインデックスの名前		文字列
indexType (producer)	動作させるインデックスのタイプ		文字列
maxRetryTimeout (producer)	再試行までの時間（ミリ秒単位）。	30000	int
操作 （プロデューサー）	実行する操作		ElasticsearchOperation
socketTimeout (producer)	ソケットがタイムアウトするまで待機するタイムアウト（ミリ秒単位）。	30000	int
waitForActiveShards (producer)	インデックス作成は、シャードの書き込みの整合性数が利用可能になるまで待機します。	1	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

92.3. メッセージ操作

現在、以下の ElasticSearch 操作がサポートされています。エンドポイント URI オプションまたはエクスチェンジヘッダーを「operation」のキーと、以下の値のいずれかに設定されるだけです。一部の操作では、他のパラメーターやメッセージボディーも設定する必要があります。

operation	メッセージボディー	説明
Index	**Map**,**String**, byte[] , XContentBuilder or IndexRequest content to index（文字列、文字列、**バイト[]**、**XContentBuilder**、または **IndexRequest** コンテンツを index にマップします）	インデックスにコンテンツを追加し、本文にコンテンツの indexId を返します。キー "indexId" でメッセージヘッダーを設定して indexId を設定できます。
GetById	取得するコンテンツの **文字列** または **GetRequest** インデックス ID	指定されたインデックスを取得し、本文の GetResult オブジェクトを返します。

operation	メッセージボディー	説明
Delete	**文字列** または **DeleteRequest** インデックス名と削除するコンテンツのタイプ	指定した indexName および indexType を削除し、本文に DeleteResponse オブジェクトを返します。
DeleteIndex	削除するインデックスの **文字列** または **DeleteRequest** インデックス名	指定された indexName を削除し、ボディーのステータスコードを返します。
BulkIndex	すでに許可されるタイプの **List**、**BulkRequest**、または **Collection**（XContentBuilder、Map、byte[]、String）	インデックスにコンテンツを追加し、本文で正常にインデックス化されたドキュメントの ID の一覧を返します。

operation	メッセージボディー	説明
一括	すでに許可されるタイプの **List**、**BulkRequest**、または **Collection**（XContentBuilder、Map、byte[]、String）	インデックスにコンテンツを追加し、本文に BulkItemResponse[] オブジェクトを返します。
Search	**マップ**、**文字列**、または **SearchRequest**	クエリー文字列のマップでコンテンツを検索します。
Exists	ヘッダーとしてのインデックス名 (indexName)	インデックスが存在するかチェックし、本文にブール値フラグを返します。

operation	メッセージボディー	説明
Update	**UpdateRequest**、**String**、**byte[]** または **XContentBuilder** コンテンツを **マップ** して更新する	インデックスにコンテンツを更新し、本文に含まれるコンテンツの indexId を返します。
Ping	なし	リモート Elasticsearch クラスターに ping を実行し、ping が成功した場合は true を返します。そうでない場合は false を返します。

92.4. コンポーネントの設定および BASIC 認証の有効化

Elasticsearch コンポーネントを使用するには、最低限の設定で設定する必要があります。

```
ElasticsearchComponent elasticsearchComponent = new ElasticsearchComponent();
elasticsearchComponent.setHostAddresses("myelkhost:9200");
camelContext.addComponent("elasticsearch-rest", elasticsearchComponent);
```

elasticsearch による Basic 認証、または elasticsearch クラスターの前にリバース http プロキシーを使用する場合、以下の例のようなコンポーネントで Basic 認証および SSL を設定するだけです。

```
ElasticsearchComponent elasticsearchComponent = new ElasticsearchComponent();
elasticsearchComponent.setHostAddresses("myelkhost:9200");
elasticsearchComponent.setUser("elkuser");
elasticsearchComponent.setPassword("secure!!");
elasticsearchComponent.setEnableSSL(true);

camelContext.addComponent("elasticsearch-rest", elasticsearchComponent);
```

92.5. インデックスの例

以下は簡単な INDEX の例です。

-

```
from("direct:index")
  .to("elasticsearch-rest://elasticsearch?
operation=Index&indexName=twitter&indexType=tweet");
```

```
<route>
    <from uri="direct:index" />
    <to uri="elasticsearch-rest://elasticsearch?
operation=Index&indexName=twitter&indexType=tweet"/>
</route>
```

クライアントは、マップを含むボディーメッセージをルートに渡すだけです。結果のボディーには、作成された indexId が含まれます。

```
Map<String, String> map = new HashMap<String, String>();
map.put("content", "test");
String indexId = template.requestBody("direct:index", map, String.class);
```

92.6. 検索例

特定のフィールドを検索し、値は Operation 'Search' を使用します。クエリー JSON 文字列またはマップを渡す

```
from("direct:search")
  .to("elasticsearch-rest://elasticsearch?
operation=Search&indexName=twitter&indexType=tweet");
```

```
<route>
    <from uri="direct:search" />
    <to uri="eelasticsearch-rest://elasticsearch?
operation=Search&indexName=twitter&indexType=tweet"/>
</route>
```

```
String query = "{\"query\":{\"match\":{\"content\":\"new release of ApacheCamel\"}}}";
SearchHits response = template.requestBody("direct:search", query, SearchHits.class);
```

Map を使用して特定のフィールドを検索します。

```
Map<String, Object> actualQuery = new HashMap<>();
actualQuery.put("content", "new release of ApacheCamel");

Map<String, Object> match = new HashMap<>();
match.put("match", actualQuery);

Map<String, Object> query = new HashMap<>();
query.put("query", match);
SearchHits response = template.requestBody("direct:search", query, SearchHits.class);
```

第93章 ELSQL コンポーネント

Camel バージョン 2.16 から利用可能

elsql: コンポーネントは、ElSql を使用して SQL クエリーを定義する既存の SQL コンポーネントの拡張です。

このコンポーネントは、実際の SQL 処理に背後で spring-jdbc を使用します。

このコンポーネントは、Transactional Client として使用できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-elsql</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

SQL コンポーネントは、以下のエンドポイント URI 表記を使用します。

```
sql:elSqlName:resourceUri[?options]
```

URI にクエリーオプションを追加するには、 ?option=value&option=value&…

SQL クエリーのパラメーターは、elsql マッピングファイルのパラメーターの名前が付けられ、特定の優先順位で Camel メッセージからの対応するキーにマップされます。

1. Camel 2.16.1: Simple 式の場合のメッセージボディーから。
2. メッセージヘッダーから 'java.util.Map'3 の場合、メッセージボディーから

名前付きパラメーターを解決できない場合、例外が発生します。

93.1. オプション

ElSQL コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
databaseVendor (common)	ベンダー固有の com.opengamma.elsql.ElSqlConfig を使用する。		ElSqlDatabaseVendor
dataSource (common)	データベースとの通信に使用する DataSource を設定します。		DataSource
elSqlConfig (advanced)	特定の設定済みの ElSqlConfig を使用します。代わりに databaseVendor オプションを使用することが推奨されます。		ElSqlConfig
resourceUri (common)	使用する elsql SQL ステートメントが含まれるリソースファイル。コンマで区切られた複数のリソースを指定できます。デフォルトでは、リソースはクラスパスに読み込まれます。前に file: 接頭辞をファイルシステムから読み込むことができます。このオプションをコンポーネントに設定し、エンドポイントで設定する必要がないことに注意してください。		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

ElSQL エンドポイントは、URI 構文を使用して設定します。

```
elsql:elsqlName:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

93.1.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
elsqlName	**必須**: 使用する elsql の名前（elsql ファイルの NAMED）。		文字列
resourceUri	使用する elsql SQL ステートメントが含まれるリソースファイル。コンマで区切られた複数のリソースを指定できます。デフォルトでは、リソースはクラスパスに読み込まれます。前に file: 接頭辞をファイルシステムから読み込むことができます。このオプションをコンポーネントに設定し、エンドポイントで設定する必要がないことに注意してください。		文字列

93.1.2. クエリーパラメーター（47 パラメーター）：

Name	説明	デフォルト	Type
allowNamedParameters (common)	クエリーで名前付きパラメーターを使用できるかどうか。	true	boolean
databaseVendor (common)	ベンダー固有の com.opengamma.elsql.ElSqlConfig を使用する。		ElSqlDatabaseVendor
dataSource (common)	データベースとの通信に使用する DataSource を設定します。		DataSource
dataSourceRef (common)	**非推奨**。データベースとの通信に使用する DataSource への参照をレジストリーから参照するように設定します。		文字列
outputClass (common)	outputType=SelectOne の変換として使用する完全なパッケージおよびクラス名を指定します。		文字列
outputHeader (common)	クエリー結果をメッセージのボディーの代わりにヘッダーに保存します。デフォルトでは、outputHeader == null とクエリー結果はメッセージボディーに保存され、メッセージボディー内の既存のコンテンツは破棄されます。outputHeader が設定されている場合、値はクエリー結果を保存するヘッダーの名前として使用され、元のメッセージボディーは保持されます。		文字列

Name	説明	デフォルト	Type
outputType (common)	コンシューマーまたはプロデューサーの出力を Map の List として選択するか、以下のように SelectOne を単一の Java オブジェクトとして選択します。クエリーに単一の列のみが含まれる場合、JDBC Column オブジェクトが返されます（SELECT COUNT（）FROM PROJECT は Long object.b を返します）、クエリーに複数の列がある場合、outputClass が設定されている場合は、その result.c の Map を返します。 次に、列名に一致するすべてのセッターを呼び出してクエリー結果を Java Bean オブジェクトに変換します。これは、クラスにインスタンスを作成するためのデフォルトのコンストラクターを想定します。クエリーが複数の行で作成された場合は、一意でない結果 exception.StreamList がイタレーターを使用してクエリーの結果をストリーミングします。これはストリーミングモードで Splitter EIP と共に使用して、ストリーミング方式で ResultSet を処理することができます。	SelectList	SqlOutputType
セパレーター（共通）	パラメーター値がメッセージボディー（本文が String タイプである場合）から取得されるときに使用する区切り文字はプレースホルダーに挿入されます。名前付きパラメーターを使用する場合は、代わりに Map タイプが使用されます。デフォルト値は comma です。	,	char
breakBatchOnConsumeFail (consumer)	消費に失敗した場合にバッチを壊すかどうかを設定します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
expectedUpdateCount (consumer)	onConsume の使用時に、予想される更新数を検証するように設定します。	-1	int
maxMessagesPerPoll (consumer)	ポーリングするメッセージの最大数を設定します。		int

Name	説明	デフォルト	Type
onConsume (consumer)	各行の処理後、Exchange が正常に処理されたときにこのクエリーを実行できます。たとえば、行を処理済みとしてマークします。クエリーにはパラメーターを指定できます。		文字列
onConsumeBatchComplete (consumer)	バッチ全体を処理したら、このクエリーを実行して行を一括更新することができます。クエリーにパラメーターを含めることはできません。		文字列
onConsumeFailed (consumer)	各行の処理後、エクスチェンジが失敗した場合、行の失敗をマークするなど、このクエリーを実行できます。クエリーにはパラメーターを指定できます。		文字列
routeEmptyResultSet (consumer)	空の結果セットを次のホップに送信できるようにするかどうかを設定します。デフォルトは false です。そのため、空の結果セットが除外されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
トランザクション (コンシューマー)	トランザクションを有効または無効にします。有効にすると、エクスチェンジの処理に失敗した場合、コンシューマーは追加のエクスチェンジを処理してロールバック Eager を生じさせます。	false	boolean
useIterator (consumer)	resultset をルートに配信する方法を設定します。配信をリストまたは個別オブジェクトのいずれかとして指定します。デフォルトは true です。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy

Name	説明	デフォルト	Type
processingStrategy (consumer)	コンシューマーが行/バッチを処理した場合に、プラグインでカスタムの org.apache.camel.component.sql.SqlProcessingStrategy を使用してクエリーを実行できます。		SqlProcessingStrategy
batch （プロデューサー）	バッチモードを有効または無効にします。	false	boolean
noop (producer)	設定されている場合、SQL クエリーの結果を無視し、既存の IN メッセージを処理の継続に OUT メッセージとして使用します。	false	boolean
useMessageBodyForSql (producer)	メッセージボディーを SQL として使用し、パラメーターのヘッダーとして使用するかどうか。このオプションを有効にすると、URI の SQL は使用されません。	false	boolean
alwaysPopulateStatement (producer)	有効にすると、org.apache.camel.component.sql.SqlPrepareStatementStrategy の populateStatement メソッドは常に呼び出されます。また、準備が予想されるパラメーターがない場合も常に呼び出されます。false の場合、1 つ以上の想定パラメーターが設定されている場合にのみ populateStatement が呼び出されます。たとえば、パラメーターを指定せずに SQL クエリーのメッセージボディー/ヘッダーを読み取ることが回避されます。	false	boolean
parametersCount (producer)	ゼロより大きい場合、Camel は JDBC メタデータ API 経由でクエリーを実行する代わりに、このパラメーター値を使用して置き換えるパラメーターの数を使用します。これは、JDBC ベンダーが正しいパラメーター数を返しないとユーザーが上書きできる場合に便利です。		int
elSqlConfig (advanced)	特定の設定済みの ElSqlConfig を使用します。代わりに databaseVendor オプションを使用することが推奨されます。		ElSqlConfig
placeholder (advanced)	SQL クエリーで置き換える文字を指定します。これは単純な String.replaceAll（）操作であり、SQL 解析が行われないことに注意してください（引用符付きの文字列も変更されます）。	#	文字列

Name	説明	デフォルト	Type
prepareStatementStrategy (advanced)	プラグインがカスタムの org.apache.camel.component.sql.SqlPrepareStatementStrategy を使用して、クエリーおよび準備済みステートメントの準備を制御できます。		SqlPrepareStatement Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
templateOptions (advanced)	マップのキー/値で Spring JdbcTemplate を設定します。		マップ
usePlaceholder (advanced)	プレースホルダーを使用するかどうかを設定し、すべてのプレースホルダー文字を SQL クエリーの署名に置き換えるかどうかを設定します。	true	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long

Name	説明	デフォルト	Type
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

93.2. クエリーの結果

選択 する操作の場合、結果は JdbcTemplate.queryFor List（）メソッドによって返される List<Map<String, Object >> タイプのインスタンスになります。更新 操作の場合、結果は更新された行数で、整数として返され ます。

デフォルトでは、結果はメッセージボディーに配置されます。 outputHeader パラメーターを設定すると、結果はヘッダーに配置されます。 これは、完全なメッセージ補完パターンを使用してヘッダーを追加する代わりに、シーケンスや他の小さな値をヘッダーにクエリーするための簡潔な構文を提供します。 outputHeader と outputType を一緒に使用すると便利です。

93.3. ヘッダーの値

更新 操作の実行時に、SQL コンポーネントは更新数を以下のメッセージヘッダーに保存します。

ヘッダー	説明
CamelSqlUpdateCount	Integer オブジェクトとして返される、**更新** 操作用に更新された行数。
CamelSqlRowCount	Integer オブジェクトとして返される、**選択** 操作のために返される行の数。

93.3.1. 例

以下の指定のルートでは、プロジェクトテーブルからすべてのプロジェクトを取得します。SQL クエリーにはパラメーター :#lic と :#min の 2 つの名前があることに留意してください。

*その後、Camel はメッセージボディーまたはメッセージヘッダーからこれらのパラメーターを検索します。上記の例では、名前付きパラメーターに定数値
を使用して 2 つのヘッダーを設定しています。*

```
from("direct:projects")
  .setHeader("lic", constant("ASF"))
  .setHeader("min", constant(123))
  .to("elsql:projects:com/foo/orders.elsql")
```

elsql マッピングファイル

```
@NAME(projects)
  SELECT *
  FROM projects
  WHERE license = :lic AND id > :min
  ORDER BY id
```

メッセージボディーが java.util.Map の場合、名前付きパラメーターはボディーから取得されます。

```
from("direct:projects")
  .to("elsql:projects:com/foo/orders.elsql")
```

Camel 2.16.1 以降では、Simple 式も使用できます。これにより、メッセージボディーの OGNL のような表記を使用できます。ここでは、 getLicense および getMinimum メソッドがあることを前提と

しています。

```
@NAME(projects)
  SELECT *
  FROM projects
  WHERE license = :${body.license} AND id > :${body.minimum}
  ORDER BY id
```

93.4. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*
- *SQL コンポーネント*
- *MyBatis*
- *JDBC*

第94章 ETCD コンポーネント

Camel バージョン 2.18 から利用可能

Camel etcd コンポーネントを使用すると、分散型の信頼できるキーと値ストアである Etcd と連携できます。

94.1. URI 形式

```
etcd:namespace/path[?options]
```

94.2. URI オプション

etcd コンポーネントは、以下に示す 7 つのオプションをサポートします。

Name	説明	デフォルト	Type
URI （共通）	クライアントが接続する URI を設定します。		文字列
sslContextParameters (common)	SSLContextParameters を使用してセキュリティーを設定します。		SSLContextParameters
userName (common)	Basic 認証に使用するユーザー名。		文字列
パスワード (common)	Basic 認証に使用するパスワード。		文字列
Configuration (advanced)	エンドポイント間で共有される共通設定を設定します。		EtcdConfiguration
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

etcd エンドポイントは URI 構文を使用します。

```
etcd:namespace/path
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

94.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
namespace	使用する API namespace が **必要です**。		EtcdNamespace
path	エンドポイントが参照するパス		文字列

94.2.2. クエリーパラメーター（29 パラメーター）：

Name	説明	デフォルト	Type
再帰 的(common)	アクションを再帰的に適用します。	false	boolean
servicePath (common)	サービス検出を検索するパス	/services/	文字列
タイムアウト (common)	アクションの完了までにかかる時間を設定します。		Long
URI （共通）	クライアントが接続する URI を設定します。	http://localhost:2379,http://localhost:4001	文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
sendEmptyExchangeOnTimeout (consumer)	キーの監視のタイムアウトが発生した場合に空のメッセージを送信する。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
fromIndex (consumer)	監視するインデックス	0	Long
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
timeToLive (producer)	キーの有効期間をミリ秒単位で設定します。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int

Name	説明	デフォルト	Type
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit

Name	説明	デフォルト	Type
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
パスワード （セキュリティー）	Basic 認証に使用するパスワード。		文字列
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定します。		SSLContextParameters
userName (security)	Basic 認証に使用するユーザー名。		文字列

第95章 OSGI EVENTADMIN COMPONENT

Camel バージョン 2.6 で利用可能

eventadmin コンポーネントは、OSGi 環境で使用することで、OSGi EventAdmin イベントを受信して処理できます。

95.1. 依存関係

Maven ユーザーは以下の依存関係を pom.xmlに追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-eventadmin</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel（2.6.0 以降）の実際のバージョンに置き換える必要があります。

95.2. URI 形式

```
eventadmin:topic[?options]
```

topic はリッスンするトピックの名前です。

95.3. URI オプション

OSGi EventAdmin コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
bundleContext (common)	OSGi BundleContext は Camel によって自動的にインジェクトされます。		BundleContext

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

OSGi EventAdmin エンドポイントは、URI 構文を使用して設定します。

```
eventadmin:topic
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

95.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
topic	リッスンまたは送信するトピックの名前		文字列

95.3.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
送信 （共通）	'send' または 'synchronous' 配信を使用するかどうか。デフォルトの false（非同期配信）	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

95.4. メッセージヘッダー

Name	Type	メッセージ
説明		

95.5. メッセージボディー

in メッセージボディーは受信したイベントに設定されます。

95.6. 使用例

```
<route>
  <from uri="eventadmin:*"/>
  <to uri="stream:out"/>
</route>
```

第96章 EXEC コンポーネント

Camel バージョン 2.3 の時点で利用可能

exec コンポーネントはシステムコマンドを実行するために使用できます。

96.1. 依存関係

Maven ユーザーは以下の依存関係を pom.xmlに追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-exec</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョン（2.3.0 以降）に置き換える必要があります。

96.2. URI 形式

```
exec://executable[?options]
```

executable は、実行されるシステムコマンドの名前またはファイルパスに置き換えます。実行可能ファイル名が使用されている場合（例： exec:java）、実行ファイルはシステムパス内で実行する必要があります。

96.3. URI オプション

Exec コンポーネントにはオプションがありません。

Exec エンドポイントは、URI 構文を使用して設定します。

```
exec:executable
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

96.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
executable	**必須**。実行する実行ファイルを設定します。実行可能ファイルは空または null にすることはできません。		文字列

96.3.2. クエリーパラメーター（8 パラメーター）：

Name	説明	デフォルト	Type
引数 （プロデューサー）	引数は、空白文字で区切ったトークンを 1 つまたは複数指定できます。		文字列
バインディング （プロデューサー）	レジストリーの org.apache.commons.exec.ExecBinding への参照。		ExecBinding
commandExecutor (producer)	コマンド実行をカスタマイズするレジストリーの org.apache.commons.exec.ExecCommandExecutor への参照。デフォルトのコマンドエグゼキューターは commons-exec ライブラリーを使用して、実行したすべてのコマンドに対してシャットダウンフックを追加します。		ExecCommandExecutor
outFile (producer)	実行可能ファイルで作成されるファイルの名前。これは出力として考慮する必要があります。outFile が設定されていない場合、実行ファイルの標準出力(stdout)が使用されます。		文字列
タイムアウト （プロデューサー）	実行を終了するタイムアウト（ミリ秒単位）。タイムアウト内で実行が完了しなかった場合、コンポーネントは終了リクエストを送信します。		Long
useStderrOnEmptyStdout (producer)	stdout が空の場合、このコンポーネントは Camel Message Body に標準エラー(stderr)が入力されることを示すブール値。この動作は、デフォルトでは無効(false)です。	false	boolean
workingDir (producer)	コマンドを実行するディレクトリー。null の場合、現在のプロセスの作業ディレクトリーが使用されます。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

96.4. メッセージヘッダー

サポートされるヘッダーは org.apache.camel.component.exec.ExecBinding で定義されます。

Name	Type	メッセージ	説明
ExecBinding.EXEC_COMMAND_EXECUTABLE	**文字列**	**in**	実行される system コマンドの名前。URI で **実行可能な** 内容を上書きします。
ExecBinding.EXEC_COMMAND_ARGS	**java.util.List<String>**	**in**	実行中のプロセスに渡すコマンドライン引数引数は文字通り使用され、引用は適用されません。URI の既存の **引数** を上書きします。
ExecBinding.EXEC_COMMAND_ARGS	**文字列**	**in**	Camel 2.5: 各引数が空白で区切られた単一文字列として実行ファイルの引数（URI オプション **の** 引数 を参照）。引数は文字通り使用され、引用は適用されません。URI の既存の **引数** を上書きします。
ExecBinding.EXEC_COMMAND_OUT_FILE	**文字列**	**in**	実行可能ファイルで作成されるファイルの名前。これは出力として考慮する必要があります。URI の既存の **outFile** を上書きします。
ExecBinding.EXEC_COMMAND_TIMEOUT	**Long**	**in**	実行を終了するタイムアウト（ミリ秒単位）。URI の既存の **タイムアウト** を上書きします。

Name	Type	メッセージ	説明
ExecBinding.EXEC_COMMAND_WORKING_DIR	**文字列**	**in**	コマンドを実行するディレクトリー。URI の既存の **workingDir** を上書きします。
ExecBinding.EXEC_EXIT_VALUE	**int**	**out**	このヘッダーの値は、実行可能ファイルの **終了値** です。通常、ゼロ以外の終了値は通常、異常な終了を示します。exit 値は OS に依存することに注意してください。
ExecBinding.EXEC_STDERR	**java.io.InputStream**	**out**	このヘッダーの値は、実行ファイルの標準エラーストリーム(stderr)を参照します。標準エラー(stderr)が書き込まれない場合、値は **null** になります。
ExecBinding.EXEC_USE_STDERR_ON_EMPTY_STDOUT	**boolean**	**in**	**stdout** が空の場合、このコンポーネントは Camel Message Body に標準エラー(**stderr**)が入力されることを示します。この動作は、デフォルトでは無効(**false**)です。

96.5. メッセージボディー

Exec コンポーネントが、java.io.InputStream に変換するメッセージボディー で 受信された場合、stdin を介して実行ファイルに入力をフィードするために使用されます。実行後、メッセージボディー は実行結果になります。つまり、stdout、stderr、終了値、および out ファイルが含まれる org.apache.camel.components.exec.ExecResult インスタンスになります。このコンポーネントは、便宜上、以下の ExecResult 型コンバーター をサポートします。

From	終了
ExecResult	**java.io.InputStream**

From	終了
ExecResult	文字列
ExecResult	byte []
ExecResult	org.w3c.dom.Document

out ファイル（ outFile 経由のエンドポイントまたは ExecBinding.EXEC_COMMAND_OUT_FILE経由のメッセージヘッダー）が指定されている場合、コンバーターは out ファイルの内容を返します。ファイルが使用されていない場合、このコンポーネントはプロセスの標準出力をターゲットタイプに変換します。詳細は、以下の使用例を参照 して ください。

96.6. 使用例

96.6.1. 単語数(Linux)の実行

以下の例では、wc （キーワード数、Linux）を実行して、/usr/share/dict/words ファイルの単語をカウントします。単語 count（出力）は、wc の標準出力ストリームに書き込まれます。

```
from("direct:exec")
.to("exec:wc?args=--words /usr/share/dict/words")
.process(new Processor() {
    public void process(Exchange exchange) throws Exception {
      // By default, the body is ExecResult instance
      assertIsInstanceOf(ExecResult.class, exchange.getIn().getBody());
      // Use the Camel Exec String type converter to convert the ExecResult to String
      // In this case, the stdout is considered as output
      String wordCountOutput = exchange.getIn().getBody(String.class);
      // do something with the word count
    }
});
```

96.6.2. Java の 実行

以下の例は、java がシステムパスにある場合に、2 つの引数 -server および -version を使用して java を実行します。

```
from("direct:exec")
.to("exec:java?args=-server -version")
```

以下の例は、引数 -server、-version、sytem プロパティー user.name で、c:\temp で java を実行します。

```
from("direct:exec")
.to("exec:c:/program files/jdk/bin/java?args=-server -version -
Duser.name=Camel&workingDir=c:/temp")
```

96.6.3. アンチスクリプトの実行

以下の例は、ant.bat がシステムパスにあり、CamelExecBuildFile.xml が現在のディレクトリーにある場合に、ビルドファイル CamelExecBuildFile.xml で Apache Ant （Windows のみ）を実行します。

```
from("direct:exec")
.to("exec:ant.bat?args=-f CamelExecBuildFile.xml")
```

以下の例では、ant.bat コマンドは出力を -l で CamelExecOutFile.txt にリダイレクトします。CamelExecOutFile.txt ファイルは out ファイルとして使用され ます。この例では、ant.bat がシステムパスにあり、CamelExecBuildFile.xml が現在のディレクトリーにあることを前提としています。

```
from("direct:exec")
.to("exec:ant.bat?args=-f CamelExecBuildFile.xml -l
CamelExecOutFile.txt&outFile=CamelExecOutFile.txt")
.process(new Processor() {
    public void process(Exchange exchange) throws Exception {
      InputStream outFile = exchange.getIn().getBody(InputStream.class);
      assertIsInstanceOf(InputStream.class, outFile);
      // do something with the out file here
    }
  });
```

96.6.4. echo (Windows)の実行

echo や dir などのコマンドは、オペレーティングシステムのコマンドインタープリターからしか実行できません。この例は、Windows でこのようなコマンド echo - の実行方法を示しています。

```
from("direct:exec").to("exec:cmd?args=/C echo echoString")
```

96.7. 関連項目

- Configuring Camel (Camel の設定)

- *コンポーネント*
- *エンドポイント*
- *はじめに*

第97章 FACEBOOK コンポーネント

Camel バージョン 2.14 から利用可能

Facebook コンポーネントは、Facebook4J でアクセス可能なすべての Facebook API へのアクセスを提供します。これにより、メッセージを生成して、コメント、写真、アルバム、ビデオ、写真、チェック、場所、リンクなどの投稿を取得、追加、削除することができます。また、投稿、ユーザー、チェック、グループ、場所などのポーリングを可能にする API もサポートしています。

Facebook には、すべてのクライアントアプリケーション認証で OAuth を使用する必要があります。アカウントで camel-facebook を使用するには、Facebook で https://developers.facebook.com/apps で新しいアプリケーションを作成し、アプリケーションがアカウントにアクセスできるようにする必要があります。Facebook アプリケーションの ID およびシークレットは、現行ユーザーを必要としない Facebook API へのアクセスを許可します。ログインユーザーを必要とする API には、ユーザーアクセストークンが必要です。ユーザーのアクセストークンの取得に関する詳細は、https://developers.facebook.com/docs/facebook-login/access-tokens/ を参照してください。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-facebook</artifactId>
  <version>${camel-version}</version>
</dependency>
```

97.1. URI 形式

```
facebook://[endpoint]?[options]
```

97.2. FACEBOOKCOMPONENT

facebook コンポーネントは、以下の Facebook アカウント設定で設定できます。これは必須です。値は、org.apache.camel.component.facebook.config.FacebookConfiguration タイプの bean プロパティー 設定 を使用してコンポーネントに指定できます。oAuthAccessToken オプションは省略できますが、アプリケーション API へのアクセスのみを許可します。

Facebook コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有設定の使用		FacebookConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Facebook エンドポイントは URI 構文を使用して設定します。

```
facebook:methodName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

97.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
methodName	***必要な*** 操作		文字列

97.2.2. クエリーパラメーター（102 パラメーター）：

Name	説明	デフォルト	Type
achievementURL (common)	達成の一意の URL		URL
albumId (common)	アルバム ID		文字列
albumUpdate (common)	作成または更新する facebook Album		AlbumUpdate
appId (common)	Facebook アプリケーションの ID		文字列
中央 （共通）	Location latitude および longitude		GeoLocation

Name	説明	デフォルト	Type
checkinId (common)	チェックイン ID		文字列
checkinUpdate (common)	**非推奨**: 作成されるチェック。代わりに、ロケーションがアタッチされた Post を作成。		CheckinUpdate
Clienturl （共通）	Facebook4J API クライアント URL		文字列
clientVersion (common)	Facebook4J クライアント API バージョン		文字列
commentId (common)	コメント ID		文字列
commentUpdate (common)	作成または更新するファンブックコメント		CommentUpdate
debugEnabled (common)	deubg 出力を有効にします。埋め込みロガーでのみ有効	false	ブール値
説明 （共通）	description テキスト		文字列
距離 （共通）	メーター内の距離		整数
domainId (common)	ドメイン ID		文字列
domainName (common)	ドメイン名		文字列
domainNames (common)	ドメイン名		リスト
eventId (common)	イベント ID		文字列
eventUpdate (common)	作成または更新するイベント		EventUpdate
friendId (common)	ファリエンド ID		文字列
friendlistId (common)	ファリエンドリスト ID		文字列
friendlistName (common)	平易なリスト名		文字列

Name	説明	デフォルト	Type
friendUserId (common)	friend ユーザー ID		文字列
groupId (common)	グループ ID		文字列
gzipEnabled (common)	Facebook GZIP エンコーディングの使用	true	ブール値
httpConnectionTimeout (common)	HTTP 接続のタイムアウト（ミリ秒単位）	20000	整数
httpDefaultMaxPerRoute (common)	ルートごとの HTTP 最大接続	2	整数
httpMaxTotalConnections (common)	HTTP 最大接続合計	20	整数
httpReadTimeout (common)	HTTP の読み取りタイムアウト（ミリ秒単位）	120000	整数
httpRetryCount (common)	HTTP 再試行の数	0	整数
httpRetryIntervalSeconds (common)	HTTP 再試行間隔（秒単位）	5	整数
httpStreamingReadTimeout (common)	HTTP ストリーミングの読み取りタイムアウト（ミリ秒単位）	40000	整数
ids (common)	ユーザーの ID		リスト
inBody (common)	エクスチェンジ In Body で渡されるパラメーターの名前を設定します。		文字列
includeRead (common)	読み取らない通知に加えて、ユーザーがすでに読み込まれていることを示す通知を有効にします。		ブール値
isHidden (common)	非表示にするかどうか		ブール値
jsonStoreEnabled (common)	true に設定すると、未加工の JSON フォームは DataObjectFactory に保存されます。	false	ブール値

Name	説明	デフォルト	Type
リンク (common)	リンク URL		URL
linkId (common)	リンク ID		文字列
locale (common)	希望する FQL ロケール		Locale
mbeanEnabled (common)	true に設定すると、Facebook4J mbean が登録されます。	false	ブール値
Message (common)	メッセージテキスト		文字列
messageId (common)	メッセージ ID		文字列
メトリクス (common)	メトリクス名		文字列
milestoneId (common)	マイルストーン ID		文字列
名前 (common)	テストユーザー名。'first last' の形式にする必要があります		文字列
noteId (common)	注記 ID		文字列
notificationId (common)	通知 ID		文字列
objectId (common)	insights オブジェクト ID		文字列
offerId (common)	提供する ID		文字列
optionDescription (common)	質問の回答オプションの説明		文字列
pageId (common)	ページ ID		文字列
permissionName (common)	パーミッション名		文字列

Name	説明	デフォルト	Type
パーミッション (common)	perm1,perm2,... 形式でユーザーパーミッションをテストします。		文字列
photoId (common)	写真 ID		文字列
pictureId (common)	図 ID		整数
pictureId2 (common)	picture2 id		整数
pictureSize (common)	図のサイズ		PictureSize
placeId (common)	場所 ID		文字列
postId (common)	POST ID		文字列
postUpdate (common)	作成または更新する POST		PostUpdate
prettyDebugEnabled (common)	true に設定されている場合に JSON デバッグ出力を Prettify します。	false	ブール値
クエリー （共通）	FQL クエリー		マップ
クエリー （共通）	FQL クエリーまたは検索エンドポイントの検索用語		文字列
questionId (common)	質問 ID		文字列
読み取り （共通）	パラメーターの読み取り（オプション）。「reading Options(reading)」を参照してください。		読み取り
readingOptions (common)	マップからキーと値のペアを使用して読み取りを設定するには、以下を行います。		マップ
restBaseURL (common)	API ベース URL	https://graph.facebook.com/	文字列
scoreValue (common)	値を持つ数値スコア		整数

Name	説明	デフォルト	Type
size (common)	図のサイズ、大、標準、小、または角		PictureSize
ソース (common)	java.io.File または java.io.Inputstream のいずれかのメディアコンテンツ。		メディア
Subject (common)	サブジェクトの注記		文字列
tabId (common)	タブ ID		文字列
tagUpdate (common)	写真タグの情報		TagUpdate
testUser1 (common)	ユーザー 1 のテスト		TestUser
testUser2 (common)	ユーザー 2 のテスト		TestUser
testUserId (common)	テストユーザーの ID		文字列
title (common)	タイトルテキスト		文字列
toUserId (common)	タグを付けるユーザーの ID		文字列
toUserIds (common)	タグ付けするユーザーの ID		リスト
userId (common)	Facebook のユーザー ID		文字列
userId1 (common)	ユーザー 1 の ID		文字列
userId2 (common)	ユーザー 2 の ID		文字列
userIds (common)	イベントに招待するユーザーの ID		リスト
userLocale (common)	テストユーザーロケール		文字列
useSSL (common)	SSL の使用	true	ブール値

Name	説明	デフォルト	Type
videoBaseURL (common)	ビデオ API のベース URL	https://graph-video.facebook.com/	文字列
videoId （共通）	ビデオ ID		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
httpProxyHost (proxy)	HTTP プロキシーサーバーのホスト名		文字列
httpProxyPassword (proxy)	HTTP プロキシーサーバーのパスワード		文字列
httpProxyPort (proxy)	HTTP プロキシーサーバーポート		整数
httpProxyUser (proxy)	HTTP プロキシーサーバーのユーザー名		文字列
oAuthAccessToken (security)	ユーザーのアクセストークン		文字列

Name	説明	デフォルト	Type
oAuthAccessTokenURL (security)	OAuth アクセストークン URL	https://graph.facebook.com/oauth/access_token	文字列
oAuthAppId (security)	アプリケーション ID		文字列
oAuthAppSecret (security)	アプリケーションシークレット		文字列
oAuthAuthorizationURL (security)	OAuth 認証 URL	https://www.facebook.com/dialog/oauth	文字列
oAuthPermissions (security)	デフォルトの OAuth パーミッション。カンマ区切りのパーミッション名。詳細は、https://developers.facebook.com/docs/reference/login/permissions を参照してください。		文字列

97.3. プロデューサーエンドポイント：

プロデューサーエンドポイントは、以下の表のエンドポイント名とオプションを使用できます。エンドポイントは、getCheckin と searchCheckin 間の曖昧さによる チェックを除き、get または search prefix なしで 短縮名を使用することもできます。必須ではないエンドポイントオプションは [] で表されます。

プロデューサーエンドポイントは、特別なオプション inBody を使用することもできます。そのオプションには、値が Camel Exchange In メッセージに含まれる endpoint オプションの名前が含まれる必要があります。たとえば、以下のルートの facebook エンドポイントは、受信メッセージのボディーでユーザー ID の値のアクティビティーを取得します。

```
from("direct:test").to("facebook://activities?inBody=userId")...
```

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は CamelFacebook の形式である必要があります。
https://cwiki.apache.org/confluence/pages/createpage.action?

spaceKey=CAMEL&title=option&linkCreation=true&fromPageId=34020899[option].たとえば、前のルートの userId オプションの値は、メッセージヘッダー CamelFacebook.userId で提供することもできます。inBody オプションはメッセージヘッダーを上書きすることに注意してください。たとえば、Body=user のエンドポイントオプションは CamelFacebook.userId ヘッダーを上書きすることに注意してください。

String を返すエンドポイントは、作成または変更したエンティティーの ID を返します。たとえば、addAlbumPhoto は新しい album Id を返します。ブール値を返すエンドポイント。成功の場合は true、それ以外の場合は false を返します。Facebook API エラーが出ると、エンドポイントは facebook4j.FacebookException 原因で RuntimeCamelException を発生させます。

97.4. コンシューマーエンドポイント：

read #reading パラメーターを取るプロデューサーエンドポイントのいずれかは、コンシューマーエンドポイントとして使用できます。ポーリングコンシューマーは since および until フィールドを使用してポーリング間隔内の応答を取得します。他の読み取りフィールドの他に、最初の ポーリング 値も最初のポーリングのエンドポイントに指定できます。

単一のルートエクスチェンジを介して List（または facebook4j.ResponseList）を返すエンドポイントではなく、camel-facebook は返されたオブジェクトごとにルートエクスチェンジを 1 つ作成します。たとえば、"facebook://home" が 5 回すると、ルートは 5 回実行されます（Post ごとに 1 回）。

97.5. オプションの読み取り

タイプが facebook4j.Reading の 読み取り オプションは、パラメーターの読み取りサポートを追加します。これにより、特定のフィールドを選択でき、結果の数を制限することができます。詳細は、Graph API#reading - Facebook Developers を参照してください。

また、ポーリング間で重複メッセージを送信しないように、Facebook データをポーリングするためにコンシューマーエンドポイントでも使用されます。

読み取りオプションは、Facebook 4j.Reading 型の参照または値か、エンドポイント URI または CamelFacebook のあるエクスチェンジヘッダーのいずれかで以下の読み取りオプションを使用して指定できます。

97.6. メッセージヘッダー

URI options#urioptions は、プロデューサーエンドポイントのメッセージヘッダーに CamelFacebook. プレフィックスを付けて指定できます。

97.7. メッセージボディー

すべての結果メッセージ本文は、Facebook4J API によって提供されるオブジェクトを使用します。プロデューサーエンドポイントは、inBody エンドポイントパラメーターに受信メッセージボディーのオプション名を指定できます。

アレイまたは facebook4j.ResponseList または java.util.List を返すエンドポイントの場合、コンシューマーエンドポイントはリスト内のすべての要素を異なるメッセージにマッピングします。

97.8. ユースケース

Facebook プロファイル内に POST を作成するには、このプロデューサーを facebook4j.PostUpdate ボディーに送信します。

```
    from("direct:foo")
       .to("facebook://postFeed/inBody=postUpdate);
```

ポーリングするには 5 つのすべてのセクション（ポーリングコンシューマー のオプションを設定するには、接頭辞「consumer」）と、ホームフィードのすべてのステータスを設定できます。

```
    from("facebook://home?consumer.delay=5000")
       .to("bean:blah");
```

ヘッダーの動的オプションでプロデューサーを使用した検索。

バーヘッダーには、公開投稿で実行する Facebook 検索文字列があるため、この値を CamelFacebook.query ヘッダーに割り当てる必要があります。

```
    from("direct:foo")
       .setHeader("CamelFacebook.query", header("bar"))
       .to("facebook://posts");
```

第98章 FHIR JSON DATAFORMAT

Camel バージョン 2.21 available as of Camel version 2.21 Available as of Camel version 2.21（Camel バージョン 2.21 の Camel バージョン 2.21 から利用可能）

FHIR-JSON Data Format は HAPI-FHIR の JSON パーサーを活用し、HAPI-FHIR の IBaseResource へ/から解析します。

98.1. FHIR JSON 形式のオプション

FHIR JSon データフォーマットは、以下に示す 2 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
fhirVersion	**DSTU3**	**文字列**	使用する FHIR のバージョン。使用できる値は DSTU2、DSTU2_HL7ORG、DSTU2_1,DSTU3,R4 です。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

第99章 FHIR XML DATAFORMAT

Camel バージョン 2.21 で利用 可能： Camel バージョン 2.21

FHIR-XML Data Format は HAPI-FHIR の XML パーサーを活用して、HAPI-FHIR の IBaseResource へ/から XML 形式から解析します。

99.1. FHIR XML 形式のオプション

FHIR XML データフォーマットは、以下に示す 2 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
fhirVersion	**DSTU3**	**文字列**	使用する FHIR のバージョン。使用できる値は DSTU2、DSTU2_HL7ORG、DSTU2_1,DSTU3,R4 です。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

第100章 ファイルコンポーネント

Camel バージョン 1.0 で利用可能

File コンポーネントはファイルシステムへのアクセスを提供し、他の Camel コンポーネントからのメッセージや、他のコンポーネントからのメッセージをディスクに処理できるようにします。

100.1. URI 形式

```
file:directoryName[?options]
```

または

```
file://directoryName[?options]
```

directoryName は基礎となるファイルディレクトリーを表します。

URI にクエリーオプションを追加するには、?option=value&option=value&…

ディレクトリーのみ

Camel は、開始ディレクトリーで設定されたエンドポイントのみをサポートします。そのため、directoryName はディレクトリーである必要があります。
単一ファイルのみを使用する場合は、fileName オプションを使用できます（例：fileName=thefilename ）。
また、開始ディレクトリーには $\{ } プレースホルダーを持つ動的式を含めることはできません。ここでも、fileName オプションを使用してファイル名の動的部分を指定します。

警告

別の アプリケーションで現在書き込まれているファイルを読み込まないように、別のアプリケーション が現在ファイルを書き込み/コピーしているかどうかを検出する場合には、JDK ファイル IO API が少し制限されています。実装は、OS プラットフォームによっても異なる場合があります。これにより、Camel が別のプロセスでロックされていないと判断し、消費を開始できる可能性があります。したがって、お使いの環境スイートを詳しく調べる必要があります。この Camel では、使用できるさまざまな readLock オプションと doneFileName オプションを利用できます。「他のファイルが直接ドロップされるフォルダーからのファイルの使用 」セクションを参照してください。

100.2. URI オプション

File コンポーネントにはオプションがありません。

File エンドポイントは、URI 構文を使用して設定します。

```
file:directoryName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

100.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
directoryName	**必要な** 起動ディレクトリー		ファイル

100.2.2. クエリーパラメーター（81 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
charset (common)	このオプションは、ファイルのエンコーディングを指定するために使用されます。これはコンシューマーで使用して、ファイルのエンコーディングを指定できます。これにより、Camel はファイルの内容にアクセスした場合にファイルの内容を読み込む必要があります。ファイルを書き込む場合も同様に、このオプションを使用して、ファイルを書き込む文字を指定することもできます。ファイルを書くときには、メッセージの内容をメモリーに読み込んで、データを設定済みの文字セットに変換しなければならない場合があります。そのため、大きなメッセージがある場合は使用しないでください。		文字列
doneFileName (common)	プロデューサー： 指定された場合、元のファイルが書き込まれると、Camel は 2 番目に完了したファイルを書き込みます。完了ファイルは空になります。このオプションは、使用するファイル名を設定します。固定名を指定できます。または、動的プレースホルダーを使用することもできます。done ファイルは、常に元のファイルと同じフォルダーに書き込まれます。コンシューマー： 提供された場合、Camel は完了したファイルが存在する場合にのみファイルを消費します。このオプションは、使用するファイル名を設定します。固定名を指定できます。または、動的プレースホルダーを使用することもできます。完成したファイルは、常に元のファイルと同じフォルダーに期待されます。$file.name および $file.name.noext のみが動的プレースホルダーとしてサポートされます。		文字列

Name	説明	デフォルト	Type
fileName (common)	File Language などの式を使用して、ファイル名を動的に設定します。コンシューマーの場合は、ファイル名フィルターとして使用されます。プロデューサーの場合、書き込みするファイル名を評価するために使用されます。式が設定されている場合は、CamelFileName ヘッダーよりも優先されます。（注記： ヘッダー自体は式にすることもできます。式オプションは String タイプと Expression 型の両方をサポートします。式が String タイプである場合、これは常に File 言語を使用して評価されます。式が Expression タイプである場合、指定された式タイプが使用されます。たとえば、OGNL 式を使用できます。コンシューマーの場合は、これを使用してファイル名を絞り込むことができます。そのため、インスタンスは File Language 構文 mydata-$date:yyyyMMdd.txt を使用して現在のファイルを使用できます。プロデューサーは、既存の CamelFileName ヘッダーよりも優先される CamelOverruleFileName ヘッダーをサポートします。CamelOverruleFileName は 1 度だけ使用されるヘッダーで、CamelFileName を一時的に保存し、後で復元する必要があるため、より簡単に使用できます。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
Delete （コンシューマー）	true の場合、ファイルは正常に処理された後に削除されます。	false	boolean
moveFailed (consumer)	Simple 言語に基づいて移動失敗式を設定します。たとえば、.error サブディレクトリーにファイルを移動するには、.error を使用します。注記： 失敗した場所にファイルを移動すると Camel はエラーを処理し、再度ファイルを取得しません。		文字列
noop （コンシューマー）	true の場合、ファイルは移動または削除されません。このオプションは、読み取り専用データまたは ETL タイプの要件に適しています。noop=true の場合、Camel はべき等=true も設定し、同じファイルを何度も消費しないようにします。	false	boolean

Name	説明	デフォルト	Type
preMove (consumer)	処理前にファイル名を動的に設定するために使用される式（ファイル言語など）。たとえば、進行中のファイルを order ディレクトリーに移動するには、この値を order に設定します。		文字列
preSort (consumer)	pre-sort を有効にすると、コンシューマーはポーリング中にファイルとディレクトリー名をソートし、ファイルシステムから取得されたものになります。これは、ソートされた順序でファイルで操作する必要がある場合があります。pre-sort は、コンシューマーがフィルターをかけ、Camel が処理するファイルを受け入れる前に実行されます。このオプションは、無効を意味する default=false です。	false	boolean
再帰的 （コンシューマー）	ディレクトリーの場合は、すべてのサブディレクトリー内のファイルも検索します。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
directoryMustExist (consumer)	startingDirectoryMustExist と同様ですが、再帰的なサブディレクトリーのポーリング時に適用されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
extendedAttributes (consumer)	対象のファイル属性を定義します。posix:permissions,posix:owner,basic:lastAccessTime と同様に、posix:、basic:lastAccessTime などの基本的なワイルドカードをサポートします。		文字列
inProgressRepository (consumer)	プラグ可能な in-progress リポジトリー org.apache.camel.spi.IdempotentRepository。進行中のリポジトリーは、現在使用中の進行中のファイルを考慮するために使用されます。デフォルトでは、メモリーベースのリポジトリーが使用されます。		String>

Name	説明	デフォルト	Type
localWorkDirectory (consumer)	使用する場合は、ローカルの作業ディレクトリーを使用してリモートファイルの内容を直接ローカルファイルに保存し、コンテンツをメモリーにロードしないようにできます。これは、非常に大きなリモートファイルを使用しているため、メモリーを節約できることに有益です。		文字列
onCompletionException Handler (consumer)	カスタムの org.apache.camel.spi.ExceptionHandler を使用して、コンシューマーがコミットまたはロールバックを行う完了プロセス時に発生する例外を処理します。デフォルトの実装は、WARN レベルですべての例外をログに記録し、無視されます。		ExceptionHandler
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。つまり、ポーリングによって情報の収集中にエラーが発生しました。たとえば、ファイルネットワークへのアクセスに失敗したため、Camel はファイルスキャンのためにアクセスできません。デフォルトの実装では、WARN レベルで原因となる例外がログに記録され、無視されます。		PollingConsumerPoll Strategy
probeContentType (consumer)	コンテンツタイプのプローブを有効にするかどうか。有効にすると、コンシューマーは link FilesprobeContentType(java.nio.file.Path)を使用してファイルのコンテンツタイプを決定し、メッセージ上に ExchangeFILE_CONTENT_TYPE キーリンクのあるヘッダーとして保存します。	false	boolean
processStrategy (consumer)	プラグ可能な org.apache.camel.component.file.GenericFileProcessStrategy を使用すると、独自の readLock オプションまたは同様のものを実装できます。また、特別な対応ファイルが存在するなど、ファイルを使用する前に特別な条件が満たされなければならない場合にも使用できます。このオプションを設定すると、readLock オプションが適用されません。		GenericFileProcess s Strategy<T>
startingDirectoryMustExist (consumer)	開始ディレクトリーが存在するかどうか。autoCreate オプションはデフォルトで有効になっています。つまり、開始ディレクトリーが存在しない場合には、通常、起動ディレクトリーが作成されます。autoCreate を無効にして、これを有効にして、開始ディレクトリーが存在することを確認できます。ディレクトリーが存在しない場合は例外が発生します。	false	boolean

Name	説明	デフォルト	Type
fileExist (producer)	同じ名前のファイルがすでに存在する場合のアクション。デフォルトのオーバーライドは、既存のファイルを置き換えます。append: 既存のファイルにコンテンツを追加します。fail: GenericFileOperationException をスローし、既存のファイルがあることを示します。ignore: 問題を警告なしで無視し、既存のファイルは上書きしませんが、すべてが問題であることを前提としています。move - オプションでは、moveExisting オプションも設定する必要があります。eagerDeleteTargetFile オプションを使用して、ファイルの移動や既存のファイルがすでに存在する場合に実行内容を制御できます。それ以外の場合は、移動操作が失敗します。Move オプションは、ターゲットファイルを書き込む前に既存のファイルを移動します。TryRename は、tempFileName オプションが使用されている場合にのみ適用されます。これにより、存在チェックを実行せずに、一時的な名前から実際の名前へのファイルの名前変更を試すことができます。一部のファイルシステムや、特に FTP サーバーでは、このチェックの方が高速である場合があります。	オーバーライド	GenericFileExist
flatten (producer)	フラット化は、ファイル名パスをフラット化して先頭のパスを削除するために使用されるので、ファイル名だけになります。これにより、サブディレクトリーに再帰的に消費できますが、たとえば別のディレクトリーにファイルを書き込むと、それらのファイルは単一のディレクトリーに書き込まれます。これをプロデューサーで true に設定すると、CamelFileName ヘッダーのファイル名が任意の先頭パスに対して削除されます。	false	boolean
moveExisting (producer)	fileExist=Move が設定されている場合に使用するファイル名の計算に使用される式（ファイル言語など）。ファイルをバックアップサブディレクトリーに移動するには、バックアップを入力します。このオプションは、file:name、file:name.ext、file:name.noext、file:onlyname、file:onlyname.noext、file:ext、および file:parent の File Language トークンのみをサポートします。FTP コンポーネントは既存のファイルをベースとして、相対ディレクトリーにのみ移動できるため、file:parent は FTP コンポーネントではサポートされないことに注意してください。		文字列
tempFileName (producer)	tempPrefix オプションと同じですが、File 言語を使用する一時的なファイル名の命名をより細かく制御できます。		文字列

Name	説明	デフォルト	Type
tempPrefix (producer)	このオプションは、一時的な名前を使用してファイルを書き込むために使用されます。次に、書き込みが完了したら、その名前を変更します。書き込まれているファイルを特定し、進行中のファイルで読み取るコンシューマー（排他的読み取りロックを使用しない）も回避できます。大容量のファイルをアップロードする場合に FTP が使用することがよくあります。		文字列
allowNullBody (producer)	ファイルの書き込み中に null ボディーを許可するかどうかを指定するのに使用します。true に設定すると、空のファイルが作成され、false に設定され、null ボディーを file コンポーネントに送信しようとすると、'Cannot write null body to file.' の GenericFileWriteException がスローされます。fileExist オプションを 'Override' に設定すると、ファイルが切り捨てられます。追加する場合は、ファイルは変更されません。	false	boolean
chmod （プロデューサー）	プロデューサーによって送信されるファイルパーミッションを指定します。chmod の値は 000 から 777 の間の値である必要があります。0755 のように先頭の数字がある場合は無視します。		文字列
chmodDirectory (producer)	プロデューサーが不足しているディレクトリーを作成する際に使用するディレクトリーパーミッションを指定します。chmod の値は 000 から 777 の間でなければなりません。0755 のように先頭の数字がある場合は無視されます。		文字列
eagerDeleteTargetFile (producer)	既存のターゲットファイルを活発に削除するかどうか。このオプションは、fileExists=Override オプションおよび tempFileName オプションを使用している場合にのみ適用されます。このパラメーターを使用して、一時ファイルが書き込まれる前にターゲットファイルを削除する（false に設定します）ことができます。たとえば、大きなファイルを作成し、一時ファイルが書き込まれている間にターゲットファイルを存在させたいとします。これにより、一時ファイルの名前がターゲットファイル名に変更される直前に、ターゲットファイルが最後の時点までのみ削除されます。また、このオプションは、fileExist=Move が有効で、既存のファイルが存在すると、既存のファイルを削除するかどうかを制御することもできます。このオプションの copyAndDeleteOnRenameFails false を指定した場合、既存のファイルが存在する場合は、移動操作の前に既存のファイルが削除されます。	true	boolean

Name	説明	デフォルト	Type
forceWrites (producer)	ファイルシステムへの書き込みを強制的に同期するかどうか。ログ/監査ログへの書き込みなど、このレベルの保証が必要ない場合に、このレベルをオフにすることができます。これにより、パフォーマンスが向上します。	true	boolean
keepLastModified (producer)	ソースファイルから最後に変更したタイムスタンプを保持します（存在する場合）。Exchange.FILE_LAST_MODIFIED ヘッダーを使用してタイムスタンプを見つけます。このヘッダーには java.util.Date またはタイムスタンプの long を含めることができます。タイムスタンプが存在し、オプションが有効な場合は、書き込まれたファイルにこのタイムスタンプを設定します。注記： このオプションは、ファイルプロデューサーにのみ適用されます。このオプションは、ftp プロデューサーでは使用できません。	false	boolean
autocreate(advanced)	ファイルのパス名に不足しているディレクトリーを自動的に作成します。ファイルコンシューマーの場合は、開始ディレクトリーを作成することを意味します。ファイルプロデューサーの場合、ファイルが書き込まれるディレクトリーを意味します。	true	boolean
bufferSize (advanced)	バイト単位で書き込みバッファーサイズを書き込みます。	131072	int
copyAndDeleteOnRenameFail (advanced)	ファイルの名前を直接変更できない場合など、フォールバックとコピーおよび削除を行うかどうか。このオプションは FTP コンポーネントでは使用できません。	true	boolean
renameUsingCopy (advanced)	コピーおよび削除ストラテジーを使用して名前変更操作を実行します。これは主に、通常の名前変更操作が信頼できる環境で使用されます（例： 異なるファイルシステムまたはネットワーク全体で）。このオプションは、コピーおよび削除ストラテジーに自動的にフォールバックする copyAndDeleteOnRenameFail パラメーターよりも優先されますが、追加の遅延の後のみになります。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
antExclude (filter)	ant スタイルのフィルターの除外。antInclude と antExclude の両方を使用する場合は、antInclude よりも antExclude が優先されます。複数の除外はコンマ区切りの形式で指定できます。		文字列
antFilterCaseSensitive (filter)	付与フィルターに大文字と小文字を区別するフラグを設定します。	true	boolean
antInclude (filter)	ant スタイルのフィルターが含まれます。複数の包含は、コンマ区切りの形式で指定できます。		文字列
eagerMaxMessagesPerPoll (filter)	maxMessagesPerPoll の制限が Eager かどうかを制御できます。Eager が Eager の場合は、制限がファイルのスキャン中になります。false の場合、すべてのファイルをスキャンし、並び替えを実行します。このオプションを false に設定すると、すべてのファイルを最初にソートし、ポーリングを制限できます。ソートを実行するには、すべてのファイルの詳細がメモリー内にあるため、メモリー使用量を高くする必要があることに注意してください。	true	boolean
exclude （フィルター）	ファイルを除外するために使用されます（ファイル名が正規表現パターンと一致する場合）。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW（）構文を使用してこれを設定する必要があります。エンドポイント URI の設定に関する詳細は、「エンドポイント URI の設定」を参照してください。		文字列
Filter （フィルター）	プラグ可能なフィルター（org.apache.camel.component.file.GenericFileFilter クラス）フィルターが accept（）メソッドで false を返すと、ファイルをスキップします。		GenericFileFilter<T>
filterDirectory (filter)	Simple 言語に基づいてディレクトリーをフィルタリングします。たとえば、現在の日付でフィルタリングするには、$date:now:yyMMdd などの単純な日付パターンを使用できます。		文字列
filterFile (filter)	Simple 言語に基づいてファイルをフィルタリングします。たとえば、ファイルサイズで絞り込むには、$file:size 5000 を使用できます。		文字列

Name	説明	デフォルト	Type
idempotent (filter)	Idempotent Consumer EIP パターンを使用して、Camel がすでに処理されたファイルをスキップするオプション。デフォルトでは、1000 エントリーを保持するメモリーベースの LRUCache を使用します。noop=true の場合は、同じファイルを何度も使用することを回避するため、べき等性も有効になります。	false	ブール値
idempotentKey (filter)	カスタムのべき等キーを使用するには、以下を行います。デフォルトでは、ファイルの絶対パスが使用されます。File 言語を使用すると、ファイル名とファイルサイズを使用できます (idempotentKey=$file:name-$file:size)。		文字列
idempotentRepository (filter)	何も指定されておらず、べき等性が true の場合、デフォルトでは MemoryMessageIdRepository を使用するプラグ可能なリポジトリー org.apache.camel.spi.IdempotentRepository。		String>
include （フィルター）	ファイルを含めるために使用されます。ファイル名が正規表現パターンと一致する場合（一致する場合は大文字と小文字が区別されます）。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW（）構文を使用してこれを設定する必要があります。エンドポイント URI の設定に関する詳細は、「エンドポイント URI の設定」を参照してください。		文字列
maxDepth (filter)	ディレクトリーを再帰的に処理する際に通過する最大深度。	2147483647	int
maxMessagesPerPoll (filter)	ポーリングごとに収集する最大メッセージを定義します。デフォルトでは最大値は設定されません。1000 などの制限を設定して、数千のファイルがあるサーバーを起動すると回避できます。無効にするには、0 または negative の値を設定します。注記： このオプションが使用されている場合、File および FTP コンポーネントはソート前に制限されます。たとえば、100000 ファイルがあり、maxMessagesPerPoll=500 を使用する場合は、最初の 500 ファイルのみが選択され、その後にソートされます。eagerMaxMessagesPerPoll オプションを使用して、これを false に設定すると、最初にすべてのファイルをスキャンし、後でソートできます。		int
minDepth (filter)	ディレクトリーを再帰的に処理する際に処理を開始する最小深度。minDepth=1 を使用すると、ベースディレクトリーを意味します。minDepth=2 を使用すると、最初のサブディレクトリーを意味します。		int

Name	説明	デフォルト	Type
Move （フィルター）	処理後にファイル名を動的に設定するために使用される式（Simple Language など）。ファイルを .done サブディレクトリーに移動するには、.done と入力します。		文字列
exclusiveReadLockStrategy (lock)	プラグ可能な read-lock を org.apache.camel.component.file.GenericFileExclusiveReadLockStrategy 実装とする。		GenericFileExclusive ReadLockStrategy <T>
readLock (lock)	ファイルに read-lock が排他的な場合にのみファイルをポーリングするために、コンシューマーによって使用されます（ファイルが進行中の場合や書き込みされていません）。Camel はファイルロックが許可されるまで待機します。このオプションは、ストラテジーでビルドを提供します。none - No read lock is in use markerFile - Camel はマーカーファイル (fileName.camelLock)を作成し、そのロックを保持します。このオプションは、FTP コンポーネントの変更には使用できません。Changed は、ファイルの長さ/変更タイムスタンプを使用して、ファイルがすでにコピーされているかどうかを検出します。少なくとも 1 つのセクションを使用してこれを判断するため、このオプションはファイルを他のプロセスとしてすぐに消費できませんが、JDK IO API はファイルを別のプロセスで現在使用しているかどうかを判断することができないため、信頼性が高まります。readLockCheckInterval オプションを使用してチェック頻度を設定できます。fileLock - は java.nio.channels.FileLock 用です。このオプションは FTP コンポーネントでは使用できません。この方法は、ファイルシステムが分散ファイルロックに対応している場合を除き、マウント/共有経由でリモートファイルシステムにアクセスするときに回避する必要があります。名前変更は、読み取り専用で読み取りロックを取得できる場合に、テストとしてファイルの名前を変更するためのものです。べき等性 -（ファイルコンポーネントのみ）べき等性は、idempotentRepository を read-lock として使用することです。これにより、べき等リポジトリーの実装がサポートする場合、クラスタリングをサポートする読み取りロックを使用できます。idempotent-changed -（ファイルコンポーネントのみ）idempotent-changed はidempotentRepository を使用し、結合された read-lock として変更されます。これにより、べき等リポジトリー実装がこれをサポートする場合、クラスタリングをサポートする読み取りロックを使用できます。idempotent-rename -（ファイルコンポーネントのみ）idempotent-rename はidempotentRepository を使用し、結合された read-lock として名前を変更しま	none	文字列

Name	説明	デフォルト	Type
	す。これにより、べき等リポジトリーの実装がサポートしている場合、クラスタリングをサポートする読み取りロックを使用できます。注記： 各種の読み取りロックは、クラスターモードですべて機能する訳ではありません。この場合、異なるノードの同時コンシューマーは共有ファイルシステムの同じファイルに対して競合しています。atomic 操作を使用して空のマーカーファイルを作成しますが、クラスター内での動作が保証されません。fileLock の方が良好に機能しますが、ファイルシステムは分散ファイルロックなどに対応する必要があります。べき等リポジトリーが Hazelcast コンポーネントや Infinispan などのクラスタリングに対応している場合、べき等リポジトリーがクラスタリングをサポートする場合は、べき等読み取りロックを使用できます。		
readLockCheckInterval (lock)	読み取りロックでサポートされている場合は、readLock の間隔（ミリ秒単位）。この間隔は、読み取りロックの取得を試みる間にスリープするために使用されます。たとえば、変更した読み取りロックを使用する場合は、低速な書き込みのために、間隔を cater に設定できます。デフォルトの 1 秒。プロデューサーがファイルを書き込む速度が非常に遅い場合は、速度が長すぎる可能性があります。注記：FTP の場合、デフォルトの readLockCheckInterval は 5000 です。readLockTimeout の値は readLockCheckInterval よりも大きくなければなりませんが、thumb のルールは readLockCheckInterval よりも少なくとも 2 倍以上タイムアウトになるようにします。これは、読み取りロックプロセスでタイムアウトがヒットするまでにロックの取得を試行するために バブル時間が許可されるようにするために必要です。	1000	Long
readLockDeleteOrphanLock Files (lock)	Camel が適切にシャットダウンされない場合（JVM クラッシュなど）、ファイルシステム上に残っている可能性のある孤立した読み取りロックファイルを削除する際にマーカーファイルを使用したロックを削除するかどうか。このオプションを false に指定すると、孤立したロックファイルがあると、Camel がそのファイルを取得しようとしません。また、別のノードが同じ共有ディレクトリーからファイルを同時に読み取ることが原因として考えられます。	true	boolean
readLockLoggingLevel (lock)	読み取りロックを取得できなかったときに使用されるロギングレベル。デフォルトでは、WARN がログに記録されます。このレベルは OFF でロギングを持たずに変更できます。このオプションは、type の readLock（changed、fileLock、idempotent、idempotent-changed、idempotent-rename、rename）にのみ適用できます。	DEBUG	LoggingLevel

Name	説明	デフォルト	Type
readLockMarkerFile (lock)	変更、名前変更、または排他的読み取りロックタイプでマーカーファイルを使用するかどうか。デフォルトでは、マーカーファイルが使用されるだけでなく、同じファイルを選択する他のプロセスに対して保護されます。このオプションを false に設定すると、この動作をオフにすることができます。たとえば、Camel アプリケーションによってマーカーファイルをファイルシステムに書き込みます。	true	boolean
readLockMinAge (lock)	このオプションは、readLock=change にのみ適用されます。このオプションでは、読み取りロックの取得を試行する前にファイルに必要な最小期間を指定できます。たとえば、readLockMinAge=300s を使用して、ファイルを最後の 5 分前とする必要があります。これにより、指定された期間以上のファイルの取得を試みるため、変更した読み取りロックが高速化されます。	0	Long
readLockMinLength (lock)	このオプションは、readLock=changed にのみ適用されます。このオプションを使用すると、最小限のファイルの長さを設定できます。デフォルトでは、Camel はファイルにデータが含まれることを想定するため、デフォルト値は 1 です。このオプションをゼロに設定するには、ゼロ長のファイルを使用できます。	1	Long
readLockRemoveOnCommit (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションでは、ファイルの処理に成功し、コミットが行われるときに、べき等リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。デフォルトでは、このファイルは削除されません。これにより、競合状態が発生しないため、別のアクティブなノードがファイルを取得できなくなります。代わりに、べき等リポジトリーは、X 分後にファイル名のエントリーをエビクトするように設定するエビクションストラテジーをサポートする可能性があります。これにより、競合状態に関する問題がなくなります。	false	boolean
readLockRemoveOnRollback (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションでは、ファイルの処理に失敗し、ロールバックが行われるときに、べき等リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。このオプションが false の場合、ファイル名のエントリーが確認されます（ファイルがコミットしたかのように）。	true	boolean

Name	説明	デフォルト	Type
readLockTimeout (lock)	read-lock で対応している場合は、read-lock のオプションのタイムアウト（ミリ秒単位）。read-lock が許可されず、タイムアウトが発生すると、Camel はファイルをスキップします。次回のポーリング Camel はファイルを再度試行し、今回は読み取りロックが付与される可能性があります。0 以下の値を使用して、永久に指定します。現在、fileLock、change、および rename はタイムアウトに対応しています。注記： FTP の場合、デフォルトの readLockTimeout 値は 10000 ではなく 20000 です。readLockTimeout の値は readLockCheckInterval よりも大きくなければなりませんが、thumb のルールは readLockCheckInterval よりも少なくとも 2 倍以上タイムアウトになるようにします。これは、読み取りロックプロセスでタイムアウトがヒットするまでにロックの取得を試行するために バブル時間が許可されるようにするために必要です。	10000	Long
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。デフォルト値は 500 です。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean

Name	説明	デフォルト	Type
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。デフォルト値は 1000 です。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。このオプションを使用すると、複数のコンシューマー間でスレッドプールを共有できます。		ScheduledExecutor Service
scheduler (scheduler)	カスタムの org.apache.camel.spi.ScheduledPollConsumerScheduler をポーリングコンシューマーの実行時に実行するスケジューラーとして使用することができます。デフォルトの実装では ScheduledExecutorService を使用し、CRON 式をサポートする Quartz2 および Spring ベースのものがあります。注記： カスタムスケジューラーを使用する場合は、initialDelay のオプション、useFixedDelay、timeUnit、および scheduledExecutorService が使用されていない可能性があります。quartz2 のテキストを使用して Quartz2 スケジューラーを参照し、Spring のテキストを使用して Spring ベースを使用し、myScheduler のテキストを使用してレジストリーの id でカスタムスケジューラーを参照します。例については、Quartz2 ページを参照してください。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

Name	説明	デフォルト	Type
shuffle (sort)	ファイルの一覧をシャッフルする方法（ランダムな順序など）	false	boolean
sortBy (sort)	File 言語を使用した組み込みソート。入れ子のソートをサポートするため、ファイル名でソートされ、2 番目にグループの種類を変更した日付でソートできます。		文字列
sorter (sort)	java.util.Comparator クラスとしてのプラグ可能なソーター。		GenericFile<T>>

ヒント

デフォルトでは、ファイルプロデューサーのデフォルト動作 は、同じ名前の既存のファイルが存在する場合は、既存のファイルを上書きします。

100.3. 移動および削除操作

移動または削除操作は、ルーティングが完了した後（コマンド後）に実行されます。そのため、Exchange の処理中には、ファイルが受信トレイフォルダーに配置されます。

例として以下を説明します。

```
from("file://inbox?move=.done").to("bean:handleOrder");
```

inbox フォルダーでファイルが削除されると、ファイルコンシューマーはこれを認識し、handleOrder Bean にルーティングされる新しい FileExchange を作成します。その後、Bean は File オブジェクトを処理します。この時点で、ファイルは 受信トレイ フォルダーにあります。Bean の完了後にルートが完了すると、ファイルコンシューマーは move 操作を実行し、そのファイルを .done サブフォルダーに移動します。

move オプションと preMove オプションはディレクトリー名とみなされます（File Language などの式を使用する場合、Simple の場合は、式評価の結果は使用するファイル名になります）- 設定すると、たとえば式評価の結果が使用されます。

```
move=../backup/copy-of-${file:name}
```

次に、使用する File 言語 を使用して、使用されるファイル名を返します。これは、相対または絶対のいずれかになります。相対すると、ディレクトリーは、ファイルが使用されたフォルダーからサブフォルダーとして作成されます。

デフォルトでは、Camel は消費されたディレクトリーと相対的に、コンシュームされたファイルを .camel サブフォルダーに移動します。

処理後にファイルを削除する場合は、ルートは以下のようになります。

```
from("file://inobox?delete=true").to("bean:handleOrder");
```

ファイルの処理 前 に、ファイルを移動するための 事前 移動操作を導入しました。これにより、スキャンされたファイルが処理される前にこのサブフォルダーに移動されたときにマークできます。

```
from("file://inbox?preMove=inprogress").to("bean:handleOrder");
```

前移動と通常の 移動 を組み合わせることができます。

```
from("file://inbox?preMove=inprogress&move=.done").to("bean:handleOrder");
```

そのため、このファイルは処理時と処理後に 進行中 のフォルダーに置かれ、.done フォルダーに移動します。

100.4. MOVE オプションおよび PREMOVE オプションに対する詳細な制御

move オプションおよび preMove オプションは Expression ベースで、ディレクトリーおよび名前パターンの詳細設定を 行う File 言語 の完全な機能を持ちます。
Camel は実際には、入力したディレクトリー名を File 言語 式に変換します。そのため、move=.done Camel はこれを ${file:parent}/.done/${"file:onlyname} に変換します。これは、オプション値に $\{ } が提供していないことを Camel が検出した場合にのみ行われます。そのため、$\{ } Camel を入力しても変換され ない ため、フル電源があります。

そのため、現在の日付のパターンでファイルをバックアップフォルダーに移動する場合は、以下を行うことができます。

```
move=backup/${date:now:yyyyMMdd}/${file:name}
```

100.5. ABOUT MOVEFAILED

moveFailed オプションを使用すると、選択したエラーフォルダーなど、別の場所に 適切に処理できない ファイルを移動できます。たとえば、タイムスタンプのあるエラーフォルダー内のファイルを移動するには、moveFailed=/error/${file:name.noext}-${date:now:yyyyMMddHHmmssSSS}.${"file:ext}を使用できます。

その他の例は、「File Language」を参照してください。

100.6. メッセージヘッダー

このコンポーネントでは、以下のヘッダーがサポートされます。

100.6.1. ファイルプロデューサーのみ

ヘッダー	説明
CamelFileName	書き込むファイルの名前を指定します（エンドポイントディレクトリーと相対的）。この名前は **String** となり、File 言語 または Simple 式を持つ文字列、または Expression オブジェクトになります。**null** の場合、Camel はメッセージー意の ID に基づいてファイル名を自動生成します。
CamelFileNameProduced	書き込まれた出力ファイルの実際の絶対パス（パス + 名）。このヘッダーは Camel によって設定され、その目的は書き込まれたファイルの名前でエンドユーザーを提供します。
CamelOverruleFileName	Camel 2.11: **CamelFileName** ヘッダーをオーバーライドするために使用され、代わりに値を使用します（ただし、プロデューサーはファイルの記述後にこのヘッダーが削除されるため、1 回のみ）。値は String のみになります。**fileName** オプションが設定されている場合、これは引き続き評価されます。

100.6.2. ファイルコンシューマーのみ

ヘッダー	説明
CamelFileName	エンドポイントに設定された開始ディレクトリーからのオフセットを使用した相対パスとして消費されたファイルの名前。

ヘッダー	説明
CamelFileNameOnly	ファイル名（先頭のパスがない名前）のみ。
CamelFileAbsolute	消費されるファイルが絶対パスを表すかどうかを指定する **ブール値** オプション。通常、相対パスでは **false** である必要があります。通常、絶対パスは使用しないでくださいが、ファイルを絶対パスに移動できるように move オプションに追加しています。ただし、他の場所も使用できます。
CamelFileAbsolutePath	ファイルへの絶対パス。相対ファイルの場合、このパスは代わりに相対パスを保持します。
CamelFilePath	ファイルパス。相対ファイルの場合、これは開始ディレクトリー + 相対ファイル名です。絶対パスの場合、これは絶対パスになります。
CamelFileRelativePath	相対パス。
CamelFileParent	親パス。
CamelFileLength	ファイルサイズを含む **long** 値。
CamelFileLastModified	ファイルの最後に変更されたタイムスタンプが含まれる **Long** 値。Camel 2.10.3 **以前** では、タイプは **Date** です。

100.7. バッチコンシューマー

このコンポーネントは Batch Consumer を実装します。

100.8. エクスチェンジプロパティー、ファイルコンシューマーのみ

ファイルコンシューマーは BatchConsumer を実装するため、ポーリングするファイルのバッチをサポートします。バッチ処理によって、Camel は以下のプロパティーが Exchange に追加されること

を意味します。したがって、ポーリングされたファイルの数、現在のインデックス、バッチがすでに完了しているかどうかを知ることができます。

プロパティー	説明
CamelBatchSize	このバッチでポーリングされたファイルの合計数。
CamelBatchIndex	バッチの現在のインデックス。0 から開始します。
CamelBatchComplete	バッチの最後のエクスチェンジを示す **ブール** 値。最後のエントリーにのみ **該当** します。

これにより、たとえば、このバッチに存在するファイル数を把握し、Aggregator2 でこの数のファイルを集約することもできます。

100.9. CHARSET の使用

Camel 2.9.3 で利用可能。charset オプションを使用すると、コンシューマーおよびプロデューサーエンドポイントの両方でファイルのエンコーディングを設定できます。たとえば、utf-8 ファイルを読み取り、ファイルを iso-8859-1 に変換する場合は、以下を行うことができます。

```
from("file:inbox?charset=utf-8")
  .to("file:outbox?charset=iso-8859-1")
```

ルートで convertBodyTo を使用することもできます。以下の例では、utf-8 形式のファイルを引き続き入力していますが、ファイルコンテンツを iso-8859-1 形式のバイトアレイに変換したいとします。そして、Beanにデータを処理させます。現在の文字セットを使用して、送信トレイフォルダーにコンテンツを書き込む前に、コンテンツを送信します。

```
from("file:inbox?charset=utf-8")
  .convertBodyTo(byte[].class, "iso-8859-1")
  .to("bean:myBean")
  .to("file:outbox");
```

コンシューマーエンドポイントで文字セットを省略すると、Camel はファイルの文字セットを認識せず、デフォルトでは「UTF-8」を使用します。ただし、JVM システムプロパティーを設定して、キー

org.apache.camel.default.charset で別のデフォルトエンコーディングを上書きして使用できます。

以下の例では、ファイルが UTF-8 エンコーディングにない場合（ファイルを読み取るデフォルトのエンコーディング）が問題になる可能性があります。
この例では、ファイルを書き込む際に、コンテンツがすでにバイトアレイに変換されているため、コンテンツをそのまま直接書き込みします（追加のエンコーディングは必要ありません）。

```
from("file:inbox")
  .convertBodyTo(byte[].class, "iso-8859-1")
  .to("bean:myBean")
  .to("file:outbox");
```

ファイルの書き込み時にエンコーディングを動的に上書きおよび制御することもできます。そのためには、エクスチェンジにエクスチェンジのプロパティーを設定することで、キー Exchange.CHARSET_NAME を使用します。たとえば、以下のルートでは、メッセージヘッダーからの値でプロパティーを設定します。

```
from("file:inbox")
  .convertBodyTo(byte[].class, "iso-8859-1")
  .to("bean:myBean")
  .setProperty(Exchange.CHARSET_NAME, header("someCharsetHeader"))
  .to("file:outbox");
```

ここでは簡単にするため、同じエンコーディングでファイルを取得して特定のエンコーディングでファイルを書きたい場合は、エンドポイントで charset オプションを使用してください。

エンドポイントに charset オプションを明示的に設定している場合は、Exchange.CHARSET_NAME プロパティーに関係なく設定が使用されます。

問題がある場合には、org.apache.camel.component.file で DEBUG ロギングを有効にし、特定の文字セットを使用してファイルの読み取り/書き込み時に Camel ログを有効にできます。
たとえば、以下のルートは以下をログに記録します。

```
from("file:inbox?charset=utf-8")
  .to("file:outbox?charset=iso-8859-1")
```

ログは以下のようになります。

```
DEBUG GenericFileConverter          - Read file /Users/davsclaus/workspace/camel/camel-core/target/charset/input/input.txt with charset utf-8
DEBUG FileOperations                - Using Reader to write file: target/charset/output.txt with charset: iso-8859-1
```

_

100.10. フォルダーとファイル名を含む一般的な検索

Camel がファイル（書き込みファイル）を生成する際に、必要なファイル名の設定方法に影響します。デフォルトでは、Camel はメッセージ ID をファイル名として使用します。メッセージ ID は通常一意の生成された ID であるため、ID-MACHINENAME-2443-1211718892437-1-0 などのファイル名で終わります。このようなファイル名が望ましくない場合は、CamelFileName メッセージヘッダーにファイル名を指定する必要があります。定数 Exchange.FILE_NAME も使用できます。

以下のサンプルコードは、メッセージ ID をファイル名として使用してファイルを作成します。

```
from("direct:report").to("file:target/reports");
```

report.txt を必要なファイル名として使用するには、以下を実行します。

```
from("direct:report").setHeader(Exchange.FILE_NAME, constant("report.txt")).to(
"file:target/reports");
```

i. 上記と同じですが、CamelFileName と同じです。

```
from("direct:report").setHeader("CamelFileName", constant("report.txt")).to(
"file:target/reports");
```

また、fileName URI オプションを使用してエンドポイントにファイル名を設定する構文。

```
from("direct:report").to("file:target/reports/?fileName=report.txt");
```

100.11. ファイル名の式

ファイル名を設定するには、式 オプションを使用するか、CamelFileName ヘッダーの文字列ベースの File Language 式として使用できます。構文およびサンプルは、「File Language」を参照してください。

100.12. ファイルを直接ドロップするフォルダーからのファイルの使用

他のアプリケーションが直接ファイルを書き込むフォルダーからファイルを使用する場合は注意してください。ユースケースに適した readLock オプションを確認してください。ただし、別のフォルダーに書き込みを行い、書き込みが drop フォルダー内のファイルを移動した後に、最善の方法です。ただし、ドロップフォルダーに直接ファイルを書き込むと、ファイルが変更されたアルゴリズムを使用して

一定期間ファイルのサイズや修正の変更の有無を確認するため、オプションが変更されたかどうかを簡単に検出できます。他の readLock オプションは、Java File API に依存しますが、これは常に検出するのに非常に適切ではありません。また、doneFileName オプションを確認することもできます。これは、マーカーファイル（完了ファイル）を使用して、ファイルが完了したときに使用でき、使用できる状態にすることができます。

100.13. 完了したファイルの使用

Camel 2.6 で利用可能

以下の「実行されたファイルの記述」 も参照してください。

完了したファイルが存在するときにのみファイルを消費する場合は、エンドポイントで doneFileName オプションを使用できます。

```
from("file:bar?doneFileName=done");
```

実行しているファイルがターゲットファイルと同じディレクトリーに存在する場合にのみ bar フォルダーからファイルを使用します。Camel は、ファイルを使用しているときに、完了ファイル を自動的に削除します。Camel 2.9.3 以降では、noop=true が設定されている場合、Camel は 完了ファイル を自動的に削除しません。

ただし、ターゲットファイルごとに 1 つの done ファイル があると一般的です。これは、1:1 の相関があることを意味します。これを行うには、doneFileName オプションで動的プレースホルダーを使用する必要があります。現在 Camel は、$\{ } で囲む必要がある file:name および file:name.noext の 2 つの動的トークンをサポートします。コンシューマーは、実行されたファイル 名の静的部分のみを接頭辞または接尾辞（両方ではない）としてサポートします。

```
from("file:bar?doneFileName=${file:name}.done");
```

この例では、ファイル名が .done のファイルが存在する場合にのみポーリングされます。たとえば、以下のようになります。

- hello.txt: 消費されるファイルです。
- hello.txt.done - 関連付けられた完了ファイルです。

完了ファイルには、以下のような接頭辞を使用することもできます。

```
from("file:bar?doneFileName=ready-${file:name}");
```

- hello.txt: 消費されるファイルです。
- ready-hello.txt: 関連する完了ファイルです。

100.14. 実行されたファイルの記述

Camel 2.6 で利用可能

ファイルを作成したら、追加の done ファイルをマーカーとして記述して、ファイル が終了し、書き込まれた他のファイルを示すことができます。これを行うには、ファイルプロデューサーエンドポイントで doneFileName オプションを使用します。

```
.to("file:bar?doneFileName=done");
```

ターゲットファイルと同じディレクトリーに 実行 されたファイルを作成します。

ただし、ターゲットファイルごとに 1 つの done ファイルがあると一般的です。これは、1:1 の相関があることを意味します。これを行うには、doneFileName オプションで動的プレースホルダーを使用する必要があります。現在 Camel は、$\{ } で囲む必要がある file:name および file:name.noext の 2 つの動的トークンをサポートします。

```
.to("file:bar?doneFileName=done-${file:name}");
```

たとえば、ターゲットファイルがターゲットファイルと同じディレクトリーに foo.txt の場合、done-foo.txt という名前のファイルを作成します。

```
.to("file:bar?doneFileName=${file:name}.done");
```

たとえば、ターゲットファイルがターゲットファイルと同じディレクトリーに foo.txt. done という名前のファイルを作成します。

```
.to("file:bar?doneFileName=${file:name.noext}.done");
```

たとえば、ターゲットファイルがターゲットファイルと同じディレクトリーに foo.txt であった場合は、foo.done という名前のファイルを作成します。

100.15. サンプル

```
#=== read from a directory and write to another directory

from("file://inputdir/?delete=true").to("file://outputdir")
```

100.15.1. overrule の動的名前を使用して、ディレクトリーからの読み書きを行い、別のディレクトリーに書き込みます。

```
from("file://inputdir/?delete=true").to("file://outputdir?overruleFile=copy-of-${file:name}")
```

ディレクトリーをリッスンし、そこで破棄された各ファイルのメッセージを作成します。outputdir にコンテンツをコピーし、inputdir のファイルを削除します。

100.15.2. ディレクトリーから再帰的に読み取り、別のディレクトリーへの書き込み

```
from("file://inputdir/?recursive=true&delete=true").to("file://outputdir")
```

ディレクトリーをリッスンし、そこで破棄された各ファイルのメッセージを作成します。outputdir にコンテンツをコピーし、inputdir のファイルを削除します。サブディレクトリーに再帰的にスキャンします。は、任意のサブディレクトリーを含む、input dir と同じディレクトリー構造にファイルを配置します。

```
inputdir/foo.txt
inputdir/sub/bar.txt
```

出力レイアウトは以下のようになります。

```
outputdir/foo.txt
outputdir/sub/bar.txt
```

100.16. フラット化の使用

ファイルを outputdir ディレクトリーに格納する場合は、ソースディレクトリーのレイアウト（例：パスをフラット化する場合など）を破棄する場合、ファイルプロデューサー側に flatten=true オプションを追加します。

```
from("file://inputdir/?recursive=true&delete=true").to("file://outputdir?flatten=true")
```

出力レイアウトは以下のようになります。

```
outputdir/foo.txt
outputdir/bar.txt
```

100.17. ディレクトリーからの読み取りとデフォルトの移動操作

Camel はデフォルトで、処理されたファイルを、ファイルが使用されたディレクトリーの .camel サブディレクトリーに移動します。

```
from("file://inputdir/?recursive=true&delete=true").to("file://outputdir")
```

レイアウトには、以下の 前に
のレイアウトに影響します。

```
inputdir/foo.txt
inputdir/sub/bar.txt
```

after

```
inputdir/.camel/foo.txt
inputdir/sub/.camel/bar.txt
outputdir/foo.txt
outputdir/sub/bar.txt
```

100.18. ディレクトリーから読み込み、JAVA でメッセージを処理します。

```
from("file://inputdir/").process(new Processor() {
  public void process(Exchange exchange) throws Exception {
    Object body = exchange.getIn().getBody();
    // do some business logic with the input body
  }
});
```

ボディーは、inputdir ディレクトリーにドロップされたファイルを参照する File オブジェクトです。

100.19. ファイルへの書き込み

Camel は当然、ファイル（つまりファイルの生成）も記述できます。以下の例では、ディレクトリーに書き込まれる前に、処理する SEDA キューに関するレポートの一部を受信します。

100.19.1. Exchange.FILE_NAMEを使用してサブディレクトリーに書き込みます。

単一ルートを使用すると、ファイルを任意の数のサブディレクトリーに書き込むことができます。以下のようなルートが設定されている場合：

```
<route>
  <from uri="bean:myBean"/>
  <to uri="file:/rootDirectory"/>
</route>
```

myBean で、ヘッダー Exchange.FILE_NAME を以下のような値に設定できます。

```
Exchange.FILE_NAME = hello.txt => /rootDirectory/hello.txt
Exchange.FILE_NAME = foo/bye.txt => /rootDirectory/foo/bye.txt
```

これにより、複数の宛先にファイルを書き込む単一のルートを作成できます。

100.19.2. 一時ディレクトリーから最終宛先と相対的なファイルを書き込む

一部のディレクトリーに、ファイルを宛先ディレクトリーと相対的に一時的に書き込む必要がある場合もあります。通常、このような状況は、フィルター機能に制限のある外部プロセスが書き込み先のディレクトリーから読み取る場合に発生します。以下の例では、ファイルは /var/myapp/filesInProgress ディレクトリーに書き込まれ、データ転送が完了すると、/var/myapp/finalDirectory の 'directory' にアトミックに移動します。

```
from("direct:start").
  to("file:///var/myapp/finalDirectory?tempPrefix=/../filesInProgress/");
```

100.20. ファイル名の式の使用

この例では、現在の日付をサブフォルダー名として使用して、使用されたファイルをバックアップフォルダーディレクトリーに移動します。

```
from("file://inbox?move=backup/${date:now:yyyyMMdd}/${file:name}").to("...");
```

その他のサンプルは、「File Language」を参照してください。

100.21. 同じファイルを複数回読み取りないようにする（べき等コンシューマー）

Camel はコンポーネント内でべき等コンシューマーを直接サポートし、すでに処理されたファイルをスキップします。この機能は、idempotent=true オプションを設定して有効にできます。

```
from("file://inbox?idempotent=true").to("...");
```

Camel は絶対ファイル名をべき等キーとして使用し、重複ファイルを検出します。Camel 2.11 以降では、idempotentKey オプションの式を使用してこのキーをカスタマイズできます。たとえば、name と file の両方をキーとして使用する場合

```
<route>
  <from uri="file://inbox?idempotent=true&amp;idempotentKey=${file:name}-${file:size}"/>
  <to uri="bean:processInbox"/>
</route>
```

デフォルトでは、Camel は消費されたファイルの追跡にメモリーベースのストアを使用し、最大 1000 エントリーを保持する、最も最近使用されたキャッシュを使用します。値に # 記号を使用して、指定の ID を持つレジストリー内の Bean を参照することを示すことで、このストアの独自の実装をプラグインでき ます。

```
<!-- define our store as a plain spring bean -->
<bean id="myStore" class="com.mycompany.MyIdempotentStore"/>

<route>
  <from uri="file://inbox?idempotent=true&amp;idempotentRepository=#myStore"/>
  <to uri="bean:processInbox"/>
</route>
```

Camel は、以前使用されたためファイルを省略した場合、DEBUG レベルでログに記録されます。

```
DEBUG FileConsumer is idempotent and the file has been consumed before. Will skip this file:
target\idempotent\report.txt
```

100.22. ファイルベースのべき等リポジトリーの使用

このセクションでは、デフォルトとして使用されるインメモリーベースではなく、ファイルベースのべき等リポジトリー org.apache.camel.processor.idempotent.FileIdempotentRepository を使用します。
このリポジトリーは、ファイルリポジトリーの読み取りを回避するために 1 ストレベルキャッシュを使

用します。ここでは、files リポジトリーのみを使用して 1 ストレベルキャッシュのコンテンツを保存します。これにより、リポジトリーはサーバーの再起動後も維持されます。起動時に、ファイルの内容が 1 ストレベルキャッシュに読み込まれます。ファイル構造は非常に簡単です。キーをファイルに別々の行に保存します。デフォルトでは、ファイルストアのサイズ制限は 1mb です。サイズの大きい Camel がファイルストアを切り捨てると、1 ストレベルキャッシュを新規の空のファイルにフラッシュしてコンテンツを再ビルドします。

Spring XML を使用してリポジトリーのべき等リポジトリーを作成し、# 記号を使用してレジストリールックアップを示す idempotentRepository でリポジトリーを使用するようにファイルコンシューマーを定義します。

100.23. JPA ベースのべき等リポジトリーの使用

このセクションでは、デフォルトとして使用されるインメモリーベースの代わりに JPA ベースのべき等リポジトリーを使用します。

まず META-INF/persistence.xml に persistence-unit が必要です。ここで、org.apache.camel.processor.idempotent.jpa.MessageProcessed クラスをモデルとして使用する必要があります。

```
<persistence-unit name="idempotentDb" transaction-type="RESOURCE_LOCAL">
  <class>org.apache.camel.processor.idempotent.jpa.MessageProcessed</class>

  <properties>
    <property name="openjpa.ConnectionURL" value="jdbc:derby:target/idempotentTest;create=true"/>
    <property name="openjpa.ConnectionDriverName"
value="org.apache.derby.jdbc.EmbeddedDriver"/>
    <property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema"/>
    <property name="openjpa.Log" value="DefaultLevel=WARN, Tool=INFO"/>
    <property name="openjpa.Multithreaded" value="true"/>
  </properties>
</persistence-unit>
```

次に、Spring XML ファイルに JPA べき等リポジトリーを作成することもできます。

```
<!-- we define our jpa based idempotent repository we want to use in the file consumer -->
<bean id="jpaStore" class="org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository">
    <!-- Here we refer to the entityManagerFactory -->
    <constructor-arg index="0" ref="entityManagerFactory"/>
    <!-- This 2nd parameter is the name  (= a category name).
         You can have different repositories with different names -->
    <constructor-arg index="1" value="FileConsumer"/>
</bean>
```

はい、# syntax オプションを使用して idempotentRepository を使用して、ファイルコンシュー

マーエンドポイントの jpaStore Bean を参照する必要があります。

```
<route>
  <from uri="file://inbox?idempotent=true&amp;idempotentRepository=#jpaStore"/>
  <to uri="bean:processInbox"/>
</route>
```

100.24. FILTER USING ORG.APACHE.CAMEL.COMPONENT.FILE.GENERICFILEFILTER

Camel はプラグ可能なフィルタリングストラテジーをサポートします。その後、このようなフィルターでエンドポイントを設定し、処理された特定のファイルをスキップできます。

この例では、ファイル名の skip で始まるファイルを省略する独自のフィルターを構築します。

そして、filter 属性を使用して Spring XML ファイルで定義したフィルターを参照する（# 表記を使用）、ルートを設定することができます。

```
<!-- define our filter as a plain spring bean -->
<bean id="myFilter" class="com.mycompany.MyFileFilter"/>

<route>
  <from uri="file://inbox?filter=#myFilter"/>
  <to uri="bean:processInbox"/>
</route>
```

100.25. ANT パス MATCHER を使用したフィルタリング

ANT パス matcher は camel-spring jar の追加設定なしで提供されます。そのため、Maven を使用している場合は camel-spring に依存する必要があります。
実際の照合を行うために Spring の AntPathMatcher を利用することが理由です。

ファイルパスは、以下のルールと一致します。

- ? 1 文字に一致します。
- * 0 以上の文字に一致します。

- ** パスの 0 個以上のディレクトリーと一致します。

ヒント

Camel 2.10 以降では、フィルターを定義せずに ANT スタイルの include/exclude を簡単に指定できるように antInclude オプションおよび antExclude オプションが追加されました。詳細は、上記の URI オプションを参照してください。

以下の例は、その使用方法を示しています。

100.25.1. Comparator を使用したソート

Camel はプラグ可能なソートストラテジーをサポートします。このストラテジーでは、Java の java.util.Comparator でビルドを使用します。その後、このようなコンパレーターでエンドポイントを設定し、Camel が処理前にファイルを並べ替えることができます。

この例では、ファイル名でソートする独自のコンパレーターを構築します。

そして、Spring XML ファイルで定義した ソー ター(mySorter)を参照する sorter オプションを使用してルートを設定することができます。

```
<!-- define our sorter as a plain spring bean -->
<bean id="mySorter" class="com.mycompany.MyFileSorter"/>

<route>
  <from uri="file://inbox?sorter=#mySorter"/>
  <to uri="bean:processInbox"/>
</route>
```

ヒント

URI オプションは、# 構文を使用して Bean を参照でき ます。上記の Spring DSL ルートの # は、ID に # をプレフィックスしてレジストリー内の Bean を参照することに注意してください。そのため、sorter=#mySorter を記述すると、Camel に対して ID(mySorter)を持つ Bean のレジストリーを検索するように指示します。

100.25.2. sortBy を使用したソート

Camel はプラグ可能なソートストラテジーをサポートします。このストラテジーでは、File 言語 を

使用してソートを設定します。**sortBy** オプションは、以下のように設定されます。

```
sortBy=group 1;group 2;group 3;...
```

各グループはセミコロンで区切られます。簡単な状況では、簡単な例としては 1 つのグループのみを使用できます。

```
sortBy=file:name
```

これはファイル名でソートされ、逆方向に 逆 順をやり直すことができるため、ソートは **Z.A** になります。

```
sortBy=reverse:file:name
```

File 言語 のフルパワーがあるため、他のパラメーターの一部を使用することができます。そのため、ファイルサイズでソートする場合は、以下を行います。

```
sortBy=file:length
```

文字列比較に **ignoreCase:** を使用してケースを無視するように設定できます。そのため、ファイル名のソートを使用し、ケースを無視する場合は、以下を行います。

```
sortBy=ignoreCase:file:name
```

ignore 大文字と小文字を組み合わせることができますが、最初に逆に指定する必要があります。

```
sortBy=reverse:ignoreCase:file:name
```

以下の例では、最後に変更したファイルで並べ替えたいので、以下を行います。

```
sortBy=file:modified
```

次に、2 番目のオプションとして名前でグループ化し、同じ変更を持つファイルは名前で並べ替えます。

```
sortBy=file:modified;file:name
```

これで問題がありましたが、その問題を見つけられましたか？ファイルの変更タイムスタンプはミリ秒単位で行われるため問題ありませんが、日付のみでソートし、名前でサブグループのサブグループを行う場合はどうなるか？
また、File Language の本当の機能があり、パターンをサポートする date コマンドを使用できます。そのため、以下のように解決できます。

```
sortBy=date:file:yyyyMMdd;file:name
```

Yeah、これは非常に強力で、グループごとに逆が可能であるため、ファイル名を元に戻すことができます。

```
sortBy=date:file:yyyyMMdd;reverse:file:name
```

100.26. USING GENERICFILEPROCESSSTRATEGY

processStrategy オプションを使用して、独自のカスタム GenericFileProcessStrategy を使用できます。これにより、独自の 開始、コミット、ロールバック ロジックを実装できます。
たとえば、システムが使用するフォルダーにファイルを書き込むことを仮定します。ただし、別の 準備済みファイルも書き込まれる前に、ファイルを使い始めるべきではありません。

そのため、独自の GenericFileProcessStrategy を実装することで、以下のように実装できます。

- begin（） メソッドでは、特別な 準備 済みファイルが存在するかどうかをテストできます。begin メソッドは、ファイルを使用するかどうかを示す ブール値 を返します。
- abort（）メソッド(Camel 2.10)では、start 操作で false が返された場合（リソースのクリーンアップなど）、特別なロジックを実行できます。
- commit（） メソッドで実際のファイルを移動し、準備状態 にあるファイルを削除することもできます。

100.27. フィルターの使用

filter オプションを使用すると、org.apache.camel.component.file.GenericFileFilter インターフェースを実装して、カスタムフィルターを Java コードに実装できます。このインターフェースには、ブール値を返す accept メソッドがあります。true を返し、ファイルを含めるには false を返し、ファイルを省略する場合は false を返します。Camel 2.10 以降では、ファイルがディレクトリーであるかどうかに関わらず、GenericFile に isDirectory メソッドがあります。これにより、不要なディレクトリーにトラバースしないように、不要なディレクトリーにフィルターを設定できます。

たとえば、名前に「skip 」で始まるディレクトリーを省略 するには、以下のように実装できます。

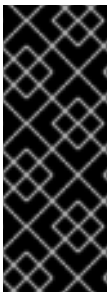
100.28. CONSUMER.BRIDGEERRORHANDLER の使用

Camel 2.10 で利用可能

Camel Error Handler を使用してファイルコンシューマーに発生した例外を処理する場合は、以下のように consumer.bridgeErrorHandler オプションを有効にできます。

```
// to handle any IOException being thrown
onException(IOException.class)
    .handled(true)
    .log("IOException occurred due: ${exception.message}")
    .transform().simple("Error ${exception.message}")
    .to("mock:error");

// this is the file route that pickup files, notice how we bridge the consumer to use the Camel routing error handler
// the exclusiveReadLockStrategy is only configured because this is from an unit test, so we use that to simulate exceptions
from("file:target/nospace?consumer.bridgeErrorHandler=true")
    .convertBodyTo(String.class)
    .to("mock:result");
```

そのため、このオプションを有効にすると、ルートのエラーハンドラーがそこから取得されます。



重要

consumer.bridgeErrorHandler 使用時 の consumer.bridgeErrorHandler を使用する場合、インターセプターは適用され ません。Exchange は Camel Error Handler によって直接処理され、onCompletion ではインターセプターなどの以前のアクションを許可しません。

100.29. デバッグロギング

このコンポーネントにはログレベルの TRACE があり、問題がある場合に役立ちます。

100.30. 関連項目

- *File 言語*
- *FTP*
- *Polling Consumer*

第101章 FILE 言語

Camel バージョン 1.1 で利用可能

*INFO:*File 言語が Simple 言語* From Camel 2.2 以降にマージされるようになり、File 言語は Simple 言語にマージされ、Simple 言語内ですべてのファイル構文を直接使用できるようになりました。*

File Expression Language は Simple 言語の拡張であり、ファイル関連の機能を追加します。これらの機能は、ファイルパスおよび名前を使用する一般的なユースケースに関連します。目的は、コンシューマーとプロデューサーの両方に動的ファイルパターンを設定するために、File コンポーネントおよび FTP コンポーネントと式を使用できるようにすることです。

101.1. FILE LANGUAGE オプション

File 言語は、以下に示す 2 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
resultType		**文字列**	結果タイプのクラス名を設定します（出力のタイプ）。
trim	**true**	**ブール値**	値をトリミングして先頭および末尾の空白と改行を削除するかどうか。

101.2. 構文

この言語は Simple 言語の 拡張 であるため、Simple 構文も適用されます。したがって、以下の表には追加としてのみ記載しています。
Simple 言語 ファイル言語とは対照的に、Constant 式 もサポートしているため、固定されたファイル名を入力できます。

すべてのファイルトークンは java.io.File オブジェクトのメソッドと同じ式名を使用します。たとえば、file:absolute は java.io.File.getAbsolute() メソッドを参照します。現在のエクスチェンジですべての式がサポートされているわけではないことに注意してください。たとえば、File コンポーネントはこれらすべてをサポートするため、FTP コンポーネントは一部のオプションをサポートします。

式	Type	ファイルコンシューマー	ファイルプロデューサー	FTP コンシューマー	FTP プロデューサー	説明
file:name	文字列	はい	いいえ	はい	いいえ	ファイル名を参照します（開始ディレクトリーに対する相対パス）。以下の注記を参照してください。
file:name.ext	文字列	はい	いいえ	はい	いいえ	**Camel 2.3:** ファイルエクステンションのみを参照します。
file:name.ext.single	文字列	はい	いいえ	はい	いいえ	**Camel 2.14.4/2.15.3**: ファイルエクステンションを参照します。ファイル拡張子にドットがある場合、この式は削除され、最後の部分のみを返します。
file:name.noext	文字列	はい	いいえ	はい	いいえ	拡張子のないファイル名を参照します（開始ディレクトリーとの相対パス）。以下の注記を参照してください。
file:name.noext.single	文字列	はい	いいえ	はい	いいえ	**Camel 2.14.4/2.15.3: エクステンションのないファイル名を参照します**（開始ディレクトリーに対する相対パス）。以下の注記を参照してください。ファイル拡張子に複数のドットがある場合、この変数は最後の部分のみを取り除き、他の部分を返します。
file:onlyname	文字列	はい	いいえ	はい	いいえ	先頭のパスのないファイル名のみを参照します。
file:onlyname.noext	文字列	はい	いいえ	はい	いいえ	拡張子なし、先頭のパスのないファイル名のみを参照します。

式	Type	ファイルコンシューマー	ファイルプロデューサー	FTP コンシューマー	FTP プロデューサー	説明
file:onlyname.noext.single	文字列	はい	いいえ	はい	いいえ	*Camel 2.14.4/2.15.3:* 拡張子なし、先頭のパスなしのみのファイル名を参照します。ファイル拡張子に複数のドットがある場合、この変数は最後の部分のみを取り除き、他の部分を返します。
file:ext	文字列	はい	いいえ	はい	いいえ	ファイル拡張子のみを参照します。
file:parent	文字列	はい	いいえ	はい	いいえ	ファイルの親を参照します。
file:path	文字列	はい	いいえ	はい	いいえ	ファイルパスを参照します。
file:absolute	ブール値	はい	いいえ	いいえ	いいえ	ファイルが絶対または相対として考慮されるかどうかを示します。
file:absolute.path	文字列	はい	いいえ	いいえ	いいえ	絶対パスを参照します。
file:length	Long	はい	いいえ	はい	いいえ	長期タイプとして返されるファイルの長さを参照します。
file:size	Long	はい	いいえ	はい	いいえ	**Camel 2.5:** Long 型として返されるファイルの長さを参照します。
file:modified	Date	はい	いいえ	はい	いいえ	最後に変更したファイルを日付タイプとして参照します。

式	Type	ファイルコンシューマー	ファイルプロデューサー	FTP コンシューマー	FTP プロデューサー	説明
date:_command:pattern_	文字列	はい	はい	はい	はい	**java.text.SimpleDateFormat** パターンを使用した日付形式の場合。Simple 言語の **拡張** です。追加コマンドは、**ファイル** の最後に変更されたタイムスタンプのファイル（コンシューマーのみ）です。注記： Simple 言語のすべてのコマンドも使用できます。

101.3. ファイルトークンの例

101.3.1. 相対パス

以下の 相対 ディレクトリーに hello.txt ファイルの java.io.File ハンドルがあります： .\filelanguage\testそして、この開始ディレクトリー .\filelanguage を使用するようにエンドポイントを設定します。ファイルトークンは以下のように返されます。

式	戻り値
file:name	test\hello.txt
file:name.ext	txt
file:name.noext	test\hello
file:onlyname	hello.txt
file:onlyname.noext	hello
file:ext	txt
file:parent	filelanguage\test
file:path	filelanguage\test\hello.txt
file:absolute	false

式	戻り値
file:absolute.path	\workspace\camel\camel-core\target\filelanguage\test\hello.txt

101.3.2. 絶対パス

\workspace\camel\camel-core\target\filelanguage\test のよう に、hello.txt ファイルの java.io.File が処理しています。また、エンドポイントの絶対起動ディレクトリー \workspace\camel\camel-core\target\filelanguage を使用するようにエンドポイントを設定します。ファイルトークンは以下のように返されます。

式	戻り値
file:name	test\hello.txt
file:name.ext	txt
file:name.noext	test\hello
file:onlyname	hello.txt
file:onlyname.noext	hello
file:ext	txt
file:parent	\workspace\camel\camel-core\target\filelanguage\test
file:path	\workspace\camel\camel-core\target\filelanguage\test\hello.txt
file:absolute	true
file:absolute.path	\workspace\camel\camel-core\target\filelanguage\test\hello.txt

101.4. サンプル

myfile.txt などの固定 Constant 式を入力できます。

```
fileName="myfile.txt"
```

こうすることで、ファイルコンシューマーを使用してファイルを読み取って、現在の日付をサブフォルダーとして使用してバックアップフォルダーに読み取りファイルを移動したいと仮定します。これは、以下のような式を使用してアーカイブできます。

```
fileName="backup/${date:now:yyyyMMdd}/${file:name.noext}.bak"
```

相対フォルダー名もサポートされています。そのため、バックアップフォルダーをシブリングフォルダーにする必要がある場合は、以下のように追加できます。

```
fileName="../backup/${date:now:yyyyMMdd}/${file:name.noext}.bak"
```

これは Simple 言語の拡張であるため、この言語からのすべての良い点にもアクセスできます。そのため、今回のユースケースでは、**in.header.type** を動的式のパラメーターとして使用します。

```
fileName="../backup/${date:now:yyyyMMdd}/type-${in.header.type}/backup-of-${file:name.noext}.bak"
```

式で使用するカスタムの日付がある場合、**Camel** はメッセージヘッダーから日付を取得することをサポートします。

```
fileName="orders/order-${in.header.customerId}-${date:in.header.orderDate:yyyyMMdd}.xml"
```

最後に、**Bean** 式を使用して、使用する **String** 出力を生成する **POJO** クラスを呼び出すこともできます（または **String** に変換されます）。

```
fileName="uniquefile-${bean:myguidgenerator.generateid}.txt"
```

当然ながら、これは 1 つの式に統合できます。この式では、File Language、Simple、および Bean 言語を 1 つの式で使用できます。これは、このような一般的なファイルパスパターンで非常に強力です。

101.5. SPRING PROPERTYPLACEHOLDERCONFIGURER と FILE コンポーネントの使用

Camel では、Simple 言語から直接 File 言語を使用することができます。これにより、**Spring XML** で **Content Based Router** を簡単に実行できます。ここでは、以下のようにファイルエクステンションに基づいてルーティングすることができます。

```
<from uri="file://input/orders"/>
  <choice>
    <when>
      <simple>${file:ext} == 'txt'</simple>
```

```
        <to uri="bean:orderService?method=handleTextFiles"/>
      </when>
      <when>
        <simple>${file:ext} == 'xml'</simple>
        <to uri="bean:orderService?method=handleXmlFiles"/>
      </when>
      <otherwise>
        <to uri="bean:orderService?method=handleOtherFiles"/>
      </otherwise>
    </choice>
```

File エンドポイントの fileName オプションを使用して File 言語を使用して動的ファイル 名を設定する場合は、
の代替構文（Camel 2.5 以降で利用可能）を使用して Springs PropertyPlaceholderConfigurer との競合を避けるようにしてください。

bundle-context.xml

```
<bean id="propertyPlaceholder"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location" value="classpath:bundle-context.cfg" />
</bean>

<bean id="sampleRoute" class="SampleRoute">
    <property name="fromEndpoint" value="${fromEndpoint}" />
    <property name="toEndpoint" value="${toEndpoint}" />
</bean>
```

bundle-context.cfg

```
fromEndpoint=activemq:queue:test
toEndpoint=file://fileRoute/out?fileName=test-$simple{date:now:yyyyMMdd}.txt
```

上記の toEndpoint で $simple\{ } 構文を使用することに注意してください。
これを行わないと、競合が発生し、Spring は次のような例外をスローします。

```
org.springframework.beans.factory.BeanDefinitionStoreException:
Invalid bean definition with name 'sampleRoute' defined in class path resource
[bundle-context.xml]:
Could not resolve placeholder 'date:now:yyyyMMdd'
```

101.6. 依存関係

File 言語は camel-core の一部です。

第102章 FLATPACK コンポーネント

Camel バージョン 1.4 で利用可能

Flatpack コンポーネントは、FlatPack ライブラリー を使用して、固定幅および区切られたファイル解析をサポートします。
注記： このコンポーネントは、flatpack ファイルからオブジェクトモデルへの消費のみをサポートします。オブジェクトモデルから flatpack 形式への書き込みはできません。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-flatpack</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

102.1. URI 形式

```
flatpack:[delim|fixed]:flatPackConfig.pzmap.xml[?options]
```

または、設定ファイルを使用しないカンマ区切りのファイルハンドラーの場合は、単に使用される。

```
flatpack:someName[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

102.2. URI オプション

Flatpack コンポーネントにはオプションがありません。

Flatpack エンドポイントは、URI 構文を使用して設定します。

```
flatpack:type:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

102.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
type	固定または区切り文字を使用するかどうか。	delim	FlatpackType
resourceUri	クラスパスまたはファイルシステムから flatpack マッピングファイルを読み込むのに **必要な** URL		文字列

102.2.2. クエリーパラメーター（25 パラメーター）：

Name	説明	デフォルト	Type
allowShortLines (common)	行が予想よりも短くし、追加文字を無視することができます。	false	boolean
delimiter (common)	コンマ区切りファイルのデフォルトの文字区切り文字。	,	char
ignoreExtraColumns (common)	行が予想よりも長くなり、追加文字を無視することができます。	false	boolean
ignoreFirstRecord (common)	区切ったファイル（列ヘッダーの場合）で最初の行を無視するかどうか。	true	boolean
splitRows (common)	解析後に各行を個別のエクスチェンジとして送信するコンポーネントを設定します。	true	boolean
textQualifier (common)	コンマ区切りファイルのテキスト修飾子。		char
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long

Name	説明	デフォルト	Type
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

102.3. 例

- ***flatpack:fixed:foo.pzmap.xml は、foo.pzmap.xml ファイル設定を使用して固定幅エンドポイントを作成します。***

- ***flatpack:delim:bar.pzmap.xml は、bar.pzmap.xml ファイル設定を使用して区切られたエンドポイントを作成します。***

- *flatpack:foo は、ファイル設定なしで foo という名前のエンドポイントを作成します。*

102.4. メッセージヘッダー

Camel は以下のヘッダーを IN メッセージに保存します。

ヘッダー	説明
camelFlatpackCounter	現在の行のインデックス。**splitRows=false** の場合、カウンターは行の合計数です。

102.5. メッセージボディー

コンポーネントは、java.util.Map または java.util.List のコンバーターが含まれる org.apache.camel.component.flatpack.DataSetList オブジェクトとして IN メッセージ内のデータを提供します。
通常、マップは一度に 1 行を処理する場合(splitRows=true)します。コンテンツ全体 (splitRows=false)に List を使用します。リストの各要素は Map です。
各 マップに は、列名と対応する値のキーが含まれます。

たとえば、以下の例から firstname を取得するには、以下を実行します。

```
Map row = exchange.getIn().getBody(Map.class);
String firstName = row.get("FIRSTNAME");
```

ただし、splitRows=trueであっても、常にこれを List として取得することもできます。同じ例は次のとおりです。

```
List data = exchange.getIn().getBody(List.class);
Map row = (Map)data.get(0);
String firstName = row.get("FIRSTNAME");
```

102.6. ヘッダーおよびトレイレコード

Flatpack でのヘッダーおよび証跡の表記に対応しています。ただし、固定レコード ID を使用する 必要があり ます。

- ヘッダー レコードのヘッダー（小文字である必要があります）

- trailer レコードのトラッカー（小文字でなければなりません）

以下の例は、ヘッダーとコントラiler があるという事実を示しています。必要でない場合は、いずれかの値または両方を省略できます。

```
<RECORD id="header" startPosition="1" endPosition="3" indicator="HBT">
    <COLUMN name="INDICATOR" length="3"/>
    <COLUMN name="DATE" length="8"/>
</RECORD>

<COLUMN name="FIRSTNAME" length="35" />
<COLUMN name="LASTNAME" length="35" />
<COLUMN name="ADDRESS" length="100" />
<COLUMN name="CITY" length="100" />
<COLUMN name="STATE" length="2" />
<COLUMN name="ZIP" length="5" />

<RECORD id="trailer" startPosition="1" endPosition="3" indicator="FBT">
    <COLUMN name="INDICATOR" length="3"/>
    <COLUMN name="STATUS" length="7"/>
</RECORD>
```

102.7. エンドポイントの使用

一般的なユースケースは、別のルートでさらに処理するために、このエンドポイントにファイルを送信します。以下に例を示します。

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="file://someDirectory"/>
    <to uri="flatpack:foo"/>
  </route>

  <route>
    <from uri="flatpack:foo"/>
    ...
  </route>
</camelContext>
```

簡単な Bean インテグレーションのために作成された各メッセージのペイロードを Map に変換することもできます。

102.8. FLATPACK DATAFORMAT

Flatpack コンポーネントには Flatpack データ形式が同梱されており、固定の幅または区切ったテキストメッセージ間のフォーマットを Map として行の リストに 使用することができます。

- *marshal = from List<Map<String, Object& gt;> から OutputStream （ Stringに変換可能）*
- *unmarshal = java .io.InputStream （ File または Stringなど）から java.apache.camel.component.flatpack.DataSetList インスタンスとしての java.util.List へのアンマーシャリング =*
  操作の結果には、すべてのデータが含まれます。各行を処理する必要がある場合は、Splitter を使用してエクスチェンジを分割できます。

注記： Flatpack ライブラリーは現在、マーシャリング操作のヘッダーおよび証跡をサポートしていません。

102.9. オプション

データフォーマットには以下のオプションがあります。

オプション	デフォルト	説明
定義	**null**	flatpack pzmap 設定ファイル。簡単な状況では省略できますが、パズマップの使用が推奨されます。
固定:	**false**	区切りまたは修正。
ignoreFirstRecord	**true**	区切ったファイル（列ヘッダーの場合）で最初の行を無視するかどうか。
textQualifier	**"**	テキストが「 などの文字 」 で修飾されている場合は、
delimiter	**,**	区切り文字の文字（could be **;**、 または同様のもの）

オプション	デフォルト	説明
parserFactory	**null**	デフォルトの Flatpack パーサーファクトリーを使用します。
allowShortLines	**false**	Camel 2.9.7 および 2.10.5 以降 では、行が予想よりも短くし、追加文字を無視します。
ignoreExtraColumns	**false**	Camel 2.9.7 以降 2.10.5 以降: 行が予想よりも長くなり、追加文字を無視することができます。

102.10. 用途

データフォーマットを使用するには、インスタンスをインスタンス化し、ルートビルダーでマーシャリングまたはアンマーシャリング操作を呼び出します。

```
FlatpackDataFormat fp = new FlatpackDataFormat();
fp.setDefinition(new ClassPathResource("INVENTORY-Delimited.pzmap.xml"));
...
from("file:order/in").unmarshal(df).to("seda:queue:neworder");
```

上記の例では、order/in フォルダーからファイルを読み込み、ファイルの構造を設定する Flatpack 設定ファイル INVENTORY-Delimited.pzmap.xml を使用して入力をアンマーシャリングします。結果は、SEDA キューに保存する DataSetList オブジェクトです。

```
FlatpackDataFormat df = new FlatpackDataFormat();
df.setDefinition(new ClassPathResource("PEOPLE-FixedLength.pzmap.xml"));
df.setFixed(true);
df.setIgnoreFirstRecord(false);

from("seda:people").marshal(df).convertBodyTo(String.class).to("jms:queue:people");
```

上記のコードでは、オブジェクト表現のデータをマップとして List としてマーシャルし ます。Map の行には、キーと対応する値の列名が含まれます。この構造は、プロセッサーなどの Java コードで作成できます。Flatpack 形式に従ってデータをマーシャリングし、結果を String オブジェクトとして変換し、JMS キューに保存します。

102.11. 依存関係

camel ルートで Flatpack を使用するには、このデータ形式を実装する camel-flatpack の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-flatpack</artifactId>
  <version>x.x.x</version>
</dependency>
```

102.12. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第103章 FLATPACK DATAFORMAT

Camel バージョン 2.1 で利用可能

Flatpack コンポーネントには Flatpack データ形式が同梱されており、固定の幅または区切ったテキストメッセージ間のフォーマットを Map として行の リストに 使用することができます。

- *marshal = from List<Map<String, Object& gt;> から OutputStream （ Stringに変換可能）*
- *unmarshal = java .io.InputStream （ File または Stringなど）から java.apache.camel.component.flatpack.DataSetList インスタンスとしての java.util.List へのアンマーシャリング =*
操作の結果には、すべてのデータが含まれます。各行を処理する必要がある場合は、Splitter を使用してエクスチェンジを分割できます。

注記： Flatpack ライブラリーは現在、マーシャリング操作のヘッダーおよび証跡をサポートしていません。

103.1. オプション

Flatpack データフォーマットは、以下に示す 9 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
定義		**文字列**	flatpack pzmap 設定ファイル。簡単な状況では省略できますが、パズマップの使用が推奨されます。
固定:	**false**	**ブール値**	区切りまたは修正。デフォルトは false = delimited です。
ignoreFirstRecord	**true**	**ブール値**	区切ったファイル（列ヘッダーの場合）で最初の行を無視するかどうか。デフォルトは true です。
textQualifier		**文字列**	テキストが文字で修飾される場合。デフォルトでは引用符を使用します。
delimiter	,	**文字列**	区切り文字の文字（could be ;、または同様のもの）

Name	デフォルト	Java タイプ	説明
allowShortLines	**false**	**ブール値**	行が予想よりも短くし、追加文字を無視することができます。
ignoreExtraColumns	**false**	**ブール値**	行が予想よりも長くなり、追加文字は無視されます。
parserFactoryRef		**文字列**	レジストリーでルックアップするカスタムパーサーファクトリーへの参照
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

103.2. 用途

データフォーマットを使用するには、インスタンスをインスタンス化し、ルートビルダーでマーシャリングまたはアンマーシャリング操作を呼び出します。

```
FlatpackDataFormat fp = new FlatpackDataFormat();
fp.setDefinition(new ClassPathResource("INVENTORY-Delimited.pzmap.xml"));
...
from("file:order/in").unmarshal(df).to("seda:queue:neworder");
```

上記の例では、order/in フォルダーからファイルを読み込み、ファイルの構造を設定する Flatpack 設定ファイル INVENTORY-Delimited.pzmap.xml を使用して入力をアンマーシャリングします。結果は、SEDA キューに保存する DataSetList オブジェクトです。

```
FlatpackDataFormat df = new FlatpackDataFormat();
df.setDefinition(new ClassPathResource("PEOPLE-FixedLength.pzmap.xml"));
df.setFixed(true);
df.setIgnoreFirstRecord(false);

from("seda:people").marshal(df).convertBodyTo(String.class).to("jms:queue:people");
```

上記のコードでは、オブジェクト表現のデータをマップとして List としてマーシャルし ます。Map の行には、キーと対応する値の列名が含まれます。この構造は、プロセッサーなどの Java コードで作成できます。Flatpack 形式に従ってデータをマーシャリングし、結果を String オブジェクトとして変換し、JMS キューに保存します。

103.3. 依存関係

camel ルートで Flatpack を使用するには、このデータ形式を実装する camel-flatpack の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-flatpack</artifactId>
  <version>x.x.x</version>
</dependency>
```

第104章 APACHE FLINK コンポーネント

Camel バージョン 2.18 から利用可能

本ドキュメントページでは、Apache Camel の Apache Flink コンポーネントについて説明します。camel-flink コンポーネントは、Camel コネクターと Flink タスク間のブリッジを提供します。この Camel Flink コネクターは、さまざまなトランスポートからメッセージをルーティングし、実行するフラッシュタスクを動的に選択して、タスクの入力データとして受信メッセージを使用し、最後に Camel パイプラインに結果を配信する方法を提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-flink</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

104.1. URI 形式

現在、Flink コンポーネントはプロデューサーのみをサポートします。DataSet、DataStream ジョブを作成できます。

```
flink:dataset?dataset=#myDataSet&dataSetCallback=#dataSetCallback
flink:datastream?datastream=#myDataStream&dataStreamCallback=#dataStreamCallback
```

FlinkEndpoint オプション

Apache Flink エンドポイントは、URI 構文を使用して設定します。

```
flink:endpointType
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

104.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
endpointType	**エンド** ポイントのタイプ（データセット、データストリーム）		EndpointType

104.1.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
collect（プロデューサー）	結果を収集またはカウントするかどうかを示します。	true	boolean
dataSet (producer)	計算先となるデータセット。		DataSet
dataSetCallback (producer)	DataSet に対するアクションの実行関数。		DataSetCallback
dataStream (producer)	計算対象となるデータストリーム。		DataStream
dataStreamCallback (producer)	DataStream に対してアクションを実行する関数。		DataStreamCallback
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

104.2. FLINKCOMPONENT オプション

Apache Flink コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
dataSet (producer)	計算先となるデータセット。		DataSet
dataStream (producer)	計算対象となるデータストリーム。		DataStream
dataSetCallback (producer)	DataSet に対するアクションの実行関数。		DataSetCallback

Name	説明	デフォルト	Type
dataStreamCallback (producer)	DataStream に対してアクションを実行する関数。		DataStreamCallback
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

104.3. FLINK DATASET コールバック

```
@Bean
public DataSetCallback<Long> dataSetCallback() {
    return new DataSetCallback<Long>() {
        public Long onDataSet(DataSet dataSet, Object... objects) {
            try {
                dataSet.print();
                return new Long(0);
            } catch (Exception e) {
                return new Long(-1);
            }
        }
    };
}
```

104.4. FLINK DATASTREAM コールバック

```
@Bean
public VoidDataStreamCallback dataStreamCallback() {
    return new VoidDataStreamCallback() {
        @Override
        public void doOnDataStream(DataStream dataStream, Object... objects) throws Exception
{
            dataStream.flatMap(new Splitter()).print();

            environment.execute("data stream test");
        }
    };
}
```

104.5. CAMEL-FLINK プロデューサー呼び出し

```
CamelContext camelContext = new SpringCamelContext(context);

String pattern = "foo";

try {
    ProducerTemplate template = camelContext.createProducerTemplate();
```

```
    camelContext.start();
    Long count = template.requestBody("flink:dataSet?
dataSet=#myDataSet&dataSetCallback=#countLinesContaining", pattern, Long.class);
    } finally {
        camelContext.stop();
    }
```

104.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第105章 FOP コンポーネント

Camel バージョン 2.10 で利用可能

FOP コンポーネントを使用すると、Apache FOP を使用してメッセージを異なる出力形式にレンダリングできます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-fop</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

105.1. URI 形式

```
fop://outputFormat?[options]
```

105.2. 出力形式

主な出力形式は PDF ですが、他の出力形式もサポートされています。http://xmlgraphics.apache.org/fop/0.95/output.html

name	outputFormat	説明
PDF	application/pdf	移植可能なドキュメント形式
PS	application/postscript	Adobe Postscript
PCL	application/x-pcl	プリンター制御言語
PNG	image/png	PNG イメージ

name	output Format	説明
JPEG	image/jpeg	JPEG イメージ
SVG	image/svg+xml	スケーラブルなベクターグラフ
XML	application/X-fop-areatree	エリアツリー表現
MIF	application/mif	FrameMaker's MIF
RTF	application/rtf	リッチテキスト形式
TXT	text/plain	テキスト

有効な出力形式の完全なリストは、こちらを参照してください。

105.3. エンドポイントオプション

FOP コンポーネントにはオプションがありません。

FOP エンドポイントは、URI 構文を使用して設定します。

```
fop:outputType
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

105.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
outputType	プライマリー出力形式は PDF ですが、他の出力形式もサポートされます。		FopOutputType

105.3.2. クエリーパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
fopFactory (producer)	カスタムの設定または org.apache.fop.apps.FopFactory の実装を使用できます。		FopFactory
userConfigURL (producer)	クラスパスまたはファイルシステムからロードできる設定ファイルの場所。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

以下の 構造 を持つ設定ファイルの場所。Camel 2.12 以降では、ファイルはデフォルトでクラスパスから読み込まれます。file:、または classpath: をプレフィックスとして使用して、ファイルまたはクラスパスからリソースを読み込むことができます。以前のリリースでは、ファイルは常にファイルシステムからロードされていました。

fopFactory

カスタムの設定または org.apache.fop.apps.FopFactory の実装を使用できます。

105.4. メッセージ操作

name	デフォルト値	説明
CamelFop.Output.Format		そのメッセージの出力形式を上書きします。
CamelFop.Encrypt.userPassword		PDF ユーザーのパスワード
CamelFop.Encrypt.ownerPassword		PDF 所有者の乗車語
CamelFop.Encrypt.allowPrint	true	PDF の印刷が可能
CamelFop.Encrypt.allowCopyContent	true	PDF の内容のコピーが可能
CamelFop.Encrypt.allowEditContent	true	PDF の内容を編集できます。
CamelFop.Encrypt.allowEditAnnotations	true	PDF のアノテーションを編集できます。
CamelFop.Render.producer	Apache FOP	ドキュメントを生成するシステム/ソフトウェアのメタデータ要素

name	デフォルト値	説明
CamelFop.Render.creator		ドキュメントを作成したユーザーのメタデータ要素
CamelFop.Render.creationDate		作成日
CamelFop.Render.author		author of the content of the document
CamelFop.Render.title		ドキュメントのタイトル
CamelFop.Render.subject		ドキュメントの件名
CamelFop.Render.keywords		本書に適用されるキーワードのセット

105.5. 例

以下は、xml データおよび xslt テンプレートから PDF をレンダリングし、ターゲットフォルダーに PDF ファイルを保存するルートの例になります。

```
from("file:source/data/xml")
    .to("xslt:xslt/template.xsl")
    .to("fop:application/pdf")
    .to("file:target/data");
```

詳細情報は、これらのリソース...

105.6. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第106章 FREEMARKER コンポーネント

Camel バージョン 2.10 で利用可能

freemarker: コンポーネントは FreeMarker テンプレートを使用したメッセージの処理を可能にします。これは、Templating を使用してリクエストの応答を生成する場合に便利です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-freemarker</artifactId>
  <version>x.x.x</version> <!-- use the same version as your Camel core version -->
</dependency>
```

106.1. URI 形式

```
freemarker:templateName[?options]
```

templateName は、呼び出すテンプレートのクラスパスローカル URI、またはリモートテンプレートの完全な URL（例： file:///folder/myfile.ftl）です。

URI にクエリーオプションを追加するには、?option=value&option=value&...

106.2. オプション

Freemarker コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	既存の freemarker.template.Configuration インスタンスを設定として使用する場合。		設定
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Freemarker エンドポイントは、URI 構文を使用して設定します。

```
freemarker:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

106.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceUri	リソースへの **必須** パス。プレフィックス：classpath、file、http、ref、または bean. classpath、file、http は、これらのプロトコルを使用してリソースをロードします（classpath はデフォルト）。ref はレジストリーのリソースを検索します。Bean はリソースとして使用される Bean でメソッドを呼び出します。Bean には、ドットの後にメソッド名を指定できます（例：bean:myBean.myMethod）。		文字列

106.2.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
設定 （プロデューサー）	使用する Freemarker 設定を設定します。		設定
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
encoding （プロデューサー）	テンプレートファイルの読み込みに使用するエンコーディングを設定します。		文字列
templateUpdateDelay (producer)	読み込まれたテンプレートリソースがキャッシュに残る秒数。		int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

106.3. HEADERS

FreeMarker 評価中に設定されるヘッダーがメッセージに返され、ヘッダーとして追加されます。これにより、FreeMarker コンポーネントのメカニズムで、値をメッセージに返すことができます。

例： FreeMarker テンプレートで fruit のヘッダー値を設定します。

```
${request.setHeader('fruit', 'Apple')}
```

ヘッダー fruit が message.out.headers から利用できるようになりました。

106.4. FREEMARKER CONTEXT

Camel は FreeMarker コンテキスト（マップのみ）で交換情報を提供し ます。エクスチェンジは以下 の ように転送されます。

key	value
exchange	エクスチェンジ **自体**。
exchange.properties	エクスチェンジ **プロパティー**。
ヘッダー	In メッセージのヘッダー。
camelContext	Camel コンテキスト。
request	In メッセージ。
ボディー	In メッセージのボディー。
response	Out メッセージ（InOut メッセージ交換パターンのみ）。

Camel 2.14 以降、メッセージヘッダーに「CamelFreemarkerDataModel"」というキーを使用して、カスタムの FreeMarker コンテキストを設定できます。

```
Map<String, Object> variableMap = new HashMap<String, Object>();
variableMap.put("headers", headersMap);
variableMap.put("body", "Monday");
variableMap.put("exchange", exchange);
exchange.getIn().setHeader("CamelFreemarkerDataModel", variableMap);
```

106.5. ホットリロード

FreeMarker テンプレートリソースはデフォルトで、ファイルとクラスパスリソース（展開 jar）の両方でホットリロードでき ません。contentCache=false を設定すると、Camel はリソースをキャッシュせず、ホットリロードが有効になります。このシナリオは開発で使用できます。

106.6. 動的テンプレート

Camel は 2 つのヘッダーを提供し、テンプレートまたはテンプレートコンテンツ自体に異なるリソースの場所を定義できます。これらのヘッダーのいずれかが設定されている場合、Camel はこれを設定されたエンドポイントで使用します。これにより、ランタイム時に動的テンプレートを提供できます。

ヘッダー	Type	説明	バージョンのサポート
FreemarkerConstants.FREEMARKER_RESOURCE	org.springframework.core.io.Resource	テンプレートリソース	⇐ 2.1
FreemarkerConstants.FREEMARKER_RESOURCE_URI	文字列	設定されたエンドポイントの代わりに使用するテンプレートリソースの URI。	>= 2.1
FreemarkerConstants.FREEMARKER_TEMPLATE	文字列	設定されたエンドポイントの代わりに使用するテンプレート。	>= 2.1

106.7. サンプル

たとえば、以下のようなものを使用できます。

```
from("activemq:My.Queue").
  to("freemarker:com/acme/MyResponse.ftl");
```

FreeMarker テンプレートを使用して InOut メッセージエクスチェンジのメッセージの応答（JMSReplyTo ヘッダーがある）を作成するには、以下を実行します。

InOnly を使用してメッセージを消費し、別の宛先に送信する場合は、以下を使用できます。

```
from("activemq:My.Queue").
  to("freemarker:com/acme/MyResponse.ftl").
  to("activemq:Another.Queue");
```

コンテンツキャッシュを無効にするには、.ftl テンプレートがホットリロードされる必要のある開発の使用など、コンテンツキャッシュを無効にします。

```
from("activemq:My.Queue").
  to("freemarker:com/acme/MyResponse.ftl?contentCache=false").
  to("activemq:Another.Queue");
```

ファイルベースのリソースの場合：

```
from("activemq:My.Queue").
  to("freemarker:file://myfolder/MyResponse.ftl?contentCache=false").
  to("activemq:Another.Queue");
```

Camel 2.1 では、以下のように、コンポーネントがヘッダーを介して動的に使用するテンプレートを指定できます。

```
from("direct:in").

setHeader(FreemarkerConstants.FREEMARKER_RESOURCE_URI).constant("path/to/my/template.ftl").
  to("freemarker:dummy");
```

106.8. メールサンプル

この例では、FreeMarker テンプレートで注文の確認メールを使用します。メールテンプレートは、FreeMarker に以下のように配置されています。

```
Dear ${headers.lastName}, ${headers.firstName}

Thanks for the order of ${headers.item}.

Regards Camel Riders Bookstore
${body}
```

そして、java コードは次のようになります。

106.9. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第107章 FTP コンポーネント

Camel バージョン 1.1 で利用可能

このコンポーネントは、FTP プロトコルおよび SFTP プロトコルを介してリモートファイルシステムへのアクセスを提供します。

リモート FTP サーバーから使用する場合は、ファイルの消費に関する詳細については、以下の Default というセクションを読み込んでください。

絶対パスはサポートされて いません。Camel 2.16 は、ディレクトリー名 からすべての先頭のスラッシュをトリミングすることで、絶対パスを相対に変換します。ログに WARN メッセージが出力されます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ftp</artifactId>
  <version>x.x.x</version>See the documentation of the Apache Commons
  <!-- use the same version as your Camel core version -->
</dependency>
```

107.1. URI 形式

```
ftp://[username@]hostname[:port]/directoryname[?options]
sftp://[username@]hostname[:port]/directoryname[?options]
ftps://[username@]hostname[:port]/directoryname[?options]
```

directoryname は、基礎となるディレクトリーを表します。ディレクトリー名は相対パスです。絶対パスはサポートされて いません。相対パスには、/inbox/us などのネストされたフォルダーを含めることができます。

Camel 2.16 より前の Camel バージョンの場合、このコンポーネントは autoCreate オプション（ファイルコンポーネントの動作）をサポートしないため、directoryName がすでに存在している 必要 があります。このため、FTP 管理者（FTP サーバー）タスクがユーザーアカウントを適切に設定し、適切なファイルパーミッションを持つホームディレクトリーも正しく設定することです。

Camel 2.16 で は、 autoCreate オプションがサポートされています。コンシューマーが起動する

と、ポーリングがスケジュールされる前に、エンドポイントに設定されたディレクトリーを作成するために追加の FTP 操作が実行されます。autoCreate のデフォルト値は true です。

ユーザー名が指定 さ れていない場合、匿名 ログインはパスワードなしで試行されます。
ポート 番号が指定されていない場合、Camel はプロトコル（ftp = 21、ftp = 22、ftps = 2222）に応じてデフォルト値を提供します。

URI にクエリーオプションを追加するには、 ?option=value&option=value&…

このコンポーネントは、実際の FTP 作業に 2 つの異なるライブラリーを使用します。FTP および FTPS は Apache Commons Net を使用しますが、SFTP は JCraft JSCH を使用します。

FTPS コンポーネントは Camel 2.2 以降でのみ利用できます。
FTPS（FTP Secure とも呼ばれています）は、Transport Layer Security(TLS)および Secure Sockets Layer(SSL)暗号化プロトコルに対応する FTP の拡張機能です。

107.2. URI オプション

以下のオプションは、FTP コンポーネント専用です。

FTP コンポーネントにはオプションがありません。

FTP エンドポイントは、URI 構文を使用して設定されます。

```
ftp:host:port/directoryName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

107.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
host	FTP サーバーに 必要な ホスト名		文字列
port	FTP サーバーのポート		int

Name	説明	デフォルト	Type
directoryName	起動ディレクトリー		文字列

107.2.2. クエリーパラメーター（108 パラメーター）：

Name	説明	デフォルト	Type
バイナリー （共通）	ファイル転送モード（BINARY または ASCII）を指定します。デフォルトは ASCII(false)です。	false	boolean
charset (common)	このオプションは、ファイルのエンコーディングを指定するために使用されます。これはコンシューマーで使用して、ファイルのエンコーディングを指定できます。これにより、Camel はファイルの内容にアクセスした場合にファイルの内容を読み込む必要があります。ファイルを書き込む場合も同様に、このオプションを使用して、ファイルを書き込む文字を指定することもできます。ファイルを書くときには、メッセージの内容をメモリーに読み込んで、データを設定済みの文字セットに変換しなければならない場合があります。そのため、大きなメッセージがある場合は使用しないでください。		文字列
切断 （共通）	使用後にリモートの FTP サーバーから適切な接続を解除するかどうか。切断すると、現在の FTP サーバーへの接続が切断されるだけです。停止するコンシューマーがある場合は、代わりにコンシューマー/ルートを停止する必要があります。	false	boolean
doneFileName (common)	プロデューサー： 指定された場合、元のファイルが書き込まれると、Camel は 2 番目に完了したファイルを書き込みます。完了ファイルは空になります。このオプションは、使用するファイル名を設定します。固定名を指定できます。または、動的プレースホルダーを使用することもできます。done ファイルは、常に元のファイルと同じフォルダーに書き込まれます。コンシューマー： 提供された場合、Camel は完了したファイルが存在する場合にのみファイルを消費します。このオプションは、使用するファイル名を設定します。固定名を指定できます。または、動的プレースホルダーを使用することもできます。完成したファイルは、常に元のファイルと同じフォルダーに期待されます。$file.name および $file.name.noext のみが動的プレースホルダーとしてサポートされます。		文字列

Name	説明	デフォルト	Type
fileName (common)	File Language などの式を使用して、ファイル名を動的に設定します。コンシューマーの場合は、ファイル名フィルターとして使用されます。プロデューサーの場合、書き込みするファイル名を評価するために使用されます。式が設定されている場合は、CamelFileName ヘッダーよりも優先されます。（注記： ヘッダー自体は式にすることもできます。式オプションは String タイプと Expression 型の両方をサポートします。式が String タイプである場合、これは常に File 言語を使用して評価されます。式が Expression タイプである場合、指定された式タイプが使用されます。たとえば、OGNL 式を使用できます。コンシューマーの場合は、これを使用してファイル名を絞り込むことができます。そのため、インスタンスは File Language 構文 mydata-$date:yyyyMMdd.txt を使用して現在のファイルを使用できます。プロデューサーは、既存の CamelFileName ヘッダーよりも優先される CamelOverruleFileName ヘッダーをサポートします。CamelOverruleFileName は 1 度だけ使用されるヘッダーで、CamelFileName を一時的に保存し、後で復元する必要があるため、より簡単に使用できます。		文字列
passiveMode (common)	パッシブモード接続を設定します。デフォルトはアクティブモード接続です。	false	boolean
セパレーター （共通）	使用するパス区切り文字を設定します。UNIX = Uses unix スタイルの path separator Windows = Uses windows style path separator Auto =(is default)Use existing path separator in file name	UNIX	PathSeparator
transferLoggingInterval Seconds (common)	インフライトでアップロードおよびダウンロード操作の進捗をログに記録するときに使用する間隔を秒単位で設定します。これは、操作に時間がかかるときにロギングの進捗に使用されます。	5	int
transferLoggingLevel (common)	アップロードおよびダウンロード操作の進捗をログに記録するときに使用するロギングレベルを設定します。	DEBUG	LoggingLevel
transferLoggingVerbose (common)	アップロード操作およびダウンロード操作の進捗の進捗状況を（詳細にわたり）実行するかどうかを設定します。	false	boolean

Name	説明	デフォルト	Type
fastExistsCheck (common)	このオプションを true に設定すると、camel-ftp はリストファイルを直接使用してファイルが存在するかどうかを確認します。一部の FTP サーバーはファイルを直接一覧表示できない可能性があるため、オプションが false の場合、camel-ftp は古い方法でディレクトリーを一覧表示し、ファイルが存在するかどうかを確認します。また、このオプションは readLock=changed にも影響し、ファイル情報を更新する高速チェックを実行するかどうかを制御します。FTP サーバーに多くのファイルがある場合に、このプロセスを迅速化するために使用できます。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
Delete （コンシューマー）	true の場合、ファイルは正常に処理された後に削除されます。	false	boolean
moveFailed (consumer)	Simple 言語に基づいて移動失敗式を設定します。たとえば、.error サブディレクトリーにファイルを移動するには、.error を使用します。注記： 失敗した場所にファイルを移動すると Camel はエラーを処理し、再度ファイルを取得しません。		文字列
noop （コンシューマー）	true の場合、ファイルは移動または削除されません。このオプションは、読み取り専用データまたは ETL タイプの要件に適しています。noop=true の場合、Camel はべき等=true も設定し、同じファイルを何度も消費しないようにします。	false	boolean
preMove (consumer)	処理前にファイル名を動的に設定するために使用される式（ファイル言語など）。たとえば、進行中のファイルを order ディレクトリーに移動するには、この値を order に設定します。		文字列

Name	説明	デフォルト	Type
preSort (consumer)	pre-sort を有効にすると、コンシューマーはポーリング中にファイルとディレクトリー名をソートし、ファイルシステムから取得されたものになります。これは、ソートされた順序でファイルで操作する必要がある場合があります。pre-sort は、コンシューマーがフィルターをかけ、Camel が処理するファイルを受け入れる前に実行されます。このオプションは、無効を意味する default=false です。	false	boolean
再帰的 （コンシューマー）	ディレクトリーの場合は、すべてのサブディレクトリー内のファイルも検索します。	false	boolean
resumeDownload (consumer)	ダウンロードを有効にするかどうかを設定します。これは FTP サーバー（ほぼすべての FTP サーバーがそれをサポートしている）でサポートされている必要があります。さらに、ダウンロードされたファイルがローカルディレクトリーに保存されるように localWorkDirectory オプションを設定し、ダウンロードの再開をサポートするのに必要なオプションのバイナリーを有効にする必要があります。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
streamDownload (consumer)	ローカルの作業ディレクトリーを使用しない場合に使用するダウンロード方法を設定します。true に設定すると、リモートファイルは読み取り時にルートにストリーミングされます。false に設定すると、リモートファイルはルートに送信される前にメモリーに読み込まれます。	false	boolean
directoryMustExist (consumer)	startingDirectoryMustExist と同様ですが、再帰的なサブディレクトリーのポーリング時に適用されます。	false	boolean
download (consumer)	FTP コンシューマーがファイルをダウンロードするかどうか。このオプションを false に設定すると、メッセージボディーは null になりますが、コンシューマーはファイル名、ファイルサイズなどのファイルの詳細が含まれる Camel Exchange のトリガーになります。ファイルはダウンロードされないだけです。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
handleDirectoryParserAbsoluteResult (consumer)	ディレクトリーパーサーが絶対パスで結果となる場合は、コンシューマーがパス内のサブフォルダーとファイルを処理する方法を設定できます。そのため、一部の FTP サーバーは絶対パスでファイル名を返す可能性があるため、FTP コンポーネントが、返されるパスを相対パスに変換してこれを処理する必要があることです。	false	boolean
ignoreFileNotFoundOrPermissionError (consumer)	いつ無視するか（ディレクトリー内のファイルを一覧表示しようとするか、またはファイルをダウンロードする際に）無視するかどうか。これは存在しないか、パーミッションエラーが原因です。デフォルトでは、ディレクトリーまたはファイルが存在しないか、パーミッションが不十分な場合に、例外が発生します。このオプションを true に設定すると、代わりにこれを無視することができます。	false	boolean
inProgressRepository (consumer)	プラグ可能な in-progress リポジトリー org.apache.camel.spi.IdempotentRepository。進行中のリポジトリーは、現在使用中の進行中のファイルを考慮するために使用されます。デフォルトでは、メモリーベースのリポジトリーが使用されます。		String>
localWorkDirectory (consumer)	使用する場合は、ローカルの作業ディレクトリーを使用してリモートファイルの内容を直接ローカルファイルに保存し、コンテンツをメモリーにロードしないようにできます。これは、非常に大きなリモートファイルを使用しているため、メモリーを節約できることに有益です。		文字列
onCompletionException Handler (consumer)	カスタムの org.apache.camel.spi.ExceptionHandler を使用して、コンシューマーがコミットまたはロールバックを行う完了プロセス時に発生する例外を処理します。デフォルトの実装は、WARN レベルですべての例外をログに記録し、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processStrategy (consumer)	プラグ可能な org.apache.camel.component.file.GenericFileProcessStrategy を使用すると、独自の readLock オプションまたは同様のものを実装できます。また、特別な対応ファイルが存在するなど、ファイルを使用する前に特別な条件が満たされなければならない場合にも使用できます。このオプションを設定すると、readLock オプションが適用されません。		GenericFileProcess Strategy<T>
receiveBufferSize (consumer)	FTPClient でのみ使用される受信（ダウンロード）バッファーサイズ	32768	int
startingDirectoryMustExist (consumer)	開始ディレクトリーが存在するかどうか。autoCreate オプションはデフォルトで有効になっています。つまり、開始ディレクトリーが存在しない場合には、通常、起動ディレクトリーが作成されます。autoCreate を無効にして、これを有効にして、開始ディレクトリーが存在することを確認できます。ディレクトリーが存在しない場合は例外が発生します。	false	boolean
useList (consumer)	ファイルのダウンロード時に LIST コマンドの使用を許可するかどうか。デフォルトは true です。ユースケースによっては、特定のファイルをダウンロードし、LIST コマンドを使用できないことがあるため、このオプションを false に設定します。このオプションを使用すると、ダウンロードする特定のファイルには、ファイルサイズ、タイムスタンプ、パーミッションなどのメタデータ情報が含まれません。これらの情報は、LIST コマンドが使用されている場合のみ取得できるためです。	true	boolean

Name	説明	デフォルト	Type
fileExist (producer)	同じ名前のファイルがすでに存在する場合のアクション。デフォルトのオーバーライドは、既存のファイルを置き換えます。append: 既存のファイルにコンテンツを追加します。fail: GenericFileOperationException をスローし、既存のファイルがあることを示します。ignore: 問題を警告なしで無視し、既存のファイルは上書きしませんが、すべてが問題であることを前提としています。move - オプションでは、moveExisting オプションも設定する必要があります。eagerDeleteTargetFile オプションを使用して、ファイルの移動や既存のファイルがすでに存在する場合に実行内容を制御できます。それ以外の場合は、移動操作が失敗します。Move オプションは、ターゲットファイルを書き込む前に既存のファイルを移動します。TryRename は、tempFileName オプションが使用されている場合にのみ適用されます。これにより、存在チェックを実行せずに、一時的な名前から実際の名前へのファイルの名前変更を試すことができます。一部のファイルシステムや、特に FTP サーバーでは、このチェックの方が高速である場合があります。	オーバーライド	GenericFileExist
flatten (producer)	フラット化は、ファイル名パスをフラット化して先頭のパスを削除するために使用されるので、ファイル名だけになります。これにより、サブディレクトリーに再帰的に消費できますが、たとえば別のディレクトリーにファイルを書き込むと、それらのファイルは単一のディレクトリーに書き込まれます。これをプロデューサーで true に設定すると、CamelFileName ヘッダーのファイル名が任意の先頭パスに対して削除されます。	false	boolean
moveExisting (producer)	fileExist=Move が設定されている場合に使用するファイル名の計算に使用される式（ファイル言語など）。ファイルをバックアップサブディレクトリーに移動するには、バックアップを入力します。このオプションは、file:name、file:name.ext、file:name.noext、file:onlyname、file:onlyname.noext、file:ext、および file:parent の File Language トークンのみをサポートします。FTP コンポーネントは既存のファイルをベースとして、相対ディレクトリーにのみ移動できるため、file:parent は FTP コンポーネントではサポートされないことに注意してください。		文字列
tempFileName (producer)	tempPrefix オプションと同じですが、File 言語を使用する一時的なファイル名の命名をより細かく制御できます。		文字列

Name	説明	デフォルト	Type
tempPrefix (producer)	このオプションは、一時的な名前を使用してファイルを書き込むために使用されます。次に、書き込みが完了したら、その名前を変更します。書き込まれているファイルを特定し、進行中のファイルで読み取るコンシューマー（排他的読み取りロックを使用しない）も回避できます。大容量のファイルをアップロードする場合に FTP が使用することがよくあります。		文字列
allowNullBody (producer)	ファイルの書き込み中に null ボディーを許可するかどうかを指定するのに使用します。true に設定すると、空のファイルが作成され、false に設定され、null ボディーを file コンポーネントに送信しようとすると、'Cannot write null body to file.' の GenericFileWriteException がスローされます。fileExist オプションを 'Override' に設定すると、ファイルが切り捨てられます。追加する場合は、ファイルは変更されません。	false	boolean
chmod （プロデューサー）	保存したファイルで chmod を設定できます。例：chmod=640		文字列
disconnectOnBatchComplete (producer)	Batch アップロードの完了後にリモート FTP サーバーから右に切断するかどうか。disconnectOnBatchComplete は、現在の FTP サーバーの接続のみを切断します。	false	boolean
eagerDeleteTargetFile (producer)	既存のターゲットファイルを活発に削除するかどうか。このオプションは、fileExists=Override オプションおよび tempFileName オプションを使用している場合にのみ適用されます。このパラメーターを使用して、一時ファイルが書き込まれる前にターゲットファイルを削除する（false に設定します）ことができます。たとえば、大きなファイルを作成し、一時ファイルが書き込まれている間にターゲットファイルを存在させたいとします。これにより、一時ファイルの名前がターゲットファイル名に変更される直前に、ターゲットファイルが最後の時点までのみ削除されます。また、このオプションは、fileExist=Move が有効で、既存のファイルが存在すると、既存のファイルを削除するかどうかを制御することもできます。このオプションの copyAndDeleteOnRenameFails false を指定した場合、既存のファイルが存在する場合は、移動操作の前に既存のファイルが削除されます。	true	boolean

Name	説明	デフォルト	Type
keepLastModified (producer)	ソースファイルから最後に変更したタイムスタンプを保持します（存在する場合）。Exchange.FILE_LAST_MODIFIED ヘッダーを使用してタイムスタンプを見つけます。このヘッダーには java.util.Date またはタイムスタンプの long を含めることができます。タイムスタンプが存在し、オプションが有効な場合は、書き込まれたファイルにこのタイムスタンプを設定します。注記： このオプションは、ファイルプロデューサーにのみ適用されます。このオプションは、ftp プロデューサーでは使用できません。	false	boolean
sendNoop (producer)	FTP サーバーにファイルをアップロードする前に、事前書き込みチェックとして noop コマンドを送信するかどうか。これは、接続の検証がまだ有効であるため、デフォルトでは、ファイルのアップロードを警告なしで再接続できます。ただし、これにより問題が発生する場合は、このオプションをオフにすることができます。	true	boolean
activePortRange (advanced)	アクティブモードでクライアント側のポート範囲を設定します。構文： minPort-maxPort 両方のポート番号は含まれます。たとえば 10000-19999（すべての 1xxxx ポートを含める）です。		文字列
autocreate(advanced)	ファイルのパス名に不足しているディレクトリーを自動的に作成します。ファイルコンシューマーの場合は、開始ディレクトリーを作成することを意味します。ファイルプロデューサーの場合、ファイルが書き込まれるディレクトリーを意味します。	true	boolean
bufferSize (advanced)	バイト単位で書き込みバッファーサイズを書き込みます。	131072	int
connectTimeout (advanced)	FTPClient と JSCH の両方によって接続が確立されるまでの接続タイムアウトを設定します。	10000	int
ftpClient (advanced)	FTPClient のカスタムインスタンスを使用するには、以下を行います。		FTPClient
ftpClientConfig (advanced)	FTPClientConfig のカスタムインスタンスを使用して FTP クライアントを設定するには、エンドポイントを使用する必要があります。		FTPClientConfig
ftpClientConfigParameters (advanced)	FtpComponent が使用し、FTPClientConfig の追加パラメーターを提供します。		マップ

Name	説明	デフォルト	Type
ftpClientParameters (advanced)	FtpComponent が使用し、FTPClient の追加パラメーターを提供します。		マップ
maximumReconnectAttempts (advanced)	リモート FTP サーバーに接続しようとすると Camel が実行を試行する最大再接続試行を指定します。この動作を無効にするには 0 を使用します。		int
reconnectDelay (advanced)	Camel は再接続を試みる前に待機します。		Long
siteCommand (advanced)	ログインの成功後に実行する任意の site コマンドを設定します。複数のサイトコマンドは、改行文字を使用して分離できます。		文字列
soTimeout (advanced)	FTPClient でのみ使用されるようにタイムアウトを設定します。	300000	int
stepwise (advanced)	ファイルのダウンロード時、またはファイルのダウンロード時、またはディレクトリーにファイルをアップロードする場合に、ディレクトリーをステップ的に変更するかどうかを設定します。たとえば、セキュリティー上の理由から FTP サーバーのディレクトリーを変更できない場合は、これを無効にすることができます。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
throwExceptionOnConnect Failed (advanced)	接続に失敗した場合（すべて）に例外がスローされるかどうか。デフォルトでは例外はスローされず、WARN がログに記録されます。これを使用して、例外がスローされ、org.apache.camel.spi.PollingConsumerPollStrategy ロールバックメソッドから発生した例外を処理できます。	false	boolean
タイムアウト （詳細）	FTPClient のみで使用された応答を待機するデータタイムアウトを設定します。	30000	int
antExclude (filter)	ant スタイルのフィルターの除外。antInclude と antExclude の両方を使用する場合は、antInclude よりも antExclude が優先されます。複数の除外はコンマ区切りの形式で指定できます。		文字列
antFilterCaseSensitive (filter)	付与フィルターに大文字と小文字を区別するフラグを設定します。	true	boolean

Name	説明	デフォルト	Type
antInclude (filter)	ant スタイルのフィルターが含まれます。複数の包含は、コンマ区切りの形式で指定できます。		文字列
eagerMaxMessagesPerPoll (filter)	maxMessagesPerPoll の制限が Eager かどうかを制御できます。Eager が Eager の場合は、制限がファイルのスキャン中になります。false の場合、すべてのファイルをスキャンし、並び替えを実行します。このオプションを false に設定すると、すべてのファイルを最初にソートし、ポーリングを制限できます。ソートを実行するには、すべてのファイルの詳細がメモリー内にあるため、メモリー使用量を高くする必要があることに注意してください。	true	boolean
exclude （フィルター）	ファイルを除外するために使用されます（ファイル名が正規表現パターンと一致する場合）。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW（）構文を使用してこれを設定する必要があります。エンドポイント URI の設定に関する詳細は、「エンドポイント URI の設定」を参照してください。		文字列
Filter （フィルター）	プラグ可能なフィルター（org.apache.camel.component.file.GenericFileFilter クラス）フィルターが accept（）メソッドで false を返すと、ファイルをスキップします。		GenericFileFilter<T>
filterDirectory (filter)	Simple 言語に基づいてディレクトリーをフィルタリングします。たとえば、現在の日付でフィルタリングするには、$date:now:yyMMdd などの単純な日付パターンを使用できます。		文字列
filterFile (filter)	Simple 言語に基づいてファイルをフィルタリングします。たとえば、ファイルサイズで絞り込むには、$file:size 5000 を使用できます。		文字列
idempotent (filter)	Idempotent Consumer EIP パターンを使用して、Camel がすでに処理されたファイルをスキップするオプション。デフォルトでは、1000 エントリーを保持するメモリーベースの LRUCache を使用します。noop=true の場合は、同じファイルを何度も使用することを回避するため、べき等性も有効になります。	false	ブール値
idempotentKey (filter)	カスタムのべき等キーを使用するには、以下を行います。デフォルトでは、ファイルの絶対パスが使用されます。File 言語を使用すると、ファイル名とファイルサイズを使用できます (idempotentKey=$file:name-$file:size)。		文字列

Name	説明	デフォルト	Type
idempotentRepository (filter)	何も指定されておらず、べき等性が true の場合、デフォルトでは MemoryMessageIdRepository を使用するプラグ可能なリポジトリー org.apache.camel.spi.IdempotentRepository。		String>
include （フィルター）	ファイルを含めるために使用されます。ファイル名が正規表現パターンと一致する場合（一致する場合は大文字と小文字が区別されます）。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW（）構文を使用してこれを設定する必要があります。エンドポイント URI の設定に関する詳細は、「エンドポイント URI の設定」を参照してください。		文字列
maxDepth (filter)	ディレクトリーを再帰的に処理する際に通過する最大深度。	2147483647	int
maxMessagesPerPoll (filter)	ポーリングごとに収集する最大メッセージを定義します。デフォルトでは最大値は設定されません。1000 などの制限を設定して、数千のファイルがあるサーバーを起動すると回避できます。無効にするには、0 または negative の値を設定します。注記： このオプションが使用されている場合、File および FTP コンポーネントはソート前に制限されます。たとえば、100000 ファイルがあり、maxMessagesPerPoll=500 を使用する場合は、最初の 500 ファイルのみが選択され、その後にソートされます。eagerMaxMessagesPerPoll オプションを使用して、これを false に設定すると、最初にすべてのファイルをスキャンし、後でソートできます。		int
minDepth (filter)	ディレクトリーを再帰的に処理する際に処理を開始する最小深度。minDepth=1 を使用すると、ベースディレクトリーを意味します。minDepth=2 を使用すると、最初のサブディレクトリーを意味します。		int
Move （フィルター）	処理後にファイル名を動的に設定するために使用される式（Simple Language など）。ファイルを .done サブディレクトリーに移動するには、.done と入力します。		文字列
exclusiveReadLockStrategy (lock)	プラグ可能な read-lock を org.apache.camel.component.file.GenericFileExclusiveReadLockStrategy 実装とする。		GenericFileExclusive ReadLockStrategy <T>
readLock (lock)	ファイルに read-lock が排他的な場合にのみファイルをポーリングするために、コンシューマーによって使用されます（ファイルが進行中の場合や書き込みされていません）。Camel はファイルロックが許可	none	文字列

Name	説明	デフォルト	Type
	されるまで待機します。このオプションは、ストラテジーでビルドを提供します。none - No read lock is in use markerFile - Camel はマーカーファイル (fileName.camelLock)を作成し、そのロックを保持します。このオプションは、FTP コンポーネントの変更には使用できません。Changed は、ファイルの長さ/変更タイムスタンプを使用して、ファイルがすでにコピーされているかどうかを検出します。少なくとも 1 つのセクションを使用してこれを判断するため、このオプションはファイルを他のプロセスとしてすぐに消費できませんが、JDK IO API はファイルを別のプロセスで現在使用しているかどうかを判断することができないため、信頼性が高まります。readLockCheckInterval オプションを使用してチェック頻度を設定できます。fileLock - は java.nio.channels.FileLock 用です。このオプションは FTP コンポーネントでは使用できません。この方法は、ファイルシステムが分散ファイルロックに対応している場合を除き、マウント/共有経由でリモートファイルシステムにアクセスするときに回避する必要があります。名前変更は、読み取り専用で読み取りロックを取得できる場合に、テストとしてファイルの名前を変更するためのものです。べき等性 -（ファイルコンポーネントのみ）べき等性は、idempotentRepository を read-lock として使用することです。これにより、べき等リポジトリーの実装がサポートする場合、クラスタリングをサポートする読み取りロックを使用できます。idempotent-changed -（ファイルコンポーネントのみ）idempotent-changed はidempotentRepository を使用し、結合された read-lock として変更されます。これにより、べき等リポジトリー実装がこれをサポートする場合、クラスタリングをサポートする読み取りロックを使用できます。idempotent-rename -（ファイルコンポーネントのみ）idempotent-rename はidempotentRepository を使用し、結合された read-lock として名前を変更します。これにより、べき等リポジトリーの実装がサポートしている場合、クラスタリングをサポートする読み取りロックを使用できます。注記： 各種の読み取りロックは、クラスターモードですべて機能する訳ではありません。この場合、異なるノードの同時コンシューマーは共有ファイルシステムの同じファイルに対して競合しています。atomic 操作を使用して空のマーカーファイルを作成しますが、クラスター内での動作が保証されません。fileLock の方が良好に機能しますが、ファイルシステムは分散ファイルロックなどに対応する必要があります。べき等リポジトリーが Hazelcast コンポーネントや Infinispan などのクラスタリングに対応している場合、べき等リポジトリーがクラスタリングをサポートする場合は、べき等読み取りロックを使用できます。		

Name	説明	デフォルト	Type
readLockCheckInterval (lock)	読み取りロックでサポートされている場合は、read-lock の間隔（ミリ秒単位）。この間隔は、読み取りロックの取得を試みる間にスリープするために使用されます。たとえば、変更した読み取りロックを使用する場合は、低速な書き込みのために、間隔を cater に設定できます。デフォルトの 1 秒。プロデューサーがファイルを書き込む速度が非常に遅い場合は、速度が長すぎる可能性があります。注記：FTP の場合、デフォルトの readLockCheckInterval は 5000 です。readLockTimeout の値は readLockCheckInterval よりも大きくなければなりませんが、thumb のルールは readLockCheckInterval よりも少なくとも 2 倍以上タイムアウトになるようにします。これは、読み取りロックプロセスでタイムアウトがヒットするまでにロックの取得を試行するために バブル時間が許可されるようにするために必要です。	1000	Long
readLockDeleteOrphanLock Files (lock)	Camel が適切にシャットダウンされない場合（JVM クラッシュなど）、ファイルシステム上に残っている可能性のある孤立した読み取りロックファイルを削除する際にマーカーファイルを使用したロックを削除するかどうか。このオプションを false に指定すると、孤立したロックファイルがあると、Camel がそのファイルを取得しようとしません。また、別のノードが同じ共有ディレクトリーからファイルを同時に読み取ることが原因として考えられます。	true	boolean
readLockLoggingLevel (lock)	読み取りロックを取得できなかったときに使用されるロギングレベル。デフォルトでは、WARN がログに記録されます。このレベルは OFF でロギングを持たずに変更できます。このオプションは、type の readLock（changed、fileLock、idempotent、idempotent-changed、idempotent-rename、rename）にのみ適用できます。	DEBUG	LoggingLevel
readLockMarkerFile (lock)	変更、名前変更、または排他的読み取りロックタイプでマーカーファイルを使用するかどうか。デフォルトでは、マーカーファイルが使用されるだけでなく、同じファイルを選択する他のプロセスに対して保護されます。このオプションを false に設定すると、この動作をオフにすることができます。たとえば、Camel アプリケーションによってマーカーファイルをファイルシステムに書き込みます。	true	boolean

Name	説明	デフォルト	Type
readLockMinAge (lock)	このオプションは、readLock=change にのみ適用されます。このオプションでは、読み取りロックの取得を試行する前にファイルに必要な最小期間を指定できます。たとえば、readLockMinAge=300s を使用して、ファイルを最後の 5 分前とする必要があります。これにより、指定された期間以上のファイルの取得を試みるため、変更した読み取りロックが高速化されます。	0	Long
readLockMinLength (lock)	このオプションは、readLock=changed にのみ適用されます。このオプションを使用すると、最小限のファイルの長さを設定できます。デフォルトでは、Camel はファイルにデータが含まれることを想定するため、デフォルト値は 1 です。このオプションをゼロに設定するには、ゼロ長のファイルを使用できます。	1	Long
readLockRemoveOnCommit (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションでは、ファイルの処理に成功し、コミットが行われるときに、べき等リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。デフォルトでは、このファイルは削除されません。これにより、競合状態が発生しないため、別のアクティブなノードがファイルを取得できなくなります。代わりに、べき等リポジトリーは、X 分後にファイル名のエントリーをエビクトするように設定するエビクションストラテジーをサポートする可能性があります。これにより、競合状態に関する問題がなくなります。	false	boolean
readLockRemoveOnRollback (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションでは、ファイルの処理に失敗し、ロールバックが行われるときに、べき等リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。このオプションが false の場合、ファイル名のエントリーが確認されます（ファイルがコミットしたかのように）。	true	boolean

Name	説明	デフォルト	Type
readLockTimeout (lock)	read-lock で対応している場合は、read-lock のオプションのタイムアウト（ミリ秒単位）。read-lock が許可されず、タイムアウトが発生すると、Camel はファイルをスキップします。次回のポーリング Camel はファイルを再度試行し、今回は読み取りロックが付与される可能性があります。0 以下の値を使用して、永久に指定します。現在、fileLock、change、および rename はタイムアウトに対応しています。注記： FTP の場合、デフォルトの readLockTimeout 値は 10000 ではなく 20000 です。readLockTimeout の値は readLockCheckInterval よりも大きくなければなりませんが、thumb のルールは readLockCheckInterval よりも少なくとも 2 倍以上タイムアウトになるようにします。これは、読み取りロックプロセスでタイムアウトがヒットするまでにロックの取得を試行するために バブル時間が許可されるようにするために必要です。	10000	Long
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long

Name	説明	デフォルト	Type
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
shuffle (sort)	ファイルの一覧をシャッフルする方法（ランダムな順序など）	false	boolean
sortBy (sort)	File 言語を使用した組み込みソート。入れ子のソートをサポートするため、ファイル名でソートされ、2 番目にグループの種類を変更した日付でソートできます。		文字列
sorter (sort)	java.util.Comparator クラスとしてのプラグ可能なソーター。		GenericFile<T>>
アカウント （セキュリティー）	ログインに使用するアカウント		文字列
パスワード （セキュリティー）	ログインに使用するパスワード		文字列

Name	説明	デフォルト	Type
ユーザー名（セキュリティー）	ログインに使用するユーザー名		文字列

107.3. FTPS コンポーネントのデフォルトトラストストア

FTPS コンポーネントで SSL に関連する ftpClient. プロパティーを使用する場合、トラストストアはすべての証明書を受け入れます。信頼で選択可能な証明書のみを使用する場合は、ftpClient.trustStore.xxx オプションでトラストストアを設定するか、カスタムの ftpClient を設定する必要があります。

sslContextParameters を使用する場合、トラストストアは提供される SSLContextParameters インスタンスの設定によって管理されます。

ftpClient または ftpClientConfig. プレフィックスを使用すると、直接 URI から ftpClient および ftpClientConfig に追加のオプションを設定できます。

たとえば、FTPClient の setDataTimeout を 30 秒に設定するには、以下を行います。

```
from("ftp://foo@myserver?password=secret&ftpClient.dataTimeout=30000").to("bean:foo");
```

日付形式またはタイムゾーンを設定するなど、両方の接頭辞を混在させ、一致させることもできます。

```
from("ftp://foo@myserver?
password=secret&ftpClient.dataTimeout=30000&ftpClientConfig.serverLanguageCode=fr").to(
"bean:foo");
```

これらのオプションは、いくつでも指定できます。

可能なオプションと詳細については、Apache Commons FTP FTPClientConfig のドキュメントを参照してください。 Apache Commons FTP FTPClient も同様です。

URL に多くの長い設定がない場合、レジストリーでの Camel ルックアップを許可することにより、ftpClient または ftpClientConfig を参照して使用することができます。

以下に例を示します。

```
<bean id="myConfig" class="org.apache.commons.net.ftp.FTPClientConfig">
    <property name="lenientFutureDates" value="true"/>
    <property name="serverLanguageCode" value="fr"/>
</bean>
```

そして、URL で # 表記を使用すると、Camel がこの Bean を検索できるようにします。

```
from("ftp://foo@myserver?password=secret&ftpClientConfig=#myConfig").to("bean:foo");
```

107.4. 例

FTP://someone@someftpserver.com/public/upload/images/holiday2008?password=secret&binary=true

FTP://someoneelse@someotherftpserver.co.uk:12049/reports/2008/password=secret&binary=false
ftp://publicftpserver.com/download

107.5. 並行処理性

FTP コンシューマーは同時実行をサポートしません。

FTP コンシューマー（同じエンドポイントを持つ）は同時実行をサポートしません（バッキング FTP クライアントはスレッドセーフではありません）。
複数の FTP コンシューマーを使用して、異なるエンドポイントからポーリングできます。これは、同時コンシューマーをサポートしない単一のエンドポイントのみです。

FTP プロデューサーにはこの問題が なく、同時実行をサポートします。

107.6. 詳細情報

このコンポーネントは、File コンポーネントの拡張機能です。そのため、File コンポーネントページにはサンプルや詳細が記載されています。

107.7. ファイルを使用する場合のデフォルト

FTP コンシューマーはデフォルトで、リモート FTP サーバーで消費されたファイルを変更しないままにします。ファイルを削除する場合や、別の場所に移動する場合は、明示的に設定する必要があります。たとえば、delete=true を使用してファイルを削除するか、move=.done を使用してファイルを非表示の完了したサブディレクトリーに移動します。

通常の File コンシューマーは、デフォルトでファイルを .camel サブディレクトリーに移動するためとは異なります。Camel は FTP コンシューマーに対してデフォルトでこれを実行しないため、ファイルを移動または削除できるようにパーミッションがデフォルトで含まれて い ない可能性があります。

107.7.1. 制限

オプション readLock を使用すると、現在書き込まれている進行中のファイルを Camel が消費し ない ように強制できます。ただし、このオプションはデフォルトで無効になっています。これにはユーザーが書き込みアクセスを持つ必要があるためです。ロックの読み取りに関する詳細は、File2 のオプションの表を参照してください。
FTP 上で現在書き込まれているファイルが使用されないようにする他のソリューションがあります。たとえば、一時的な宛先に書き込みを行い、書き込み後にファイルを移動できます。

move または preMove オプションを使用してファイルを移動すると、ファイルは FTP_ROOT フォルダーに制限されます。これにより、FTP 領域外でファイルを移動できなくなります。別の領域にファイルを移動する場合は、ソフトリンクを使用して、ファイルをソフトリンクフォルダーに移動できます。

107.8. メッセージヘッダー

以下のメッセージヘッダーを使用して、コンポーネントの動作に影響を与えることができます。

ヘッダー	説明
CamelFileName	エンドポイントに送信する際に出力メッセージに使用される出力ファイル名（エンドポイントディレクトリーと相対的）を指定します。これがなく、式がいずれもない場合は、生成されたメッセージ ID がファイル名として使用されます。
CamelFileNameProduced	書き込まれた出力ファイルのパス（パス + 名）。このヘッダーは Camel によって設定され、その目的は書き込まれたファイルの名前をエンドユーザーに提供します。

ヘッダー	説明
CamelFileIndex	このバッチで消費されるファイルの合計数の現在のインデックス。
CamelFileSize	このバッチで消費されているファイルの合計数。
CamelFileHost	リモートホスト名。
CamelFileLocalWorkPath	ローカルのワークディレクトリーへのパス（ローカルの作業ディレクトリーが使用される場合）。

FTP/FTPS コンシューマーおよびプロデューサーに加えて、以下のヘッダーを使用して Camel メッセージ を強化します。

ヘッダー	説明
CamelFtpReplyCode	Camel 2.11.1: FTP クライアント応答コード（タイプは整数）
CamelFtpReplyString	Camel 2.11.1: FTP クライアント応答文字列

107.9. タイムアウトについて

タイムアウトの設定には、2 つのライブラリーセット（top を参照）には異なる API があります。connectTimeout オプションは両方に使用して、ネットワーク接続を確立するためにタイムアウトをミリ秒単位で設定できます。個別の soTimeout は FTP/FTPS でも設定できます。これは ftpClient.soTimeout の使用に対応します。SFTP は、自動的に connectTimeout を soTimeout として使用します。timeout オプションは、ftpClient.dataTimeout 値に対応するデータタイムアウトとして FTP/FTSP にのみ適用されます。すべてのタイムアウト値はミリ秒単位です。

107.10. ローカルワークディレクトリーの使用

Camel は、リモート FTP サーバーからの使用をサポートし、ファイルをローカルの作業ディレクトリーに直接ダウンロードします。これにより、FileOutputStream を使用してローカルファイルに直接ストリーミングされるため、リモートファイルの内容全体をメモリーに読み込めないようにします。

Camel はファイルのダウンロード中に .inprogress がエクステンションと同じ名前を持つローカルファイルに保存されますが、.inprogress はエクステンションとして機能します。その後、ファイルの名前が変更され、.inprogress 接尾辞が削除されます。最後に、Exchange が完了するとローカルファイルが削除されます。

そのため、リモートの FTP サーバーからファイルをダウンロードし、これをファイルとして保存する場合は、以下のようなファイルエンドポイントにルーティングする必要があります。

```
from("ftp://someone@someserver.com?
password=secret&localWorkDirectory=/tmp").to("file://inbox");
```

ヒント

上記のルートは、ファイルコンテンツ全体をメモリーに読み込まないようにするため、非常に効率的です。リモートファイルをローカルファイルストリームに直接ダウンロードします。java.io.File ハンドルがエクスチェンジボディーとして使用されます。ファイルプロデューサーはこのファクトを活用し、ワークファイル java.io.File で直接作業し、ターゲットファイル名に対して java.io.File.rename を実行できます。Camel はローカルワークファイルを認識しているので、ワークファイルは削除することが意図されているため、ファイルコピーの代わりに名前を最適化し、名前を使用することができます。

107.11. ディレクトリーをステップ的に変更

ファイルを使用する場合（ダウンロードなど）またはファイルの生成（アップロードなど）の 2 つのモードで、Camel FTP がディレクトリーをトラバースできます。

- stepwise
- stepwise

状態およびセキュリティーの問題に応じて、1 つを選択できます。一部の Camel エンドユーザーは、ステップ的にのみファイルをダウンロードできるだけでなく、ダウンロードできない場合にのみダウンロードできます。少なくとも選択（Camel 2.6 以降）を選択できます。

Camel 2.0 - 2.5 では、モードが 1 つのみあり、以下のモードがあります。

- Before 2.5 not stepwise
- 2.5 stepwise

Camel 2.6 以降では、任意で動作 を 制御できるようになりました。

ディレクトリーの変更は、ほとんどの場合、ユーザーがホームディレクトリーに制限され、ホームディレクトリーが「/」と報告される場合にのみ機能します。

これら 2 つの違いは、例を使用して見ておくのが最適です。リモート FTP サーバーに次のディレクトリー構造がある場合は、ファイルをトラバースしてダウンロードする必要があります。

```
/
/one
/one/two
/one/two/sub-a
/one/two/sub-b
```

そして、各サブア(a.txt)および sub-b(b.txt)フォルダーのファイルがあります。

107.11.1. stepwise=true の使用（デフォルトモード）

```
TYPE A
200 Type set to A
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
CWD two
250 CWD successful. "/one/two" is current directory.
SYST
215 UNIX emulated by FileZilla
PORT 127,0,0,1,17,94
200 Port command successful
LIST
150 Opening data channel for directory list.
226 Transfer OK
CWD sub-a
250 CWD successful. "/one/two/sub-a" is current directory.
PORT 127,0,0,1,17,95
200 Port command successful
LIST
150 Opening data channel for directory list.
226 Transfer OK
CDUP
200 CDUP successful. "/one/two" is current directory.
CWD sub-b
250 CWD successful. "/one/two/sub-b" is current directory.
PORT 127,0,0,1,17,96
200 Port command successful
LIST
150 Opening data channel for directory list.
226 Transfer OK
CDUP
200 CDUP successful. "/one/two" is current directory.
```

```
CWD /
250 CWD successful. "/" is current directory.
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
CWD two
250 CWD successful. "/one/two" is current directory.
PORT 127,0,0,1,17,97
200 Port command successful
RETR foo.txt
150 Opening data channel for file transfer.
226 Transfer OK
CWD /
250 CWD successful. "/" is current directory.
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
CWD two
250 CWD successful. "/one/two" is current directory.
CWD sub-a
250 CWD successful. "/one/two/sub-a" is current directory.
PORT 127,0,0,1,17,98
200 Port command successful
RETR a.txt
150 Opening data channel for file transfer.
226 Transfer OK
CWD /
250 CWD successful. "/" is current directory.
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
CWD two
250 CWD successful. "/one/two" is current directory.
CWD sub-b
250 CWD successful. "/one/two/sub-b" is current directory.
PORT 127,0,0,1,17,99
200 Port command successful
RETR b.txt
150 Opening data channel for file transfer.
226 Transfer OK
CWD /
250 CWD successful. "/" is current directory.
QUIT
221 Goodbye
disconnected.
```

ステップの強制が有効になっていると分かるように、CD xxx を使用してディレクトリー構造をトラバースします。

107.11.2. stepwise=false の使用

```
230 Logged on
TYPE A
200 Type set to A
SYST
215 UNIX emulated by FileZilla
PORT 127,0,0,1,4,122
200 Port command successful
LIST one/two
150 Opening data channel for directory list
226 Transfer OK
PORT 127,0,0,1,4,123
200 Port command successful
LIST one/two/sub-a
150 Opening data channel for directory list
226 Transfer OK
PORT 127,0,0,1,4,124
200 Port command successful
LIST one/two/sub-b
150 Opening data channel for directory list
226 Transfer OK
PORT 127,0,0,1,4,125
200 Port command successful
RETR one/two/foo.txt
150 Opening data channel for file transfer.
226 Transfer OK
PORT 127,0,0,1,4,126
200 Port command successful
RETR one/two/sub-a/a.txt
150 Opening data channel for file transfer.
226 Transfer OK
PORT 127,0,0,1,4,127
200 Port command successful
RETR one/two/sub-b/b.txt
150 Opening data channel for file transfer.
226 Transfer OK
QUIT
221 Goodbye
disconnected.
```

ステップ単位で表示されないように、CD 操作は全く起動されません。

107.12. サンプル

以下の例では、毎時（60 分）を BINARY コンテンツとして 1 回に FTP サーバーからすべてのレポートをダウンロードし、ローカルファイルシステムにファイルとして保存するように Camel を設定します。

Spring DSL を使用したルートの場合：

```
<route>
  <from uri="ftp://scott@localhost/public/reports?
password=tiger&amp;binary=true&amp;delay=60000"/>
  <to uri="file://target/test-reports"/>
</route>
```

107.12.1. リモート FTPS サーバー（暗黙的な SSL）およびクライアント認証の使用

```
from("ftps://admin@localhost:2222/public/camel?
password=admin&securityProtocol=SSL&isImplicit=true
    &ftpClient.keyStore.file=./src/test/resources/server.jks
    &ftpClient.keyStore.password=password&ftpClient.keyStore.keyPassword=password")
  .to("bean:foo");
```

107.12.2. リモート FTPS サーバー（明示的な TLS）とカスタムトラストストア設定の使用

```
from("ftps://admin@localhost:2222/public/camel?
password=admin&ftpClient.trustStore.file=./src/test/resources/server.jks&ftpClient.trustStore.
password=password")
  .to("bean:foo");
```

107.13. FILTER USING ORG.APACHE.CAMEL.COMPONENT.FILE.GENERICFILEFILTER

Camel はプラグ可能なフィルタリングストラテジーをサポートします。このストラテジーでは、Java の org.apache.camel.component.file.GenericFileFilter でビルドを使用します。その後、このようなフィルターでエンドポイントを設定し、処理される前に特定のフィルターをスキップできます。

この例では、ファイル名のレポートで始まるファイルのみを許可する独自のフィルターを構築します。

そして、filter 属性を使用して Spring XML ファイルで定義したフィルターを参照する（# 表記を使用）、ルートを設定することができます。

```
<!-- define our sorter as a plain spring bean -->
<bean id="myFilter" class="com.mycompany.MyFileFilter"/>

<route>
  <from uri="ftp://someuser@someftpserver.com?password=secret&amp;filter=#myFilter"/>
  <to uri="bean:processInbox"/>
</route>
```

107.14. ANT パス MATCHER を使用したフィルタリング

ANT path matcher は、camel-spring jar の追加設定なしで提供されるフィルターです。そのため、

Maven を使用している場合は camel-spring に依存する必要があります。
これは、Spring の AntPathMatcher を使用して実際の一致を実行する理由です。

ファイルパスは、以下のルールと一致します。

- *? 1 文字に一致します。*
- ** 0 以上の文字に一致します。*
- *** パスの 0 個以上のディレクトリーと一致します。*

以下の例は、その使用方法を示しています。

107.15. SFTP でのプロキシーの使用

HTTP プロキシーを使用してリモートホストに接続するには、以下のようにルートを設定します。

```
<!-- define our sorter as a plain spring bean -->
<bean id="proxy" class="com.jcraft.jsch.ProxyHTTP">
  <constructor-arg value="localhost"/>
  <constructor-arg value="7777"/>
</bean>

<route>
  <from uri="sftp://localhost:9999/root?username=admin&password=admin&proxy=#proxy"/>
  <to uri="bean:processFile"/>
</route>
```

必要に応じて、ユーザー名およびパスワードをプロキシーに割り当てることもできます。com.jCreation.jsch.Proxy のドキュメントを参照してください。 すべてのオプションを検出します。

107.16. 優先 SFTP 認証方法の設定

sftp コンポーネントが使用する認証方法の一覧を明示的に指定する場合は、preferredAuthentications オプションを使用します。たとえば、公開鍵が利用できない場合に Camel がプライベート/パブリック SSH キーで認証を試み、ユーザー/パスワード認証にフォールバックする場合は、以下のルート設定を使用します。

```
from("sftp://localhost:9999/root?
username=admin&password=admin&preferredAuthentications=publickey,password").
  to("bean:processFile");
```

107.17. 固定名を使用した単一ファイルの使用

単一ファイルをダウンロードし、ファイル名を認識したい場合は、fileName=myFileName.txt を使用して Camel にダウンロードするファイルの名前を指示できます。デフォルトでは、コンシューマーは FTP LIST コマンドを実行し、ディレクトリーの一覧を実行して fileName オプションに基づいてこれらのファイルをフィルタリングします。このユースケースでは、useList=false を設定してディレクトリーの一覧をオフにすることが推奨されます。たとえば、FTP サーバーへのログインに使用されるユーザーアカウントには、FTP LIST コマンドを実行するパーミッションがない可能性があります。そのため、useList=false を使用してこれをオフにしてから、fileName=myFileName.txt でダウンロードするファイルの固定名を指定してから、FTP コンシューマーはファイルをダウンロードできます。何らかの理由でファイルが存在しない場合、Camel はデフォルトで例外をスローします。これをオフにして ignoreFileNotFoundOrPermissionError=true を設定して無視できます。

たとえば、1 つのファイルを取得し、使用後に削除する Camel ルートがあるとします。

```
from("ftp://admin@localhost:21/nolist/?
password=admin&stepwise=false&useList=false&ignoreFileNotFoundOrPermissionError=true
&fileName=report.txt&delete=true")
  .to("activemq:queue:report");
```

上記で説明したすべてのオプションを使用していることに注意してください。

これは ConsumerTemplate とともに使用することもできます。たとえば、単一ファイル（存在する場合）をダウンロードし、ファイルの内容を String タイプとして取得するには、以下を実行します。

```
String data = template.retrieveBodyNoWait("ftp://admin@localhost:21/nolist/?
password=admin&stepwise=false&useList=false&ignoreFileNotFoundOrPermissionError=true
&fileName=report.txt&delete=true", String.class);
```

107.18. デバッグロギング

このコンポーネントにはログレベルの TRACE があり、問題がある場合に役立ちます。

107.19. 関連項目

- Configuring Camel (Camel の設定)

- *コンポーネント*
- *エンドポイント*
- *はじめに*
- *File2*

第108章 FTPS コンポーネント

Camel バージョン 2.2 で利用可能

このコンポーネントは、FTP プロトコルおよび SFTP プロトコルを介してリモートファイルシステムへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ftp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

詳細は、FTP コンポーネントを参照してください。

108.1. URI オプション

以下のオプションは、FTPS コンポーネント専用です。

FTPS コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

FTPS エンドポイントは、URI 構文を使用して設定します。

```
ftps:host:port/directoryName
```

*以下の **path** パラメーターおよびクエリーパラメーターを使用します。*

108.1.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
host	FTP サーバーに **必要な** ホスト名		文字列
port	FTP サーバーのポート		int
directoryName	起動ディレクトリー		文字列

108.1.2. クエリーパラメーター（116 パラメーター）：

Name	説明	デフォルト	Type
バイナリー （共通）	ファイル転送モード（BINARY または ASCII）を指定します。デフォルトは ASCII(false)です。	false	boolean
charset (common)	このオプションは、ファイルのエンコーディングを指定するために使用されます。これはコンシューマーで使用して、ファイルのエンコーディングを指定できます。これにより、Camel はファイルの内容にアクセスした場合にファイルの内容を読み込む必要があります。ファイルを書き込む場合も同様に、このオプションを使用して、ファイルを書き込む文字を指定することもできます。ファイルを書くときには、メッセージの内容をメモリーに読み込んで、データを設定済みの文字セットに変換しなければならない場合があります。そのため、大きなメッセージがある場合は使用しないでください。		文字列
切断 （共通）	使用後にリモートの FTP サーバーから適切な接続を解除するかどうか。切断すると、現在の FTP サーバーへの接続が切断されるだけです。停止するコンシューマーがある場合は、代わりにコンシューマー/ルートを停止する必要があります。	false	boolean

Name	説明	デフォルト	Type
doneFileName (common)	プロデューサー： 指定された場合、元のファイルが書き込まれると、Camel は 2 番目に完了したファイルを書き込みます。完了ファイルは空になります。このオプションは、使用するファイル名を設定します。固定名を指定できます。または、動的プレースホルダーを使用することもできます。done ファイルは、常に元のファイルと同じフォルダーに書き込まれます。コンシューマー： 提供された場合、Camel は完了したファイルが存在する場合にのみファイルを消費します。このオプションは、使用するファイル名を設定します。固定名を指定できます。または、動的プレースホルダーを使用することもできます。完成したファイルは、常に元のファイルと同じフォルダーに期待されます。$file.name および $file.name.noext のみが動的プレースホルダーとしてサポートされます。		文字列
fileName (common)	File Language などの式を使用して、ファイル名を動的に設定します。コンシューマーの場合は、ファイル名フィルターとして使用されます。プロデューサーの場合、書き込みするファイル名を評価するために使用されます。式が設定されている場合は、CamelFileName ヘッダーよりも優先されます。（注記： ヘッダー自体は式にすることもできます。式オプションは String タイプと Expression 型の両方をサポートします。式が String タイプである場合、これは常に File 言語を使用して評価されます。式が Expression タイプである場合、指定された式タイプが使用されます。たとえば、OGNL 式を使用できます。コンシューマーの場合は、これを使用してファイル名を絞り込むことができます。そのため、インスタンスは File Language 構文 mydata-$date:yyyyMMdd.txt を使用して現在のファイルを使用できます。プロデューサーは、既存の CamelFileName ヘッダーよりも優先される CamelOverruleFileName ヘッダーをサポートします。CamelOverruleFileName は 1 度だけ使用されるヘッダーで、CamelFileName を一時的に保存し、後で復元する必要があるため、より簡単に使用できます。		文字列
passiveMode (common)	パッシブモード接続を設定します。デフォルトはアクティブモード接続です。	false	boolean
セパレーター （共通）	使用するパス区切り文字を設定します。UNIX = Uses unix スタイルの path separator Windows = Uses windows style path separator Auto =(is default)Use existing path separator in file name	UNIX	PathSeparator

Name	説明	デフォルト	Type
transferLoggingInterval Seconds (common)	インフライトでアップロードおよびダウンロード操作の進捗をログに記録するときに使用する間隔を秒単位で設定します。これは、操作に時間がかかるときにロギングの進捗に使用されます。	5	int
transferLoggingLevel (common)	アップロードおよびダウンロード操作の進捗をログに記録するときに使用するロギングレベルを設定します。	DEBUG	LoggingLevel
transferLoggingVerbose (common)	アップロード操作およびダウンロード操作の進捗の進捗状況を（詳細にわたり）実行するかどうかを設定します。	false	boolean
fastExistsCheck (common)	このオプションを true に設定すると、camel-ftp はリストファイルを直接使用してファイルが存在するかどうかを確認します。一部の FTP サーバーはファイルを直接一覧表示できない可能性があるため、オプションが false の場合、camel-ftp は古い方法でディレクトリーを一覧表示し、ファイルが存在するかどうかを確認します。また、このオプションは readLock=changed にも影響し、ファイル情報を更新する高速チェックを実行するかどうかを制御します。FTP サーバーに多くのファイルがある場合に、このプロセスを迅速化するために使用できます。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
Delete （コンシューマー）	true の場合、ファイルは正常に処理された後に削除されます。	false	boolean
moveFailed (consumer)	Simple 言語に基づいて移動失敗式を設定します。たとえば、.error サブディレクトリーにファイルを移動するには、.error を使用します。注記： 失敗した場所にファイルを移動すると Camel はエラーを処理し、再度ファイルを取得しません。		文字列
noop （コンシューマー）	true の場合、ファイルは移動または削除されません。このオプションは、読み取り専用データまたは ETL タイプの要件に適しています。noop=true の場合、Camel はべき等=true も設定し、同じファイルを何度も消費しないようにします。	false	boolean

Name	説明	デフォルト	Type
preMove (consumer)	処理前にファイル名を動的に設定するために使用される式（ファイル言語など）。たとえば、進行中のファイルを order ディレクトリーに移動するには、この値を order に設定します。		文字列
preSort (consumer)	pre-sort を有効にすると、コンシューマーはポーリング中にファイルとディレクトリー名をソートし、ファイルシステムから取得されたものになります。これは、ソートされた順序でファイルで操作する必要がある場合があります。pre-sort は、コンシューマーがフィルターをかけ、Camel が処理するファイルを受け入れる前に実行されます。このオプションは、無効を意味する default=false です。	false	boolean
再帰的 （コンシューマー）	ディレクトリーの場合は、すべてのサブディレクトリー内のファイルも検索します。	false	boolean
resumeDownload (consumer)	ダウンロードを有効にするかどうかを設定します。これは FTP サーバー（ほぼすべての FTP サーバーがそれをサポートしている）でサポートされている必要があります。さらに、ダウンロードされたファイルがローカルディレクトリーに保存されるように localWorkDirectory オプションを設定し、ダウンロードの再開をサポートするのに必要なオプションのバイナリーを有効にする必要があります。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
streamDownload (consumer)	ローカルの作業ディレクトリーを使用しない場合に使用するダウンロード方法を設定します。true に設定すると、リモートファイルは読み取り時にルートにストリーミングされます。false に設定すると、リモートファイルはルートに送信される前にメモリーに読み込まれます。	false	boolean
directoryMustExist (consumer)	startingDirectoryMustExist と同様ですが、再帰的なサブディレクトリーのポーリング時に適用されます。	false	boolean
download (consumer)	FTP コンシューマーがファイルをダウンロードするかどうか。このオプションを false に設定すると、メッセージボディーは null になりますが、コンシューマーはファイル名、ファイルサイズなどのファイルの詳細が含まれる Camel Exchange のトリガーになります。ファイルはダウンロードされないだけです。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
handleDirectoryParserAbsoluteResult (consumer)	ディレクトリーパーサーが絶対パスで結果となる場合は、コンシューマーがパス内のサブフォルダーとファイルを処理する方法を設定できます。そのため、一部の FTP サーバーは絶対パスでファイル名を返す可能性があるため、FTP コンポーネントが、返されるパスを相対パスに変換してこれを処理する必要があることです。	false	boolean
ignoreFileNotFoundOrPermissionError (consumer)	いつ無視するか（ディレクトリー内のファイルを一覧表示しようとするか、またはファイルをダウンロードする際に）無視するかどうか。これは存在しないか、パーミッションエラーが原因です。デフォルトでは、ディレクトリーまたはファイルが存在しないか、パーミッションが不十分な場合に、例外が発生します。このオプションを true に設定すると、代わりにこれを無視することができます。	false	boolean
inProgressRepository (consumer)	プラグ可能な in-progress リポジトリー org.apache.camel.spi.IdempotentRepository。進行中のリポジトリーは、現在使用中の進行中のファイルを考慮するために使用されます。デフォルトでは、メモリーベースのリポジトリーが使用されます。		String>
localWorkDirectory (consumer)	使用する場合は、ローカルの作業ディレクトリーを使用してリモートファイルの内容を直接ローカルファイルに保存し、コンテンツをメモリーにロードしないようにできます。これは、非常に大きなリモートファイルを使用しているため、メモリーを節約できることに有益です。		文字列
onCompletionException Handler (consumer)	カスタムの org.apache.camel.spi.ExceptionHandler を使用して、コンシューマーがコミットまたはロールバックを行う完了プロセス時に発生する例外を処理します。デフォルトの実装は、WARN レベルですべての例外をログに記録し、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processStrategy (consumer)	プラグ可能な org.apache.camel.component.file.GenericFileProcessStrategy を使用すると、独自の readLock オプションまたは同様のものを実装できます。また、特別な対応ファイルが存在するなど、ファイルを使用する前に特別な条件が満たされなければならない場合にも使用できます。このオプションを設定すると、readLock オプションが適用されません。		GenericFileProcess Strategy<T>
receiveBufferSize (consumer)	FTPClient でのみ使用される受信（ダウンロード）バッファーサイズ	32768	int
startingDirectoryMustExist (consumer)	開始ディレクトリーが存在するかどうか。autoCreate オプションはデフォルトで有効になっています。つまり、開始ディレクトリーが存在しない場合には、通常、起動ディレクトリーが作成されます。autoCreate を無効にして、これを有効にして、開始ディレクトリーが存在することを確認できます。ディレクトリーが存在しない場合は例外が発生します。	false	boolean
useList (consumer)	ファイルのダウンロード時に LIST コマンドの使用を許可するかどうか。デフォルトは true です。ユースケースによっては、特定のファイルをダウンロードし、LIST コマンドを使用できないことがあるため、このオプションを false に設定します。このオプションを使用すると、ダウンロードする特定のファイルには、ファイルサイズ、タイムスタンプ、パーミッションなどのメタデータ情報が含まれません。これらの情報は、LIST コマンドが使用されている場合のみ取得できるためです。	true	boolean

Name	説明	デフォルト	Type
fileExist (producer)	同じ名前のファイルがすでに存在する場合のアクション。デフォルトのオーバーライドは、既存のファイルを置き換えます。append: 既存のファイルにコンテンツを追加します。fail: GenericFileOperationException をスローし、既存のファイルがあることを示します。ignore: 問題を警告なしで無視し、既存のファイルは上書きしませんが、すべてが問題であることを前提としています。move - オプションでは、moveExisting オプションも設定する必要があります。eagerDeleteTargetFile オプションを使用して、ファイルの移動や既存のファイルがすでに存在する場合に実行内容を制御できます。それ以外の場合は、移動操作が失敗します。Move オプションは、ターゲットファイルを書き込む前に既存のファイルを移動します。TryRename は、tempFileName オプションが使用されている場合にのみ適用されます。これにより、存在チェックを実行せずに、一時的な名前から実際の名前へのファイルの名前変更を試すことができます。一部のファイルシステムや、特に FTP サーバーでは、このチェックの方が高速である場合があります。	オーバーライド	GenericFileExist
flatten (producer)	フラット化は、ファイル名パスをフラット化して先頭のパスを削除するために使用されるので、ファイル名だけになります。これにより、サブディレクトリーに再帰的に消費できますが、たとえば別のディレクトリーにファイルを書き込むと、それらのファイルは単一のディレクトリーに書き込まれます。これをプロデューサーで true に設定すると、CamelFileName ヘッダーのファイル名が任意の先頭パスに対して削除されます。	false	boolean
moveExisting (producer)	fileExist=Move が設定されている場合に使用するファイル名の計算に使用される式（ファイル言語など）。ファイルをバックアップサブディレクトリーに移動するには、バックアップを入力します。このオプションは、file:name、file:name.ext、file:name.noext、file:onlyname、file:onlyname.noext、file:ext、および file:parent の File Language トークンのみをサポートします。FTP コンポーネントは既存のファイルをベースとして、相対ディレクトリーにのみ移動できるため、file:parent は FTP コンポーネントではサポートされないことに注意してください。		文字列
tempFileName (producer)	tempPrefix オプションと同じですが、File 言語を使用する一時的なファイル名の命名をより細かく制御できます。		文字列

Name	説明	デフォルト	Type
tempPrefix (producer)	このオプションは、一時的な名前を使用してファイルを書き込むために使用されます。次に、書き込みが完了したら、その名前を変更します。書き込まれているファイルを特定し、進行中のファイルで読み取るコンシューマー（排他的読み取りロックを使用しない）も回避できます。大容量のファイルをアップロードする場合に FTP が使用することがよくあります。		文字列
allowNullBody (producer)	ファイルの書き込み中に null ボディーを許可するかどうかを指定するのに使用します。true に設定すると、空のファイルが作成され、false に設定され、null ボディーを file コンポーネントに送信しようとすると、'Cannot write null body to file.' の GenericFileWriteException がスローされます。fileExist オプションを 'Override' に設定すると、ファイルが切り捨てられます。追加する場合は、ファイルは変更されません。	false	boolean
chmod （プロデューサー）	保存したファイルで chmod を設定できます。例：chmod=640		文字列
disconnectOnBatchComplete (producer)	Batch アップロードの完了後にリモート FTP サーバーから右に切断するかどうか。disconnectOnBatchComplete は、現在の FTP サーバーの接続のみを切断します。	false	boolean
eagerDeleteTargetFile (producer)	既存のターゲットファイルを活発に削除するかどうか。このオプションは、fileExists=Override オプションおよび tempFileName オプションを使用している場合にのみ適用されます。このパラメーターを使用して、一時ファイルが書き込まれる前にターゲットファイルを削除する（false に設定します）ことができます。たとえば、大きなファイルを作成し、一時ファイルが書き込まれている間にターゲットファイルを存在させたいとします。これにより、一時ファイルの名前がターゲットファイル名に変更される直前に、ターゲットファイルが最後の時点までのみ削除されます。また、このオプションは、fileExist=Move が有効で、既存のファイルが存在すると、既存のファイルを削除するかどうかを制御することもできます。このオプションの copyAndDeleteOnRenameFails false を指定した場合、既存のファイルが存在する場合は、移動操作の前に既存のファイルが削除されます。	true	boolean

Name	説明	デフォルト	Type
keepLastModified (producer)	ソースファイルから最後に変更したタイムスタンプを保持します（存在する場合）。Exchange.FILE_LAST_MODIFIED ヘッダーを使用してタイムスタンプを見つけます。このヘッダーには java.util.Date またはタイムスタンプの long を含めることができます。タイムスタンプが存在し、オプションが有効な場合は、書き込まれたファイルにこのタイムスタンプを設定します。注記： このオプションは、ファイルプロデューサーにのみ適用されます。このオプションは、ftp プロデューサーでは使用できません。	false	boolean
sendNoop (producer)	FTP サーバーにファイルをアップロードする前に、事前書き込みチェックとして noop コマンドを送信するかどうか。これは、接続の検証がまだ有効であるため、デフォルトでは、ファイルのアップロードを警告なしで再接続できます。ただし、これにより問題が発生する場合は、このオプションをオフにすることができます。	true	boolean
activePortRange (advanced)	アクティブモードでクライアント側のポート範囲を設定します。構文： minPort-maxPort 両方のポート番号は含まれます。たとえば 10000-19999（すべての 1xxxx ポートを含める）です。		文字列
autocreate(advanced)	ファイルのパス名に不足しているディレクトリーを自動的に作成します。ファイルコンシューマーの場合は、開始ディレクトリーを作成することを意味します。ファイルプロデューサーの場合、ファイルが書き込まれるディレクトリーを意味します。	true	boolean
bufferSize (advanced)	バイト単位で書き込みバッファーサイズを書き込みます。	131072	int
connectTimeout (advanced)	FTPClient と JSCH の両方によって接続が確立されるまでの接続タイムアウトを設定します。	10000	int
ftpClient (advanced)	FTPClient のカスタムインスタンスを使用するには、以下を行います。		FTPClient
ftpClientConfig (advanced)	FTPClientConfig のカスタムインスタンスを使用して FTP クライアントを設定するには、エンドポイントを使用する必要があります。		FTPClientConfig
ftpClientConfigParameters (advanced)	FtpComponent が使用し、FTPClientConfig の追加パラメーターを提供します。		マップ

Name	説明	デフォルト	Type
ftpClientParameters (advanced)	FtpComponent が使用し、FTPClient の追加パラメーターを提供します。		マップ
maximumReconnectAttempts (advanced)	リモート FTP サーバーに接続しようとすると Camel が実行を試行する最大再接続試行を指定します。この動作を無効にするには 0 を使用します。		int
reconnectDelay (advanced)	Camel は再接続を試みる前に待機します。		Long
siteCommand (advanced)	ログインの成功後に実行する任意の site コマンドを設定します。複数のサイトコマンドは、改行文字を使用して分離できます。		文字列
soTimeout (advanced)	FTPClient でのみ使用されるようにタイムアウトを設定します。	300000	int
stepwise (advanced)	ファイルのダウンロード時、またはファイルのダウンロード時、またはディレクトリーにファイルをアップロードする場合に、ディレクトリーをステップ的に変更するかどうかを設定します。たとえば、セキュリティー上の理由から FTP サーバーのディレクトリーを変更できない場合は、これを無効にすることができます。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
throwExceptionOnConnect Failed (advanced)	接続に失敗した場合（すべて）に例外がスローされるかどうか。デフォルトでは例外はスローされず、WARN がログに記録されます。これを使用して、例外がスローされ、org.apache.camel.spi.PollingConsumerPollStrategy ロールバックメソッドから発生した例外を処理できます。	false	boolean
タイムアウト （詳細）	FTPClient のみで使用された応答を待機するデータタイムアウトを設定します。	30000	int
antExclude (filter)	ant スタイルのフィルターの除外。antInclude と antExclude の両方を使用する場合は、antInclude よりも antExclude が優先されます。複数の除外はコンマ区切りの形式で指定できます。		文字列
antFilterCaseSensitive (filter)	付与フィルターに大文字と小文字を区別するフラグを設定します。	true	boolean

Name	説明	デフォルト	Type
antInclude (filter)	ant スタイルのフィルターが含まれます。複数の包含は、コンマ区切りの形式で指定できます。		文字列
eagerMaxMessagesPerPoll (filter)	maxMessagesPerPoll の制限が Eager かどうかを制御できます。Eager が Eager の場合は、制限がファイルのスキャン中になります。false の場合、すべてのファイルをスキャンし、並び替えを実行します。このオプションを false に設定すると、すべてのファイルを最初にソートし、ポーリングを制限できます。ソートを実行するには、すべてのファイルの詳細がメモリー内にあるため、メモリー使用量を高くする必要があることに注意してください。	true	boolean
exclude （フィルター）	ファイルを除外するために使用されます（ファイル名が正規表現パターンと一致する場合）。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW（）構文を使用してこれを設定する必要があります。エンドポイント URI の設定に関する詳細は、「エンドポイント URI の設定」を参照してください。		文字列
Filter （フィルター）	プラグ可能なフィルター（org.apache.camel.component.file.GenericFileFilter クラス）フィルターが accept（）メソッドで false を返すと、ファイルをスキップします。		GenericFileFilter<T>
filterDirectory (filter)	Simple 言語に基づいてディレクトリーをフィルタリングします。たとえば、現在の日付でフィルタリングするには、$date:now:yyMMdd などの単純な日付パターンを使用できます。		文字列
filterFile (filter)	Simple 言語に基づいてファイルをフィルタリングします。たとえば、ファイルサイズで絞り込むには、$file:size 5000 を使用できます。		文字列
idempotent (filter)	Idempotent Consumer EIP パターンを使用して、Camel がすでに処理されたファイルをスキップするオプション。デフォルトでは、1000 エントリーを保持するメモリーベースの LRUCache を使用します。noop=true の場合は、同じファイルを何度も使用することを回避するため、べき等性も有効になります。	false	ブール値
idempotentKey (filter)	カスタムのべき等キーを使用するには、以下を行います。デフォルトでは、ファイルの絶対パスが使用されます。File 言語を使用すると、ファイル名とファイルサイズを使用できます (idempotentKey=$file:name-$file:size)。		文字列

Name	説明	デフォルト	Type
idempotentRepository (filter)	何も指定されておらず、べき等性が true の場合、デフォルトでは MemoryMessageIdRepository を使用するプラグ可能なリポジトリー org.apache.camel.spi.IdempotentRepository。		String>
include （フィルター）	ファイルを含めるために使用されます。ファイル名が正規表現パターンと一致する場合（一致する場合は大文字と小文字が区別されます）。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW（）構文を使用してこれを設定する必要があります。エンドポイント URI の設定に関する詳細は、「エンドポイント URI の設定」を参照してください。		文字列
maxDepth (filter)	ディレクトリーを再帰的に処理する際に通過する最大深度。	2147483647	int
maxMessagesPerPoll (filter)	ポーリングごとに収集する最大メッセージを定義します。デフォルトでは最大値は設定されません。1000 などの制限を設定して、数千のファイルがあるサーバーを起動すると回避できます。無効にするには、0 または negative の値を設定します。注記： このオプションが使用されている場合、File および FTP コンポーネントはソート前に制限されます。たとえば、100000 ファイルがあり、maxMessagesPerPoll=500 を使用する場合は、最初の 500 ファイルのみが選択され、その後にソートされます。eagerMaxMessagesPerPoll オプションを使用して、これを false に設定すると、最初にすべてのファイルをスキャンし、後でソートできます。		int
minDepth (filter)	ディレクトリーを再帰的に処理する際に処理を開始する最小深度。minDepth=1 を使用すると、ベースディレクトリーを意味します。minDepth=2 を使用すると、最初のサブディレクトリーを意味します。		int
Move （フィルター）	処理後にファイル名を動的に設定するために使用される式（Simple Language など）。ファイルを .done サブディレクトリーに移動するには、.done と入力します。		文字列
exclusiveReadLockStrategy (lock)	プラグ可能な read-lock を org.apache.camel.component.file.GenericFileExclusiveReadLockStrategy 実装とする。		GenericFileExclusive ReadLockStrategy <T>
readLock (lock)	ファイルに read-lock が排他的な場合にのみファイルをポーリングするために、コンシューマーによって	none	文字列

Name	説明	デフォルト	Type
	使用されます（ファイルが進行中の場合や書き込みされていません）。Camel はファイルロックが許可されるまで待機します。このオプションは、ストラテジーでビルドを提供します。none - No read lock is in use markerFile - Camel はマーカーファイル (fileName.camelLock)を作成し、そのロックを保持します。このオプションは、FTP コンポーネントの変更には使用できません。Changed は、ファイルの長さ/変更タイムスタンプを使用して、ファイルがすでにコピーされているかどうかを検出します。少なくとも 1 つのセクションを使用してこれを判断するため、このオプションはファイルを他のプロセスとしてすぐに消費できませんが、JDK IO API はファイルを別のプロセスで現在使用しているかどうかを判断することができないため、信頼性が高まります。readLockCheckInterval オプションを使用してチェック頻度を設定できます。fileLock - は java.nio.channels.FileLock 用です。このオプションは FTP コンポーネントでは使用できません。この方法は、ファイルシステムが分散ファイルロックに対応している場合を除き、マウント/共有経由でリモートファイルシステムにアクセスするときに回避する必要があります。名前変更は、読み取り専用で読み取りロックを取得できる場合に、テストとしてファイルの名前を変更するためのものです。べき等性 -（ファイルコンポーネントのみ）べき等性は、idempotentRepository を read-lock として使用することです。これにより、べき等リポジトリーの実装がサポートする場合、クラスタリングをサポートする読み取りロックを使用できます。idempotent-changed -（ファイルコンポーネントのみ）idempotent-changed はidempotentRepository を使用し、結合された read-lock として変更されます。これにより、べき等リポジトリー実装がこれをサポートする場合、クラスタリングをサポートする読み取りロックを使用できます。idempotent-rename -（ファイルコンポーネントのみ）idempotent-rename はidempotentRepository を使用し、結合された read-lock として名前を変更します。これにより、べき等リポジトリーの実装がサポートしている場合、クラスタリングをサポートする読み取りロックを使用できます。注記： 各種の読み取りロックは、クラスターモードですべて機能する訳ではありません。この場合、異なるノードの同時コンシューマーは共有ファイルシステムの同じファイルに対して競合しています。atomic 操作を使用して空のマーカーファイルを作成しますが、クラスター内での動作が保証されません。fileLock の方が良好に機能しますが、ファイルシステムは分散ファイルロックなどに対応する必要があります。べき等リポジトリーが Hazelcast コンポーネントや Infinispan などのクラスタリングに対応している場合、べき等リポジトリーがクラスタリングをサポートする場合は、べき等読み取りロックを使用できます。		

Name	説明	デフォルト	Type
readLockCheckInterval (lock)	読み取りロックでサポートされている場合は、read-lock の間隔（ミリ秒単位）。この間隔は、読み取りロックの取得を試みる間にスリープするために使用されます。たとえば、変更した読み取りロックを使用する場合は、低速な書き込みのために、間隔を cater に設定できます。デフォルトの 1 秒。プロデューサーがファイルを書き込む速度が非常に遅い場合は、速度が長すぎる可能性があります。注記：FTP の場合、デフォルトの readLockCheckInterval は 5000 です。readLockTimeout の値は readLockCheckInterval よりも大きくなければなりませんが、thumb のルールは readLockCheckInterval よりも少なくとも 2 倍以上タイムアウトになるようにします。これは、読み取りロックプロセスでタイムアウトがヒットするまでにロックの取得を試行するために バブル時間が許可されるようにするために必要です。	1000	Long
readLockDeleteOrphanLock Files (lock)	Camel が適切にシャットダウンされない場合（JVM クラッシュなど）、ファイルシステム上に残っている可能性のある孤立した読み取りロックファイルを削除する際にマーカーファイルを使用したロックを削除するかどうか。このオプションを false に指定すると、孤立したロックファイルがあると、Camel がそのファイルを取得しようとしません。また、別のノードが同じ共有ディレクトリーからファイルを同時に読み取ることが原因として考えられます。	true	boolean
readLockLoggingLevel (lock)	読み取りロックを取得できなかったときに使用されるロギングレベル。デフォルトでは、WARN がログに記録されます。このレベルは OFF でロギングを持たずに変更できます。このオプションは、type の readLock（changed、fileLock、idempotent、idempotent-changed、idempotent-rename、rename）にのみ適用できます。	DEBUG	LoggingLevel
readLockMarkerFile (lock)	変更、名前変更、または排他的読み取りロックタイプでマーカーファイルを使用するかどうか。デフォルトでは、マーカーファイルが使用されるだけでなく、同じファイルを選択する他のプロセスに対して保護されます。このオプションを false に設定すると、この動作をオフにすることができます。たとえば、Camel アプリケーションによってマーカーファイルをファイルシステムに書き込みます。	true	boolean

Name	説明	デフォルト	Type
readLockMinAge (lock)	このオプションは、readLock=change にのみ適用されます。このオプションでは、読み取りロックの取得を試行する前にファイルに必要な最小期間を指定できます。たとえば、readLockMinAge=300s を使用して、ファイルを最後の 5 分前とする必要があります。これにより、指定された期間以上のファイルの取得を試みるため、変更した読み取りロックが高速化されます。	0	Long
readLockMinLength (lock)	このオプションは、readLock=changed にのみ適用されます。このオプションを使用すると、最小限のファイルの長さを設定できます。デフォルトでは、Camel はファイルにデータが含まれることを想定するため、デフォルト値は 1 です。このオプションをゼロに設定するには、ゼロ長のファイルを使用できます。	1	Long
readLockRemoveOnCommit (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションでは、ファイルの処理に成功し、コミットが行われるときに、べき等リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。デフォルトでは、このファイルは削除されません。これにより、競合状態が発生しないため、別のアクティブなノードがファイルを取得できなくなります。代わりに、べき等リポジトリーは、X 分後にファイル名のエントリーをエビクトするように設定するエビクションストラテジーをサポートする可能性があります。これにより、競合状態に関する問題がなくなります。	false	boolean
readLockRemoveOnRollback (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションでは、ファイルの処理に失敗し、ロールバックが行われるときに、べき等リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。このオプションが false の場合、ファイル名のエントリーが確認されます（ファイルがコミットしたかのように）。	true	boolean

Name	説明	デフォルト	Type
readLockTimeout (lock)	read-lock で対応している場合は、read-lock のオプションのタイムアウト（ミリ秒単位）。read-lock が許可されず、タイムアウトが発生すると、Camel はファイルをスキップします。次回のポーリング Camel はファイルを再度試行し、今回は読み取りロックが付与される可能性があります。0 以下の値を使用して、永久に指定します。現在、fileLock、change、および rename はタイムアウトに対応しています。注記： FTP の場合、デフォルトの readLockTimeout 値は 10000 ではなく 20000 です。readLockTimeout の値は readLockCheckInterval よりも大きくなければなりませんが、thumb のルールは readLockCheckInterval よりも少なくとも 2 倍以上タイムアウトになるようにします。これは、読み取りロックプロセスでタイムアウトがヒットするまでにロックの取得を試行するために バブル時間が許可されるようにするために必要です。	10000	Long
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long

Name	説明	デフォルト	Type
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
shuffle (sort)	ファイルの一覧をシャッフルする方法（ランダムな順序など）	false	boolean
sortBy (sort)	File 言語を使用した組み込みソート。入れ子のソートをサポートするため、ファイル名でソートされ、2 番目にグループの種類を変更した日付でソートできます。		文字列
sorter (sort)	java.util.Comparator クラスとしてのプラグ可能なソーター。		GenericFile<T>>
アカウント （セキュリティー）	ログインに使用するアカウント		文字列
disableSecureDataChannelDefaults （セキュリティー）	このオプションを使用して、セキュアなデータチャネルを使用する際にデフォルトのオプションを無効にします。これにより、execPbsz および execProt 設定がどのように使用されるかを完全に制御できます。デフォルトは false です。	false	boolean

Name	説明	デフォルト	Type
execPbsz (security)	セキュアなデータチャネルを使用する場合は、実行保護バッファーサイズを設定できます。		Long
execProt (security)	exec 保護レベルの PROT コマンド。c - Clear S - Safe(SSL protocol only)E - Confidential(SSL protocol only)P - Private		文字列
ftpClientKeyStore Parameters (security)	キーストアパラメーターの設定		マップ
ftpClientTrustStore Parameters (security)	トラストストアパラメーターの設定		マップ
isImplicit （セキュリティー）	セキュリティーモード(Implicit/Explicit)を設定します。true - Implicit Mode / False - Explicit Mode	false	boolean
パスワード （セキュリティー）	ログインに使用するパスワード		文字列
securityProtocol (security)	ベースとなるセキュリティープロトコルを設定します。	TLS	文字列
sslContextParameters (security)	リンク FtpsEndpointftpClientKeyStoreParameters の設定を上書きする JSSE 設定を取得し、ftpClientTrustStoreParameters リンク、FtpsConfigurationgetSecurityProtocol（）リンク。		SSLContextParameters
ユーザー名 （セキュリティー）	ログインに使用するユーザー名		文字列

第109章 GANGLIA コンポーネント

Camel バージョン 2.15 から利用可能

Ganglia 監視システムにメトリックとして値（メッセージボディー）を送信するメカニズムを提供します。 gmetric4j ライブラリーを使用します。 単一のプラットフォームで OS、JVM、およびビジネスプロセスからメトリクスを監視するのに、標準の Ganglia および JMXetric と併用できます。

JVM が実行されるマシンで Ganglia gmond エージェントが実行されている必要があります。 gmond はハートビートを Ganglia インフラストラクチャーに送信し、camel-ganglia は現在ハートビート自体を送信できません。

ほとんどの Linux システム（Debian、Ubuntu、Fedora、および EPEL を使用する RHEL/CentOS）では、Ganglia エージェントパッケージをインストールして、マルチキャスト設定を使用して自動的に実行します。 必要に応じて、通常の UDP ユニキャストを使用するように設定できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ganglia</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

109.1. URI 形式

```
ganglia:address:port[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

109.2. GANGLIA コンポーネントおよびエンドポイント URI オプション

Ganglia コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有設定の使用		GangliaConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ganglia エンドポイントは、URI 構文を使用して設定します。

```
ganglia:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

109.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
host	Ganglia サーバーのホスト名	239.2.11.71	文字列
port	Ganglia サーバーのポート	8649	int

109.2.2. クエリーパラメーター（13 パラメーター）：

Name	説明	デフォルト	Type
dmax (producer)	Ganglia が期限切れになると、メトリクス値をパージするまでの時間（秒単位）。0 に設定すると、値は gmond エージェントが再起動するまで Ganglia に残ります。	0	int
groupName (producer)	メトリクスが属するグループ。	java	文字列
metricName (producer)	メトリクスに使用する名前。	metric	文字列

Name	説明	デフォルト	Type
モード（プロデューサー）	MULTICAST または UNICAST を使用して UDP メトリクスパケットを送信します。	MULTICAST	UDPAddressingMode
prefix (producer)	メトリクス名の前にこの文字列とアンダースコアを付けます。		文字列
slope（プロデューサー）	slope	BOTH	GMetricSlope
spoofHostname (producer)	spoofing information IP:hostname		文字列
tmax (producer)	値が最新とみなされる最大時間（秒単位）。この後、Ganglia は値を期限切れとみなします。	60	int
ttl (producer)	マルチキャストを使用する場合は、パケットの TTL を設定します。	5	int
type (producer)	値の型	STRING	GMetricType
ユニット（プロデューサー）	メトリックを構成する計測単位（ウィジェット、litres、バイトなど）。k(kilo)や m(milli)などの接頭辞を含めないでください。他のツールは後でスケールする可能性があります。値はスケーリング解除する必要があります。		文字列
wireFormat31x (producer)	Ganglia 3.1.0 以降のバージョンの Wire 形式を使用します。Ganglia 3.0.x 以前を使用するには、このパラメーターを false に設定します。	true	boolean
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

109.3. メッセージボディー

ボディーの値（文字列または数値型など）は Ganglia システムに送信されます。

109.4. 戻り値/応答

ganglia は、一方向 UDP またはマルチキャストを使用してメトリクスを送信します。 メッセージボ

ディーへの応答や変更はありません。

109.5. 例

109.5.1. 文字列メトリクスの送信

メッセージボディーは String に変換され、メトリクス値として送信されます。 数値メトリクスとは異なり、文字列の値はチャート化できませんが、Ganglia はそれらを報告に利用できるようにします。 すべての Ganglia ホストページの上部にある os_version 文字列は、String メトリクスの例です。

```
from("direct:string.for.ganglia")
    .setHeader(GangliaConstants.METRIC_NAME, simple("my_string_metric"))
    .setHeader(GangliaConstants.METRIC_TYPE, GMetricType.STRING)
    .to("direct:ganglia.tx");

from("direct:ganglia.tx")
    .to("ganglia:239.2.11.71:8649?mode=MULTICAST&prefix=test");
```

109.5.2. 数値メトリクスの送信

```
from("direct:value.for.ganglia")
    .setHeader(GangliaConstants.METRIC_NAME, simple("widgets_in_stock"))
    .setHeader(GangliaConstants.METRIC_TYPE, GMetricType.UINT32)
    .setHeader(GangliaConstants.METRIC_UNITS, simple("widgets"))
    .to("direct:ganglia.tx");

from("direct:ganglia.tx")
    .to("ganglia:239.2.11.71:8649?mode=MULTICAST&prefix=test");
```

第110章 GEOCODER コンポーネント

Camel バージョン 2.12 から利用可能

ジオコーダー： コンポーネントは、特定のアドレスまたは逆引きルックアップでジオコード（latitude および longitude）を検索するために使用されます。コンポーネントは、Google Geocoder ライブラリーに Java API を使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-geocoder</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

110.1. URI 形式

```
geocoder:address:name[?options]
geocoder:latlng:latitude,longitude[?options]
```

110.2. オプション

Geocoder コンポーネントにはオプションがありません。

Geocoder エンドポイントは、URI 構文を使用して設定します。

```
geocoder:address:latlng
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

110.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
address	アドレスのプレフィックスが付くジオアドレス：		文字列

Name	説明	デフォルト	Type
latlng	geolatitude および longitude でプレフィックスとして latlng を付ける必要があります。		文字列

110.2.2. クエリーパラメーター（14 パラメーター）：

Name	説明	デフォルト	Type
clientId (producer)	このクライアント ID で google プレミアムを使用する		文字列
clientKey (producer)	このクライアントキーで google プレミアムを使用する		文字列
headersOnly (producer)	ヘッダーで Exchange のみを強化し、ボディーをそのまま残すかどうか。	false	boolean
言語 （プロデューサー）	使用する言語。	en	文字列
httpClientConfigurer (advanced)	認証メカニズムの設定など、プロデューサーやコンシューマーによって作成された新しい HttpClient インスタンスのカスタム設定ストラテジーを登録します。		HttpClientConfigurer
httpConnectionManager (advanced)	カスタムの HttpConnectionManager を使用して接続を管理するには		HttpConnectionManager
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
proxyAuthDomain (proxy)	プロキシー NTML 認証用のドメイン		文字列
proxyAuthHost (proxy)	プロキシー NTML 認証のオプションホスト		文字列
proxyAuthMethod (proxy)	Basic、Digest、または NTLM としてプロキシーの認証方法。		文字列
proxyAuthPassword (proxy)	プロキシー認証のパスワード		文字列

Name	説明	デフォルト	Type
proxyAuthUsername (proxy)	プロキシー認証のユーザー名		文字列
proxyHost (proxy)	プロキシーホスト名		文字列
proxyPort (proxy)	プロキシーポート番号		整数

110.3. データフォーマットの交換

Camel はボディーを com.google.code.geocoder.model.GeocodeResponse タイプとして提供します。
また、アドレスが 「current」 の場合、応答は現在の場所の JSON 表現を持つ String タイプになります。

headersOnly オプションを true に設定すると、メッセージボディーはそのまま残り、エクスチェンジにヘッダーのみが追加されます。

110.4. メッセージヘッダー

ヘッダー	説明
CamelGeoCoderStatus	必須。geocoder ライブラリーからのステータスコード。ステータスが **GeocoderStatus.OK** の場合は、追加のヘッダーがエンリッチされます。
CamelGeoCoderAddress	フォーマットされたアドレス
CamelGeoCoderLat	場所のフラット化。
CamelGeoCoderLng	ロケーションの長さ。
CamelGeoCoderLatlng	場所のlatitude および longitude。コンマで区切ります。
CamelGeoCoderCity	都市の長い名前。
CamelGeoCoderRegionCode	リージョンコード。
CamelGeoCoderRegionName	リージョン名。

ヘッダー	説明
CamelGeoCoderCountryLong	国の長さの名前。
CamelGeoCoderCountryShort	国の短縮名。

使用中の利用可能なデータおよびモード（アドレスと latlng）によっては、すべてのヘッダーを指定できない点に注意してください。

110.5. サンプル

以下の例では、平易な Paris の latitude と longitude を、France

```
from("direct:start")
  .to("geocoder:address:Paris, France")
```

CamelGeoCoderAddress でヘッダーを指定する場合、エンドポイント設定を上書きするため、Copenhagen の場所を取得するために、以下のようにヘッダーでメッセージを送信できます。

```
template.sendBodyAndHeader("direct:start", "Hello", GeoCoderConstants.ADDRESS,
"Copenhagen, Denmark");
```

latitude および longitude のアドレスを取得するには、以下を行います。

```
from("direct:start")
  .to("geocoder:latlng:40.714224,-73.961452")
  .log("Location ${header.CamelGeocoderAddress} is at lat/lng:
${header.CamelGeocoderLatlng} and in country ${header.CamelGeoCoderCountryShort}")
```

ログの対象

```
Location 285 Bedford Avenue, Brooklyn, NY 11211, USA is at lat/lng:
40.71412890,-73.96140740 and in country US
```

現在の場所を取得するには、以下のように「current」をアドレスとして使用できます。

```
from("direct:start")
  .to("geocoder:address:current")
```

第111章 GIT コンポーネント

Camel バージョン 2.16 から利用可能

git: コンポーネントを使用すると、汎用の Git リポジトリーと連携できます。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-git</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

URI 形式

```
git://localRepositoryPath[?options]
```

111.1. URI オプション

プロデューサーを使用すると、特定のリポジトリーで操作を実行できます。
コンシューマーは、特定のリポジトリーでコミット、タグ、およびブランチを使用できます。

Git コンポーネントにはオプションがありません。

Git エンドポイントは URI 構文を使用して設定します。

```
git:localPath
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

111.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
localPath	**必要な** ローカルリポジトリーパス		文字列

111.1.2. クエリーパラメーター（13 パラメーター）：

Name	説明	デフォルト	Type
branchName (common)	作業するブランチ名		文字列
パスワード (common)	リモートリポジトリーのパスワード		文字列
remoteName (common)	pull などの特定の操作で使用するリモートリポジトリー名		文字列
remotePath (common)	リモートリポジトリーパス		文字列
tagName (common)	作業するタグ名		文字列
username (common)	リモートリポジトリーのユーザー名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
Type (consumer)	コンシューマータイプ		GitType
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
allowEmpty (producer)	空の git コミットを管理するフラグ	true	boolean
操作 （プロデューサー）	リポジトリーで実行する操作		文字列

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

111.2. メッセージヘッダー

Name	デフォルト値	Type	コンテキスト	説明
CamelGitOperation	**null**	文字列	プロデューサー	endpoint オプションとして指定されていない場合、リポジトリーで実行する操作
CamelGitFilename	**null**	文字列	プロデューサー	add 操作のファイル名
CamelGitCommitMessage	**null**	文字列	プロデューサー	コミット操作に関連するコミットメッセージ
CamelGitCommitUsername	**null**	文字列	プロデューサー	コミット操作のコミットユーザー名
CamelGitCommitEmail	**null**	文字列	プロデューサー	コミット操作のコミットメール
CamelGitCommitId	**null**	文字列	プロデューサー	コミット ID
CamelGitAllowEmpty	**null**	ブール値	プロデューサー	空の git コミットを管理するフラグ

111.3. プロデューサーの例

以下は、ファイル test.java をローカルリポジトリーに追加し、master ブランチの特定のメッセージ

でコミットし、リモートリポジトリーにプッシュするプロデューサーのルートの例になります。

```
from("direct:start")
        .setHeader(GitConstants.GIT_FILE_NAME, constant("test.java"))
        .to("git:///tmp/testRepo?operation=add")
        .setHeader(GitConstants.GIT_COMMIT_MESSAGE, constant("first commit"))
        .to("git:///tmp/testRepo?operation=commit")
        .to("git:///tmp/testRepo?
operation=push&remotePath=https://foo.com/test/test.git&username=xxx&password=xxx")
```

111.4. コンシューマーの例

以下は、コミットを消費するコンシューマーのルートの例です。

```
from("git:///tmp/testRepo?type=commit")
                        .to(....)
```

第112章 GITHUB COMPONENT

Camel バージョン 2.15 から利用可能

GitHub コンポーネントは、 egit-github をカプセル化して GitHub API と対話します。現在、新しいプル要求、プル要求のコメント、タグ、コミットのポーリングを行います。 また、プル要求にコメントを作成したり、プル要求を完全に閉じることもできます。

このエンドポイントは Webhook ではなく、単純なポーリングに依存します。理由は次のとおりです。

- 信頼性/安定性の懸念
- ポーリングしているペイロードのタイプは通常大きくありません（さらに、ページングは API で利用可能です）
- Webhook が一般にアクセスできない場所で実行されているアプリケーションをサポートする必要がある

GitHub API はかなり拡張されていることに注意してください。 そのため、このコンポーネントは、追加の対話を提供するために簡単に拡張できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-github</artifactId>
  <version>${camel-version}</version>
</dependency>
```

112.1. URI 形式

```
github://endpoint[?options]
```

112.2. 必須オプション：

これらはエンドポイントから直接設定できることに注意してください。

GitHub コンポーネントにはオプションがありません。

GitHub エンドポイントは URI 構文を使用して設定します。

```
github:type/branchName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

112.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
type	実行する git 操作が **必要**		GitHubType
branchName	ブランチ名		文字列

112.2.2. クエリーパラメーター（12 パラメーター）：

Name	説明	デフォルト	Type
oauthToken (common)	GitHub OAuth トークン（ユーザー名とパスワードが指定されない限り必要）		文字列
パスワード (common)	oauthToken が指定されていない限り GitHub パスワードが必要		文字列
repoName (common)	**必要な** GitHub リポジトリー名		文字列
repoOwner (common)	**必要な** GitHub リポジトリーの所有者（組織）		文字列
username (common)	oauthToken が指定されていない限り GitHub ユーザー名が必要		文字列

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
encoding （プロデューサー）	git コミットファイルの取得時に指定されたエンコーディングを使用するには、以下を実行します。		文字列
状態 （プロデューサー）	git コミットステータスの状態を設定するには、以下を実行します。		文字列
targetUrl (producer)	git コミットステータスターゲット URL を設定するには、以下を実行します。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

112.3. コンシューマーエンドポイント：

エンドポイント	コンテキスト	本文タイプ
pullRequest	ポーリング	org.eclipse.egit.github.core.PullRequest
pullRequestComment	ポーリング	org.eclipse.egit.github.core.Comment（一般的なプル要求に関するコメント）または org.eclipse.egit.github.core.CommitComment（プル要求の差分のインラインコメント）
tag	ポーリング	org.eclipse.egit.github.core.RepositoryTag

エンドポイント	コンテキスト	本文タイプ
commit	ポーリング	org.eclipse.egit.github.core.RepositoryCommit

112.4. プロデューサーエンドポイント：

エンドポイント	ボディー	メッセージヘッダー
pullRequestComment	string（コメントテキスト）	- GitHubPullRequest（整数）(REQUIRED): プル要求番号。 - GitHubInResponseTo（整数）: プル要求の差分のインラインコメントに応答する場合に必要です。オフのままにすると、プル要求に関する一般的なコメントが想定されます。
closePullRequest	none	- GitHubPullRequest（整数）(REQUIRED): プル要求番号。
createIssue(From Camel 2.18)	文字列（本文テキストを発行する）	- GitHubIssueTitle（文字列）(REQUIRED): 問題のタイトル

第113章 GZIP DATAFORMAT

Camel バージョン 2.0 で利用可能

GZip Data Format は、メッセージ圧縮と圧縮解除形式です。これは、Zip DataFormat で使用されるものと同じ deflate アルゴリズムを使用しますが、追加のヘッダーが提供されます。この形式は、一般的な gzip/gunzip ツールにより生成されます。GZip 圧縮を使用してマーシャリングされたメッセージは、エンドポイントで消費される直前に GZip の展開を使用してアンマーシャリングすることができます。圧縮機能は、大きな XML およびテキストベースのペイロードを扱う場合や、以前は gzip ツールを使用して開始されたメッセージを読み取る場合に非常に便利です。

113.1. オプション

GZip データフォーマットは、以下に示す 1 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

113.2. マーシャリング

この例では、gzip 圧縮形式を使用する圧縮ペイロードに通常のテキスト/XML ペイロードをマーシャリングし、MY_QUEUE という ActiveMQ キューを送信します。

```
from("direct:start").marshal().gzip().to("activemq:queue:MY_QUEUE");
```

113.3. アンマーシャリング

この例では、MY_QUEUE という ActiveMQ キューの gzip 形式のペイロードを元のフォーマットにアンマーシャリングし、これを UnGZippedMessageProcessor に転送します。

```
from("activemq:queue:MY_QUEUE").unmarshal().gzip().process(new
UnGZippedMessageProcessor());
```

113.4. 依存関係

このデータ形式は camel-core で提供されるため、追加の依存関係は必要ありません。

第114章 GOOGLE BIGQUERY COMPONENT

Camel バージョン 2.20 で利用可能

114.1. コンポーネントの説明

Google Bigquery コンポーネントは、Google Client Services API 経由で Cloud BigQuery Infrastructure へのアクセスを提供します。

現在の実装は gRPC を使用しません。

現在の実装は BigQuery のクエリーをサポートしません（プロデューサーのみ）。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-google-bigquery</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

114.2. 認証設定

Google BigQuery コンポーネントの認証は、GCP サービスアカウントで使用するためのものです。詳細は『Google Cloud Platform Auth Guide』を参照してください。

Google セキュリティー認証情報は、以下のいずれかのオプションを使用して明示的に設定できます。

- サービスアカウントメールおよびサービスアカウントキー（PEM 形式）
- GCP 認証情報ファイルの場所

両方が設定されている場合、Service Account Email/Key が優先されます。

または、接続ファクトリーが アプリケーションのデフォルト認証情報にフォールバックすることを意味します。

OBS!デフォルトの認証情報ファイルの場所は、GOOGLE_APPLICATION_CREDENTIALS 環境変数により設定可能です。

Service Account Email および Service Account Key は、それぞれ client_email および private_key として GCP JSON 認証情報ファイルで確認することができます。

114.3. URI 形式

```
google-bigquery://project-id:datasetId[:tableId]?[options]
```

114.4. オプション

Google BigQuery コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
projectId (producer)	Google Cloud Project Id		文字列
datasetId (producer)	BigQuery Dataset Id		文字列
connectionFactory (producer)	Bigquery サービスへの接続を取得する ConnectionFactory。指定しない場合は、デフォルトのものが使用されます。		GoogleBigQueryConnectionFactory
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Google BigQuery エンドポイントは、URI 構文を使用して設定します。

google-bigquery:projectId:datasetId:tableName

以下の path パラメーターおよびクエリーパラメーターを使用します。

114.4.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
projectId	**必要な** Google Cloud Project ID		文字列
datasetId	**必要な** BigQuery データセット ID		文字列
tableId	BigQuery テーブル ID		文字列

114.4.2. クエリーパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
connectionFactory (producer)	Bigquery サービスへの接続を取得する ConnectionFactory。指定しない場合には、デフォルトが使用されます。		GoogleBigQueryConnectionFactory
useAsInsertId (producer)	id の挿入として使用するフィールド名		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

114.5. メッセージヘッダー

Name	タイプ	説明
CamelGoogleBigQuery.TableSuffix	**文字列**	データの挿入時に使用するテーブルサフィックス

Name	タイプ	説明
CamelGoogleBigQuery.InsertId	**文字列**	データの挿入時に使用する InsertId
CamelGoogleBigQuery.PartitionDecorator	**文字列**	データの挿入時に使用するパーティションを示すパーティションデコレーター
CamelGoogleBigQuery.TableId	**文字列**	データの送信先となるテーブル ID。指定された場合はエンドポイント設定が上書きされます。

114.6. プロデューサーエンドポイント

プロデューサーエンドポイントは BigQuery の個別およびグループ化されたエクスチェンジを受け入れて配信できます。グループ化されたエクスチェンジには、Exchange.GROUPED_EXCHANGE プロパティーセットがあります。

Goole BigQuery プロデューサーは、異なるテーブルサフィックスまたはパーティションデコレーターが指定されていない限り、単一の api 呼び出しでグループ化されたエクスチェンジを送信します。この場合、データが正しい接尾辞またはパーティションデコレーターで記述されるようにします。

Google BigQuery エンドポイントは、ペイロードがマップまたはマップのリストであることを想定します。マップを含むペイロードは単一行を挿入し、マップのリストが含まれるペイロードがリストの各エントリーの行を挿入します。

114.7. テンプレートテーブル

参照： https://cloud.google.com/bigquery/streaming-data-into-bigquery#template-tables

テンプレート化されたテーブルは、GoogleBigQueryConstants.TABLE_SUFFIX ヘッダーを使用し

て指定できます。

つまり、以下のルートは、1 日ごとにテーブルを作成し、シャードされたレコードを挿入します。

```
from("direct:start")
.header(GoogleBigQueryConstants.TABLE_SUFFIX, "_${date:now:yyyyMMdd}")
.to("google-bigquery:sampleDataset:sampleTable")
```

このユースケースには、パーティション設定を使用することが推奨されます。

114.8. パーティション設定

参照： https://cloud.google.com/bigquery/docs/creating-partitioned-tables

テーブルの作成時にパーティションを指定します。設定データを自動的に別のテーブルに分割します。データを挿入する場合は、エクスチェンジに GoogleBigQueryConstants.PARTITION_DECORATOR ヘッダーを設定して特定のパーティションを指定できます。

114.9. データの整合性の確保

参照： https://cloud.google.com/bigquery/streaming-data-into-bigquery#dataconsistency

挿入 ID は、ヘッダー GoogleBigQueryConstants.INSERT_ID を使用するか、クエリーパラメーター useAsInsertId を指定してエクスチェンジに設定できます。挿入 ID は、1 行ごとに指定する必要があるため、ペイロードがリストの場合、エクスチェンジヘッダーは使用できません。ペイロードがリストの場合、GoogleBigQueryConstants.INSERT_ID は無視されます。このような場合には、クエリーパラメーター useAsInsertId を使用します。

第115章 GOOGLE カレンダーコンポーネント

Camel バージョン 2.15 から利用可能

Google Calendar コンポーネントは、Google Calendar Web API 経由で Google カレンダー へのアクセスを提供します。

Google カレンダーは、Google アカウントの認証に OAuth 2.0 プロトコル を使用し、ユーザーデータへのアクセスを承認します。このコンポーネントを使用する前に、 アカウントを作成し、OAuth 認証情報を生成 する必要があります。認証情報は、clientId、clientSecret、および refreshToken で構成されます。有効期間の長い refreshToken を生成するための便利なリソースは OAuth playground です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-google-calendar</artifactId>
    <version>2.15.0</version>
</dependency>
```

115.1. 1.GOOGLE カレンダーオプション

Google Calendar コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有設定の使用		GoogleCalendarConfiguration
clientFactory (advanced)	クライアントを作成するために GoogleCalendarClientFactory をファクトリーとして使用します。デフォルトでは BatchGoogleCalendarClientFactory を使用します。		GoogleCalendarClient Factory
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Google カレンダーエンドポイントは、URI 構文を使用して設定します。

```
google-calendar:apiName/methodName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

115.1.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
apiName	**必要な** 操作の種類		GoogleCalendarApiName
methodName	選択した操作に使用するサブ操作が **必要**		文字列

115.1.2. クエリーパラメーター（14 パラメーター）：

Name	説明	デフォルト	Type
accessToken (common)	OAuth 2 アクセストークンこれは通常 1 時間後に期限切れになるため、長期の使用には refreshToken が推奨されます。		文字列
applicationName (common)	Google カレンダーアプリケーション名。例： camel-google-calendar/1.0		文字列
clientId (common)	カレンダーアプリケーションのクライアント ID		文字列
clientSecret (common)	カレンダーアプリケーションのクライアントシークレット		文字列
emailAddress (common)	Google サービスアカウント の emailAddress。		文字列
inBody (common)	エクスチェンジ In Body で渡されるパラメーターの名前を設定します。		文字列
p12FileName (common)	Google サービスアカウントで使用する秘密鍵を持つ p12 ファイルの名前。		文字列

Name	説明	デフォルト	Type
refreshToken (common)	OAuth 2 更新トークン。これを使用すると、Google カレンダーコンポーネントは、現在期限切れになるたびに新しい accessToken を取得できます。アプリケーションの有効期間は必要となります。		文字列
scopes (common)	カレンダーアプリケーションが必要とするパーミッションをユーザーアカウントに指定します。複数のスコープはコンマで区切ることができます。詳細は、https://developers.google.com/google-apps/calendar/auth を参照してください。	https://www.googleapis.com/auth/calendar	文字列
User (common)	アプリケーションがサービスアカウントフローで権限を借用しようとしているユーザーのメールアドレス		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

115.2. URI 形式

GoogleCalendar コンポーネントは以下の URI 形式を使用します。

```
google-calendar://endpoint-prefix/endpoint?[options]
```

エンドポイントプレフィックスは以下のいずれかになります。

- acl
- calendars
- チャネル
- colors
- events
- freebusy
- list
- settings

115.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、エンドポイント接頭辞の後にエンドポイント名および関連するオプションを使用できます。省略形のエイリアスは、一部のエンドポイントに使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションは [] で表されます。エンドポイントに必須オプションがない場合は、[] オプションのセットの 1 つを指定する必要があります。プロデューサーエンドポイントは、特別なオプション inBody を使用することもできます。そのオプションには、値が Camel Exchange In メッセージに含まれる endpoint オプションの名前が含まれる必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は CamelGoogleCalendar.<option> 形式である必要があります。inBody

オプションはメッセージヘッダーを上書きする（例： body = option）は CamelGoogleCalendar.option ヘッダーを上書きすることに注意してください。

115.4. コンシューマーエンドポイント

プロデューサーエンドポイントはいずれもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、consumer. プレフィックスと共に Scheduled Poll Consumer オプション を使用してエンドポイント呼び出しをスケジュールできます。アレイまたはコレクションを返すコンシューマーエンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 度実行されます。

115.5. メッセージヘッダー

URI オプションは、CamelGoogleCalendar. プレフィックスを持つプロデューサーエンドポイントのメッセージヘッダーに指定できます。

115.6. メッセージボディー

すべての結果メッセージ本文は、GoogleCalendarComponent によって使用される基礎となる API によって提供されるオブジェクトを使用します。プロデューサーエンドポイントは、inBody エンドポイント URI パラメーターに受信メッセージボディーのオプション名を指定できます。アレイまたはコレクションを返すエンドポイントでは、コンシューマーエンドポイントはすべての要素を個別のメッセージにマッピングします。

第116章 GOOGLE ドライブコンポーネント

Camel バージョン 2.14 から利用可能

Google Drive コンポーネントは、Google Drive Web API 経由で Google ドライブファイルストレージサービス へのアクセスを提供します。

Google ドライブは OAuth 2.0 プロトコル を使用して Google アカウントを認証し、ユーザーデータへのアクセスを承認します。このコンポーネントを使用する前に、アカウントを作成し、OAuth 認証情報を生成 する必要があります。認証情報は、clientId、clientSecret、および refreshToken で構成されます。有効期間の長い refreshToken を生成するための便利なリソースは OAuth playground です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-google-drive</artifactId>
    <version>2.14-SNAPSHOT</version>
</dependency>
```

116.1. URI 形式

GoogleDrive コンポーネントは以下の URI 形式を使用します。

```
google-drive://endpoint-prefix/endpoint?[options]
```

エンドポイントプレフィックスは以下のいずれかになります。

- **drive-about**
- **drive-apps**

- *drive-changes*
- *drive-channels*
- *drive-children*
- *drive-comments*
- *drive-files*
- *drive-parents*
- *drive-permissions*
- *drive-properties*
- *drive-realtime*
- *drive-replies*
- *drive-revisions*

116.2. GOOGLEDRIVECOMPONENT

Google Drive コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有設定の使用		GoogleDrive Configuration

Name	説明	デフォルト	Type
clientFactory (advanced)	クライアントを作成するために GoogleCalendarClientFactory をファクトリーとして使用します。デフォルトでは BatchGoogleDriveClientFactory を使用します。		GoogleDriveClientFactory
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Google ドライブエンドポイントは、URI 構文を使用して設定します。

```
google-drive:apiName/methodName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

116.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
apiName	**必要な** 操作の種類		GoogleDriveApiName
methodName	選択した操作に使用するサブ操作が **必要**		文字列

116.2.2. クエリーパラメーター（12 パラメーター）：

Name	説明	デフォルト	Type
accessToken (common)	OAuth 2 アクセストークンこれは通常 1 時間後に期限切れになるため、長期の使用には refreshToken が推奨されます。		文字列
applicationName (common)	Google ドライブのアプリケーション名。例： camel-google-drive/1.0		文字列

Name	説明	デフォルト	Type
clientFactory (common)	クライアントを作成するために GoogleCalendarClientFactory をファクトリーとして使用します。デフォルトでは BatchGoogleDriveClientFactory を使用します。		GoogleDriveClientFactory
clientId (common)	ドライブアプリケーションのクライアント ID		文字列
clientSecret (common)	ドライブアプリケーションのクライアントシークレット		文字列
inBody (common)	エクスチェンジ In Body で渡されるパラメーターの名前を設定します。		文字列
refreshToken (common)	OAuth 2 更新トークン。これを使用すると、Google カレンダーコンポーネントは、現在期限切れになるたびに新しい accessToken を取得できます。アプリケーションの有効期間は必要となります。		文字列
scopes (common)	ドライブアプリケーションがユーザーアカウントに割り当てるパーミッションのレベルを指定します。詳細は、https://developers.google.com/drive/web/scopes を参照してください。		リスト
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

116.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、エンドポイント接頭辞の後にエンドポイント名および関連するオプションを使用できます。省略形のエイリアスは、一部のエンドポイントに使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションは [] で表されます。エンドポイントに必須オプションがない場合は、[] オプションのセットの 1 つを指定する必要があります。プロデューサーエンドポイントは、特別なオプション inBody を使用することもできます。そのオプションには、値が Camel Exchange In メッセージに含まれる endpoint オプションの名前が含まれる必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は CamelGoogleDrive.<option> 形式である必要があります。inBody オプションはメッセージヘッダーを上書きすることに注意してください。つまり、エンドポイントオプション inBody=option は CamelGoogleDrive.option ヘッダーを上書きすることに注意してください。

エンドポイントおよびオプションの詳細は、 https://developers.google.com/drive/v2/reference/の API ドキュメントを参照してください。

116.4. コンシューマーエンドポイント

プロデューサーエンドポイントはいずれもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、consumer. プレフィックスと共に Scheduled Poll Consumer オプション を使用してエンドポイント呼び出しをスケジュールできます。アレイまたはコレクションを返すコンシューマーエンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 度実行されます。

116.5. メッセージヘッダー

すべての URI オプションは、CamelGoogleDrive 接頭辞とともに、プロデューサーエンドポイントのメッセージヘッダーに提供できます。

116.6. メッセージボディー

すべての結果メッセージ本文は、GoogleDriveComponent が使用する基盤の API が提供するオブジェクトを使用します。プロデューサーエンドポイントは、inBody エンドポイント URI パラメーターに受信メッセージボディーのオプション名を指定できます。アレイまたはコレクションを返すエンドポイントでは、コンシューマーエンドポイントはすべての要素を個別のメッセージにマッピングします。

第117章 GOOGLE メールコンポーネント

Camel バージョン 2.15 から利用可能

Google Mail コンポーネントは、 Google Mail Web API 経由で Gmail へのアクセスを提供します。

Google Mail は OAuth 2.0 プロトコル を使用して Google アカウントを認証し、ユーザーデータへのアクセスを承認します。このコンポーネントを使用する前に、 アカウントを作成し、OAuth 認証情報を生成 する必要があります。認証情報は、clientId、clientSecret、および refreshToken で構成されます。有効期間の長い refreshToken を生成するための便利なリソースは OAuth playground です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-google-mail</artifactId>
    <version>2.15-SNAPSHOT</version>
</dependency>
```

117.1. URI 形式

GoogleMail コンポーネントは以下の URI 形式を使用します。

```
google-mail://endpoint-prefix/endpoint?[options]
```

エンドポイントプレフィックスは以下のいずれかになります。

- **attachments**
- **ドラフト**
- **history**
- **labels**

- *メッセージ*
- *threads*
- *users*

117.2. GOOGLEMAILCOMPONENT

Google Mail コンポーネントは以下の 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有設定の使用		GoogleMailConfiguration
clientFactory (advanced)	クライアントを作成するために GoogleCalendarClientFactory をファクトリーとして使用します。デフォルトでは BatchGoogleMailClientFactory を使用します。		GoogleMailClientFactory
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Google Mail エンドポイントは、URI 構文を使用して設定します。

```
google-mail:apiName/methodName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

117.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
apiName	**必要な** 操作の種類		GoogleMailApiName
methodName	選択した操作に使用するサブ操作が **必要**		文字列

117.2.2. クエリーパラメーター（11 パラメーター）：

Name	説明	デフォルト	Type
accessToken (common)	OAuth 2 アクセストークンこれは通常 1 時間後に期限切れになるため、長期の使用には refreshToken が推奨されます。		文字列
applicationName (common)	Google メールアプリケーション名。例： camel-google-mail/1.0		文字列
clientId (common)	メールアプリケーションのクライアント ID		文字列
clientSecret (common)	メールアプリケーションのクライアントシークレット		文字列
inBody (common)	エクスチェンジ In Body で渡されるパラメーターの名前を設定します。		文字列
refreshToken (common)	OAuth 2 更新トークン。これを使用すると、Google カレンダーコンポーネントは、現在期限切れになるたびに新しい accessToken を取得できます。アプリケーションの有効期間は必要となります。		文字列
scopes (common)	メールアプリケーションがユーザーアカウントに割り当てるパーミッションのレベルを指定します。詳細は、https://developers.google.com/gmail/api/auth/scopes を参照してください。		リスト
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

117.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、エンドポイント接頭辞の後にエンドポイント名および関連するオプションを使用できます。省略形のエイリアスは、一部のエンドポイントに使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションは [] で表されます。エンドポイントに必須オプションがない場合は、[] オプションのセットの 1 つを指定する必要があります。プロデューサーエンドポイントは、特別なオプション inBody を使用することもできます。そのオプションには、値が Camel Exchange In メッセージに含まれる endpoint オプションの名前が含まれる必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は CamelGoogleMail.<option> 形式である必要があります。inBody オプションはメッセージヘッダー（例： body = option）が CamelGoogleMail.option ヘッダーを上書きすることに注意してください。

エンドポイントおよびオプションの詳細は、
https://developers.google.com/gmail/api/v1/reference/ の API ドキュメントを参照してください。

117.4. コンシューマーエンドポイント

プロデューサーエンドポイントはいずれもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、consumer. プレフィックスと共に Scheduled Poll Consumer オプション を使用してエンドポイント呼び出しをスケジュールできます。アレイまたはコレクションを返すコン

シューマーエンドポイントは、要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 度実行されます。

117.5. メッセージヘッダー

すべての URI オプションは、CamelGoogleMail. プレフィックスを持つプロデューサーエンドポイントのメッセージヘッダーに提供できます。

117.6. メッセージボディー

すべての結果メッセージ本文は、GoogleMailComponent で使用される基礎となる API によって提供されるオブジェクトを使用します。プロデューサーエンドポイントは、inBody エンドポイント URI パラメーターに受信メッセージボディーのオプション名を指定できます。アレイまたはコレクションを返すエンドポイントでは、コンシューマーエンドポイントはすべての要素を個別のメッセージにマッピングします。

第118章 GOOGLE PUBSUB コンポーネント

Camel バージョン 2.19 から利用可能

Google Pubsub コンポーネントは、Google Client Services API 経由で Cloud Pub/Sub Infrastructure へのアクセスを提供します。

現在の実装は gRPC を使用しません。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-google-pubsub</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

118.1. URI 形式

GoogleMail コンポーネントは以下の URI 形式を使用します。

```
google-pubsub://project-id:destinationName?[options]
```

Destination Name にはトピックまたはサブスクリプション名を指定できます。

118.2. オプション

Google Pubsub コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
connectionFactory (common)	使用する接続ファクトリーを設定します。接続認証情報を明示的に管理できる機能を提供します。- キーファイルへのパス - サービスアカウントキー/電子メールのペアです。		GooglePubsubConnection Factory

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Google Pubsub エンドポイントは、URI 構文を使用して設定します。

```
google-pubsub:projectId:destinationName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

118.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
projectId	**必要な** プロジェクト ID		文字列
destinationName	**必要な** 宛先名		文字列

118.2.2. クエリーパラメーター（9 パラメーター）：

Name	説明	デフォルト	Type
ackMode (common)	AUTO = Exchange は完了時に ack された/nack になります。NONE = ダウンストリームのプロセスでは、明示的に ack/nack を行う必要がある	AUTO	AckMode
concurrentConsumers (common)	サブスクリプションから消費している並列ストリームの数	1	整数
connectionFactory (common)	PubSub サービスへの接続を取得する ConnectionFactory。指定しない場合には、デフォルトが使用されます。		GooglePubsubConnection Factory
loggerId (common)	親ルートと一致する必要がある場合に使用するロガー ID		文字列

Name	説明	デフォルト	Type
maxMessagesPerPoll (common)	単一の API 呼び出しでサーバーから受信するメッセージの最大数。	1	整数
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

118.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、PubSub の個別およびグループ化されたエクスチェンジを受け付け、配信できます。グループ化されたエクスチェンジには、Exchange.GROUPED_EXCHANGE プロパティーセットがあります。

Google PubSub ではペイロードが byte[] 配列であることが予想され、Producer エンドポイントは以下を送信します。

- ***文字列ボディーは UTF-8 としてエンコードされる byte[] としてエンコードされる***
- ***byte[] ボディー is***

- それ以外はすべて byte[] 配列にシリアライズされます。

メッセージヘッダー GooglePubsubConstants として設定されたマップ。ATTRIBUTES は PubSub 属性として送信されます。エクスチェンジが PubSub に配信されると、PubSub メッセージ ID はヘッダー GooglePubsubConstants.MESSAGE_ID に割り当てられます。

118.4. コンシューマーエンドポイント

Google PubSub は、サブスクリプションで設定オプションとして設定された期間内にメッセージが確認されていない場合は、メッセージを再配信します。

エクスチェンジの処理が完了すると、コンポーネントはメッセージを承認します。

ルートが例外をスローすると、エクスチェンジは失敗とマークされ、コンポーネントはメッセージを NACK します。すぐに再配信されます。

メッセージを ack/nack するために、コンポーネントはヘッダー GooglePubsubConstants.ACK_ID として保存された Acknowledgement ID を使用します。ヘッダーが削除されるか、改ざんされる場合に、ack が失敗し、メッセージが ack 期限後に再度送出されます。

118.5. メッセージヘッダー

コンシューマーエンドポイントによって設定されるヘッダー：

- GooglePubsubConstants.MESSAGE_ID
- GooglePubsubConstants.ATTRIBUTES
- GooglePubsubConstants.PUBLISH_TIME
- GooglePubsubConstants.ACK_ID

118.6. メッセージボディー

コンシューマーエンドポイントは、メッセージの内容を byte[] として返します。これは、基となるシステムが送信するのと同じです。コンテンツを変換/アンマーシャリングするルートは up です。

118.7. 認証設定

Google Pubsub コンポーネントの認証は、GCP サービスアカウントで使用するためのものです。詳細は『Google Cloud Platform Auth Guide』を参照してください。

Google セキュリティー認証情報は、以下のいずれかのオプションを使用して明示的に設定できます。

- サービスアカウントメールおよびサービスアカウントキー（PEM 形式）
- GCP 認証情報ファイルの場所

両方が設定されている場合、Service Account Email/Key が優先されます。

または、接続ファクトリーがアプリケーションのデフォルト認証情報にフォールバックすることを意味します。

OBS!デフォルトの認証情報ファイルの場所は、GOOGLE_APPLICATION_CREDENTIALS 環境変数により設定可能です。

Service Account Email および Service Account Key は、それぞれ client_email および private_key として GCP JSON 認証情報ファイルで確認することができます。

118.8. ロールバックと再配信

Google PubSub のロールバックは、Acknowledgement Deadline のアイデアに依存します。これは、Google PubSub が確認応答を受信するまでの期間です。確認が受信されていないと、メッセージは再配信されます。

Google は、メッセージの期限を拡張する API を提供します。

詳細は、Google PubSub のドキュメントを参照してください。

そのため、ロールバックは基本的にゼロ値を持つ期限の延長 API 呼び出しであり、期限に達しており、メッセージが次のコンシューマーに再配信できます。

メッセージヘッダー GooglePubsubConstants.ACK_DEADLINE を秒単位で設定すると、メッセージの再配信をロールバックについて明示的に設定できます。

第119章 GROOVY 言語

Camel バージョン 1.3 で利用可能

Camel は他のスクリプト言語間で Groovy をサポートし、DSL または Xml 設定 で式または述語を使用できるようにします。

Groovy 式を使用するには、以下の Java コードを使用します。

```
... groovy("someGroovyExpression") ...
```

たとえば、groovy 関数を使用して、メッセージフィルター や Recipient List の式として述語を作成できます。

119.1. GROOVY オプション

Groovy 言語は、以下に示す 1 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
trim	**true**	**ブール値**	値をトリミングして先頭および末尾の空白と改行を削除するかどうか。

119.2. GROOVY SHELL のカスタマイズ

カスタムの GroovyShell インスタンスを Groovy 式で使用する必要がある場合があります。カスタムの GroovyShell を提供するには、 org.apache.camel.language.groovy.GroovyShellFactory SPI インターフェースの実装を Camel レジストリーに追加します。たとえば、以下の Bean を Spring コンテキストに追加した後

```
public class CustomGroovyShellFactory implements GroovyShellFactory {

  public GroovyShell createGroovyShell(Exchange exchange) {
    ImportCustomizer importCustomizer = new ImportCustomizer();
    importCustomizer.addStaticStars("com.example.Utils");
    CompilerConfiguration configuration = new CompilerConfiguration();
    configuration.addCompilationCustomizers(importCustomizer);
    return new GroovyShell(configuration);
```

```
  }

}
```

...Camel は、デフォルトのインスタンスではなく、カスタムの GroovyShell インスタンス（カスタムの静的インポートを含む）を使用します。

119.3. 例

```
// lets route if a line item is over $100
from("queue:foo").filter(groovy("request.lineItems.any { i -> i.value > 100 }")).to("queue:bar")
```

Spring DSL の場合：

```
<route>
  <from uri="queue:foo"/>
  <filter>
    <groovy>request.lineItems.any { i -> i.value > 100 }</groovy>
    <to uri="queue:bar"/>
  </filter>
</route>
```

119.4. SCRIPTCONTEXT

JSR-223 スクリプト言語 ScriptContext は、すべて ENGINE_SCOPE に設定された以下の属性で設定されます。

属性	型	値
context	**org.apache.camel.CamelContext**	Camel コンテキスト（groovy では使用できません）
camelContext	**org.apache.camel.CamelContext**	Camel コンテキスト

属性	型	値
exchange	**org.apache.camel.Exchange**	現在のエクスチェンジ
request	**org.apache.camel.Message**	メッセージ（IN メッセージ）
response	**org.apache.camel.Message**	**非推奨**: OUT メッセージ。null（デフォルトで null）の場合に OUT メッセージ。代わりに IN メッセージを使用してください。
properties	**org.apache.camel.builder.script.PropertiesFunction**	Camel 2.9: スクリプトから CamelsProperties コンポーネントを簡単に使用できるようにする **解決** メソッドで機能。以下に例を示します。

明示的な DSL サポートのある言語の一覧は、「Scripting Languages」を参照してください。

119.5. スクリプト化ENGINE への追加の引数

Camel 2.8 から利用可能

CamelScriptArguments キーで Camel メッセージのヘッダーを使用して、ScriptingEngine に追加の引数を提供できます。
以下の例を参照してください。

119.6. プロパティー関数の使用

Camel 2.9 で利用可能

スクリプトから Properties コンポーネントを使用してプロパティープレースホルダーを検索する必要がある場合は、少し複雑にします。たとえば、プロパティープレースホルダーの値でヘッダー名 myHeader を設定します。このキーは「foo」というヘッダーで提供されます。

```
.setHeader("myHeader").groovy(""context.resolvePropertyPlaceholders( + '{{' +
request.headers.get(&#39;foo&#39;) + '}}' + ")")
```

Camel 2.9 以降では、Properties 関数と同じ例が簡単に使用できるようになりました。

```
.setHeader("myHeader").groovy("properties.resolve(request.headers.get(&#39;foo&#39;))")
```

119.7. 外部リソースからのスクリプトの読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、「classpath:」、"file:"、または "http: " などのリソースから Camel に読み込むことができます。
これは、「 resource:scheme:location」 構文を使用して行われます。たとえば、実行可能なクラスパスのファイルを参照します。

```
.setHeader("myHeader").groovy("resource:classpath:mygroovy.groovy")
```

119.8. 複数のステートメントスクリプトからの結果を取得する方法

Camel 2.14 から利用可能

scripteengine evale メソッドは、複数の統計スクリプトを実行すると Null を返します。Camel は、値セットから「result」のキーを使用してスクリプト結果の値を検索するようになりました。複数のステートメントスクリプトがある場合は、結果変数の値をスクリプトの戻り値として設定する必要があります。

```
bar = "baz";
# some other statements ...
# camel take the result value as the script evaluation result
result = body * 2 + 1
```

119.9. 依存関係

camel ルートでスクリプト言語を使用するには、JSR-223 スクリプトエンジンを統合する camel-

script の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-script</artifactId>
  <version>x.x.x</version>
</dependency>
```

第120章 GRPC コンポーネント

Camel バージョン 2.19 から利用可能

gRPC コンポーネントを使用すると、HTTP/2 トランスポートで プロトコルバッファー(protobuf)交換形式を使用して、RPC(Remote Procedure Call) サービスを呼び出すことができます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-grpc</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

120.1. URI 形式

```
grpc://service[?options]
```

120.2. エンドポイントオプション

gRPC コンポーネントにはオプションがありません。

gRPC エンドポイントは URI 構文を使用します。

```
grpc:host:port/service
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

120.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
host	gRPC サーバーのホスト名が **必要です**。これは、プロデューサーの使用時にコンシューマーまたはリモートサーバーのホスト名である場合の localhost または 0.0.0.0 です。		文字列
port	**必須**: gRPC のローカルまたはリモートサーバーポート		int
サービス	プロトコルバッファー記述子ファイルからの **必要な** 完全修飾サービス名（パッケージドットサービス定義名）		文字列

120.2.2. クエリーパラメーター（25 パラメーター）：

Name	説明	デフォルト	Type
flowControlWindow (common)	HTTP/2 フロー制御ウインドウサイズ(MiB)	1048576	int
maxMessageSize (common)	受信/送信の最大メッセージサイズ(MiB)	4194304	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumerStrategy (consumer)	このオプションは、ストリーミングモードでサービス要求と応答を処理する最上位ストラテジーを指定します。集約ストラテジーが選択された場合は、すべてのリクエストがリストに累積され、フローに転送され、累積された応答が送信者に送信されます。伝播ストラテジーが選択されている場合、リクエストはストリームに送信され、応答は送信側に即座に返されます。	伝搬	GrpcConsumerStrategy
forwardOnCompleted (consumer)	onCompleted イベントが Camel ルートにプッシュされるべきかどうかを決定します。	false	boolean

Name	説明	デフォルト	Type
forwardOnError (consumer)	onError イベントを Camel ルートにプッシュする必要があるかどうかを決定します。例外はメッセージボディーとして設定されます。	false	boolean
maxConcurrentCallsPerConnection (consumer)	各受信サーバー接続で許可される同時呼び出しの最大数。	2147483647	int
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
メソッド （プロデューサー）	gRPC メソッド名		文字列
producerStrategy (producer)	リモート gRPC サーバーと通信するために使用されるモード。SIMPLE モードでは、単一のエクスチェンジがリモートプロシージャー呼び出しに変換されます。STREAMING モードでは、すべてのエクスチェンジは同じリクエスト内に送信されます（受信者 gRPC サービスの入力と出力は 'stream' タイプである必要があります）。	SIMPLE	GrpcProducerStrategy
streamRepliesTo (producer)	STREAMING クライアントモードを使用する場合、レスポンスを転送するエンドポイントを指定します。		文字列
userAgent (producer)	サーバーに渡されるユーザーエージェントのヘッダー		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
authenticationType (security)	SSL/TLS ネゴシエーションの前に付ける認証方法タイプ	NONE	GrpcAuthType
jwtAlgorithm （セキュリティー）	JSON Web Token sign algorithm	HMAC256	JwtAlgorithm

Name	説明	デフォルト	Type
jwtIssuer (security)	JSON Web Token issuer		文字列
jwtSecret (security)	JSON Web トークンシークレット		文字列
jwtSubject (security)	JSON Web トークンサブジェクト		文字列
keyCertChainResource (security)	X.509 証明書チェーンファイルの PEM 形式のリンク		文字列
keyPassword (security)	PKCS8 秘密鍵ファイルのパスワード		文字列
keyResource (security)	PEM 形式のリンクの PKCS8 秘密鍵ファイルリソース		文字列
negotiationType (security)	HTTP/2 通信に使用されるセキュリティーネゴシエーションタイプを特定します。	PLAIN TEXT	NegotiationType
serviceAccountResource (security)	Google Cloud SDK によってサポートされる JSON 形式のリソースリンクのサービスアカウントキーファイル		文字列
trustCertCollectionResource (security)	リモートエンドポイントの証明書を検証するための PEM 形式の信頼される証明書コレクションファイルリソース		文字列

120.3. トランスポートセキュリティーおよび認証サポート（ CAMEL 2.20から利用可能）

以下の 認証 メカニズムは gRPC に組み込まれたもので、このコンポーネントで利用できます。

- *SSL/TLS: gRPC には SSL/TLS 統合があり、SSL/TLS を使用してサーバーを認証し、クライアントとサーバー間で交換されたすべてのデータを暗号化します。クライアントが相互認証の証明書を提供できるようにオプションのメカニズムを使用できます。*

- *Google でのトークンベースの認証： gRPC は、要求および応答にメタデータベースの認証情報をアタッチするための汎用メカニズムを提供します。gRPC で Google API にアクセスする際にアクセストークンを取得するための追加のサポートが提供されます。通常、このメカニズムとチャンネル上の SSL/TLS を使用する必要があります。*

これらの機能を有効にするには、以下のコンポーネントプロパティーの組み合わせを設定する必要があります。

num.	オプション	パラメーター	値	必須/オプション
1	**SSL/TLS**	negotiationType	TLS	必須
		keyCertChainResource		必須
		keyResource		必須
		keyPassword		任意
		trustCertCollectionResource		任意
2	**Google API を使用したトークンベースの認証**	authenticationType	GOOGLE	必須
		negotiationType	TLS	必須
		serviceAccountResource		必須
3	**カスタム JSON Web トークン実装認証**	authenticationType	JWT	必須
		negotiationType	NONE または TLS	オプション。TLS/SSL はこのタイプのチェックは行いませんが、強く推奨されます。
		jwtAlgorithm	HMAC256(default)または (HMAC384, HMAC512)	任意
		jwtSecret		必須
		jwtIssuer		任意
		jwtSubject		任意

*現在、**OpenSSL** での **TLS** は **TLS** コンポーネントで **gRPC** を使用するための推奨される方法です。通常、**ALLPN** への **JDK** の使用は遅くなるため、**HTTP2** に必要な暗号に対応していない可能性があり*

ます。この関数はコンポーネントには実装されません。

120.4. GRPC プロデューサーリソースタイプのマッピング

以下の表は、受信パラメーターおよび送信パラメーターのタイプ（簡易またはストリーム）ならびに呼び出しスタイル（同期または非同期）に応じて、メッセージボディーのオブジェクトのタイプを示しています。非同期スタイルで受信ストリームパラメーターを使用した手順は許可されないことに注意してください。

呼び出しスタイル	リクエストタイプ	応答タイプ	リクエストボディーのタイプ	結果ボディーのタイプ
同期	simple	simple	オブジェクト	オブジェクト
同期	simple	stream	オブジェクト	List<Object>
同期	stream	simple	許可されていません	許可されていません
同期	stream	stream	許可されていません	許可されていません
非同期	simple	simple	オブジェクト	List<Object>
非同期	simple	stream	オブジェクト	List<Object>
非同期	stream	simple	オブジェクトまたは List<Object>	List<Object>
非同期	stream	stream	オブジェクトまたは List<Object>	List<Object>

120.5. GRPC コンシューマーヘッダー（コンシューマーの呼び出し後にインストールされる）

ヘッダー名	説明	可能な値
CamelGrpcMethodName	コンシューマーサービスによって処理されるメソッド名	
CamelGrpcEventType	送信した要求から受信したイベントタイプ	onNext、onCompleted または onError
CamelGrpcUserAgent	これが指定されている場合、指定されたエージェントは gRPC ライブラリーのユーザーエージェント情報の先頭に追加します。	

120.6. 例

以下は、ホストおよびポートパラメーターで呼び出される簡単な同期メソッドです。

```
from("direct:grpc-sync")
.to("grpc://remotehost:1101/org.apache.camel.component.grpc.PingPong?
method=sendPing&synchronous=true");
```

```
<route>
    <from uri="direct:grpc-sync" />
    <to uri="grpc://remotehost:1101/org.apache.camel.component.grpc.PingPong?
method=sendPing&synchronous=true"/>
</route>
```

非同期メソッド呼び出し

```
from("direct:grpc-async")
.to("grpc://remotehost:1101/org.apache.camel.component.grpc.PingPong?
method=pingAsyncResponse");
```

コンシューマーストラテジーの伝播を持つ gRPC サービスコンシューマー

```
from("grpc://localhost:1101/org.apache.camel.component.grpc.PingPong?
consumerStrategy=PROPAGATION")
.to("direct:grpc-service");
```

ストリーミングプロデューサーストラテジーのある gRPC サービスプロデューサー（「ストリーム」モードを入出力として使用するサービスが必要）

```
from("direct:grpc-request-stream")
.to("grpc://remotehost:1101/org.apache.camel.component.grpc.PingPong?
method=PingAsyncAsync&producerStrategy=STREAMING&streamRepliesTo=direct:grpc-
response-stream");

from("direct:grpc-response-stream")
.log("Response received: ${body}");
```

gRPC サービスコンシューマー TLS/SLL セキュリティーネゴシエーションの有効化

```
from("grpc://localhost:1101/org.apache.camel.component.grpc.PingPong?
consumerStrategy=PROPAGATION&negotiationType=TLS&keyCertChainResource=file:src/te
st/resources/certs/server.pem&keyResource=file:src/test/resources/certs/server.key&trustCert
```

```
CollectionResource=file:src/test/resources/certs/ca.pem")
.to("direct:tls-enable")
```

カスタム JSON Web Token 実装認証を使用した gRPC サービスプロデューサー

```
from("direct:grpc-jwt")
.to("grpc://localhost:1101/org.apache.camel.component.grpc.PingPong?
method=pingSyncSync&synchronous=true&authenticationType=JWT&jwtSecret=supersecure
dsecret");
```

120.7. 設定

Protocol Buffer Compiler(protoc)ツールを呼び出して、カスタムプロジェクトの .proto(protocol buffer definition)ファイルから Java ソースファイルを生成する Maven Protocol Buffers Plugin を使用することが推奨されます。このプラグインは、手順リクエストと応答クラス、ビルダー、gRPC の手順もスタブクラスを生成します。

以下の手順が必要です。

プロジェクトの pom.xml の < build > タグまたは ${os.detected.classifier} パラメーター内にオペレーティングシステムおよび CPU アーキテクチャー検出拡張を手動で挿入します。

```
<extensions>
  <extension>
    <groupId>kr.motd.maven</groupId>
    <artifactId>os-maven-plugin</artifactId>
    <version>1.4.1.Final</version>
  </extension>
</extensions>
```

プロジェクトの pom.xml の gRPC および protobuf Java コードジェネレータープラグイン <plugins > タグを挿入します。

```
<plugin>
  <groupId>org.xolstice.maven.plugins</groupId>
  <artifactId>protobuf-maven-plugin</artifactId>
  <version>0.5.0</version>
  <configuration>
    <protocArtifact>com.google.protobuf:protoc:${protobuf-
version}:exe:${os.detected.classifier}</protocArtifact>
    <pluginId>grpc-java</pluginId>
    <pluginArtifact>io.grpc:protoc-gen-grpc-java:${grpc-
version}:exe:${os.detected.classifier}</pluginArtifact>
  </configuration>
```

```
    <executions>
      <execution>
        <goals>
          <goal>compile</goal>
          <goal>compile-custom</goal>
          <goal>test-compile</goal>
          <goal>test-compile-custom</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
```

120.8. 詳細情報は、これらのリソースを参照してください。

gRPC プロジェクトサイト

Maven プロトコルバッファープラグイン

120.9. 関連項目

- はじめに
- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- プロトコルバッファーデータフォーマット

第121章 GUAVA EVENTBUS COMPONENT

Camel バージョン 2.10 で利用可能

Google Guava EventBus を使用すると、コンポーネント間で明示的に登録する必要なしに、コンポーネント間のパブリッシュサブスクライブスタイルの通信が可能になります（つまり、お互いを認識）。guava-eventbus: コンポーネントは Camel と Google Guava EventBus インフラストラクチャー間の統合ブリッジを提供します。後者のコンポーネントでは、Guava EventBus で交換されたメッセージを Camel ルートに透過的に転送することができます。EventBus コンポーネントを使用すると、Camel エクスチェンジのボディーを Guava EventBus にルーティングすることもできます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-guava-eventbus</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

121.1. URI 形式

```
guava-eventbus:busName[?options]
```

busName は、Camel レジストリーにある com.google.common.eventbus.EventBus インスタンスの名前を表します。

121.2. オプション

Guava EventBus コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
eventBus (common)	指定の Guava EventBus インスタンスの使用		EventBus

Name	説明	デフォルト	Type
listenerInterface (common)	Subscribe アノテーションでマークされたメソッドを持つインターフェース。動的プロキシーはインターフェース上で作成され、EventBus リスナーとして登録できます。マルチイベントリスナーを作成し、deadEvent を適切に処理する際に特に便利です。このオプションは eventClass オプションと併用することはできません。		class<?>
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Guava EventBus エンドポイントは、URI 構文を使用して設定します。

```
guava-eventbus:eventBusRef
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

121.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
eventBusRef	指定の名前でレジストリーから Guava EventBus を検索するには、以下を実行します。		文字列

121.2.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
eventClass (common)	ルートのコンシューマー側で使用すると、EventBus から受信したイベントを、class および eventClass のクラスインスタンスにフィルターします。このオプションの null 値は java.lang.Object に設定することと同じになります。つまり、コンシューマーはイベントバスに受信されたすべてのメッセージを取得します。このオプションは、listenerInterface オプションと併用することはできません。		class<?>

Name	説明	デフォルト	Type
listenerInterface (common)	Subscribe アノテーションでマークされたメソッドを持つインターフェース。動的プロキシーはインターフェース上で作成され、EventBus リスナーとして登録できます。マルチイベントリスナーを作成し、deadEvent を適切に処理する際に特に便利です。このオプションは eventClass オプションと併用することはできません。		class<?>
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

121.3. 用途

ルートのコンシューマー側で guava-eventbus コンポーネントを使用すると、Guava EventBus に送信されたメッセージをキャプチャーし、Camel ルートに転送します。Guava EventBus コンシューマーは受信メッセージを 非同期 に処理します。

```
SimpleRegistry registry = new SimpleRegistry();
EventBus eventBus = new EventBus();
registry.put("busName", eventBus);
CamelContext camel = new DefaultCamelContext(registry);

from("guava-eventbus:busName").to("seda:queue");

eventBus.post("Send me to the SEDA queue.");
```

ルートのプロデューサー側で guava-eventbus コンポーネントを使用すると、Camel エクスチェンジのボディーが Guava EventBus インスタンスに転送されます。

```
SimpleRegistry registry = new SimpleRegistry();
EventBus eventBus = new EventBus();
registry.put("busName", eventBus);
CamelContext camel = new DefaultCamelContext(registry);

from("direct:start").to("guava-eventbus:busName");

ProducerTemplate producerTemplate = camel.createProducerTemplate();
producer.sendBody("direct:start", "Send me to the Guava EventBus.");

eventBus.register(new Object(){
  @Subscribe
  public void messageHander(String message) {
    System.out.println("Message received from the Camel: " + message);
  }
});
```

121.4. DEADEVENT に関する考慮事項

Guava EventBus の設計によって生じる制限により、@Subscribe メソッドでアノテーションが付けられたクラスを作成せずにリスナーが受け取るイベントクラスを指定できないことに注意してください。この制限は、eventClass オプションが指定されたエンドポイントが実際にすべてのイベント (java.lang.Object)をリッスンし、実行時にプログラム的に適切なメッセージをフィルターすることを意味します。以下は、Camel コードベースからの適切な抜粋を示しています。

```
@Subscribe
public void eventReceived(Object event) {
  if (eventClass == null || eventClass.isAssignableFrom(event.getClass())) {
    doEventReceived(event);
...
```

このアプローチの欠点は、Camel が使用する EventBus インスタンスが com.google.common.eventbus.DeadEvent 通知を生成しないことです。Camel が正確に指定されたイベントのみをリッスンするようにするには（そのため、DeadEvent サポートを有効にする）、listenerInterface エンドポイントオプションを使用します。Camel は、後者 オプションで指定したインターフェース上で動的プロキシーを作成し、インターフェースハンドラーメソッドが指定するメッセージのみをリッスンします。以下は、特定のイベント インスタンスのみを処理する単一のメソッドを持つリスナーインターフェースの例になります。

```
package com.example;

public interface CustomListener {

  @Subscribe
```

```
    void eventReceived(SpecificEvent event);

  }
```

上記のリスナーは、以下のようにエンドポイント定義で使用できます。

```
from("guava-eventbus:busName?
listenerInterface=com.example.CustomListener").to("seda:queue");
```

121.5. 複数のタイプのイベントの使用

Guava EventBus コンシューマーが使用する複数のタイプのイベントを定義するには、listenerInterface エンドポイントオプションを使用します。リスナーインターフェースは、@Subscribe アノテーションでマークされた複数のメソッドを提供できます。

```
package com.example;

public interface MultipleEventsListener {

  @Subscribe
  void someEventReceived(SomeEvent event);

  @Subscribe
  void anotherEventReceived(AnotherEvent event);

}
```

上記のリスナーは、以下のようにエンドポイント定義で使用できます。

```
from("guava-eventbus:busName?
listenerInterface=com.example.MultipleEventsListener").to("seda:queue");
```

121.6. HAWTDB

Camel 2.3 の時点で利用可能

HawtDB は非常に軽量で埋め込み可能なキー値データベースです。これにより、Camel とともに Aggregator などのさまざまな Camel 機能に永続的なサポートを提供できます。

非推奨

HawtDB プロジェクトは非推奨となり、軽量で埋め込み可能なキー値データベースとして leveldb に置き換えられます。leveldb を簡単に使用するには、leveldbjni プロジェクトがあります。Apache ActiveMQ プロジェクトは、今後は kahadb に代わる主要なファイルベースのメッセージストアとして leveldb を使用する計画です。

これの代わりに使用する camel-leveldb コンポーネントがあります。

HawtDB 1.4 以前の問題

HawtDB 1.4 以前にはバグがあり、filestore では未使用の領域が解放されません。これは、ファイルが拡大中であることを意味します。これは、Camel 2.5 以降に同梱される HawtDB 1.5 で修正されました。

提供されている現在の機能：

- *HawtDBAggregationRepository*

121.6.1. Using HawtDBAggregationRepository

HawtDBAggregationRepository は AggregationRepository で、集約されたメッセージを即座に永続化します。これにより、デフォルトのアグリゲーターは AggregationRepository のみのメモリーで使用されるため、メッセージを緩めることはありません。

このオプションには以下のオプションが含まれます。

オプション	型	説明
repositoryName	文字列	必須のリポジトリー名。複数のリポジトリーに共有 **HawtDBFile** を使用できます。
persistentFileName	文字列	永続ストレージのファイル名。起動時にファイルが存在しない場合には、新しいファイルが作成されます。

オプション	型	説明
bufferSize	int	ファイルストアにマッピングされるメモリーセグメントバッファーのサイズ。デフォルトでは 8mb です。値はバイト単位です。
sync	boolean	**HawtDBFile** が書き込みで同期するべきかどうか。デフォルトは **true** です。書き込み時に同期することで、すべての書き込みがディスクにスプーピングされるのを常に待機し、アップデートが悪化しないようにします。このオプションを無効にすると、多数の書き込みをバッチ処理する際に HawtDB が自動同期されます。
pageSize	short	メモリーページのサイズ。デフォルトでは 512 バイトです。値はバイト単位です。
hawtDBFile	HawtDBFile	既存の設定された **org.apache.camel.component.hawtdb.HawtDBFile** インスタンスを使用します。
returnOldExchange	boolean	get 操作によって古い既存の Exchange が返されるかどうか。デフォルトでは、このオプションは最適化時に古いエクスチェンジを必要としないため、最適化するために **false** になっています。
useRecovery	boolean	リカバリーが有効になっているかどうか。このオプションはデフォルトで **true** です。Camel Aggregator の自動リカバリーの失敗されたエクスチェンジのリカバリーを有効にして、再提出します。
recoveryInterval	Long	リカバリーが有効になっている場合、バックグラウンドタスクは、失敗したエクスチェンジをスキャンして再送信するために、失敗したエクスチェンジをスキャンするたびに実行されます。デフォルトでは、この間隔は 5000 ミリ秒です。
maximumRedeliveries	int	リカバリーされたエクスチェンジの再配信試行の最大数を制限できます。有効にすると、再配信の試行に失敗した場合、エクスチェンジはデッドレターチャネルに移動します。デフォルトでは、このオプションは無効になっています。このオプションを使用する場合は、**deadLetterUri** オプションも指定する必要があります。
deadLetterUri	文字列	リカバリーされたエクスチェンジが移動される Dead Letter Channel のエンドポイント URI。このオプションを使用する場合は、max **Redeliveries** オプションも指定する必要があります。
optimisticLocking	**false**	Camel 2.12: 楽観的ロックを有効にするには、複数の Camel アプリケーションが同じ HawtDB ベースの集約リポジトリーを共有するクラスター環境で必要になることがよくあります。

repositoryName オプションを指定する必要があります。次に persistentFileName または hawtDBFile のいずれかを指定する必要があります。

121.6.2. 永続時に保持されるもの

HawtDBAggregationRepository は、pidtDBAggregationRepository と互換性のある データ タイプのみを保持します。データ型がドロップされたタイプではなく、WARN がログに記録されます。そして、メッセージ 本文とメッセージヘッダーのみが永続化 さ れます。エクスチェンジプロパティー は永続化され ません。

121.6.3. 復元

HawtDBAggregationRepository はデフォルトで失敗したエクスチェンジを復元します。これは、永続ストアで失敗したエクスチェンジの有無をスキャンするバックグラウンドタスクを設定することによって行われます。checkInterval オプションを使用して、このタスクの実行頻度を設定できます。リカバリーはトランザクションとして機能し、Camel が失敗したエクスチェンジのリカバリーおよび再配信を試行します。復元されたエクスチェンジは永続ストアから復元され、再送信されて再度送信されます。

Exchange がリカバリーまたは再配信されるときに、以下のヘッダーが設定されます。

ヘッダー	Type	説明
Exchange.REDELIVERED	ブール値	エクスチェンジが再配信されることを示すには、true に設定します。
Exchange.REDELIVERY_COUNTER	整数	1 から開始する再配信の試行。

Exchange が正常に処理された場合にのみ、完了とマークされます。これは、確認 メソッドが AggregationRepository で呼び出されると発生します。つまり、同じエクスチェンジが再び失敗すると、成功するまで再試行されます。

maximumRedeliveries オプションを使用して、特定のリカバリーされたエクスチェンジの再配信の最大試行回数を制限できます。また、maximumRedeliveries に達したときに Camel が Exchange を送信する場所を認識できるように deadLetterUri オプションも設定する必要があります。

この テストなど、camel-hawtdb のユニットテストにはいくつかの例があります。

121.6.3.1. Java DSL での HawtDBAggregationRepository の使用

この例では、target/data/hawtdb.dat ファイルで集約されたメッセージを永続化します。

121.6.3.2. Spring XML での HawtDBAggregationRepository の使用

同じ例で、代わりに Spring XML を使用します。

121.6.4. 依存関係

camel ルートで HawtDB を使用するには、camel-hawtdb の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hawtdb</artifactId>
  <version>2.3.0</version>
</dependency>
```

121.6.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- アグリゲーター
- コンポーネント

第122章 HAZELCAST コンポーネント

Camel バージョン 2.7 で利用可能

hazelcast- コンポーネントを使用すると、Hazelcast 分散データ収集/キャッシュと連携できます。Hazelcast は、Java（単一の jar）で完全に書かれたメモリーデータ構造です。マップ、マルチマップ（同じキー、n 値）、キュー、リスト、アトミック番号など、さまざまなデータストアの大きなパレットを提供します。Hazelcast を使用する主な理由は、単純なクラスターサポートです。ネットワークでマルチキャストを有効にしている場合は、追加設定なしで 100 ノードを使用してクラスターを実行できます。Hazelcast は、ノード間の n 個のコピー（デフォルトは 1）、キャッシュの永続性、ネットワーク設定（必要な場合）、ニアキャッシュ、エンベローティングなどの機能を追加するよう設定できます。詳細は、http://www.hazelcast.com/docs.jsp の Hazelcast のドキュメントを参照してください。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hazelcast</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

122.1. HAZELCAST コンポーネント

各コンポーネントの使用については、以下を参照してください。* map * multimap * queue * topic * list * seda * set * atomic number * cluster support(instance) * replicatedmap * ringbuffer

122.2. HAZELCAST リファレンスの使用

122.2.1. 名前

```
<bean id="hazelcastLifecycle" class="com.hazelcast.core.LifecycleService"
      factory-bean="hazelcastInstance" factory-method="getLifecycleService"
      destroy-method="shutdown" />

<bean id="config" class="com.hazelcast.config.Config">
  <constructor-arg type="java.lang.String" value="HZ.INSTANCE" />
</bean>

<bean id="hazelcastInstance" class="com.hazelcast.core.Hazelcast" factory-
method="newHazelcastInstance">
  <constructor-arg type="com.hazelcast.config.Config" ref="config"/>
</bean>
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route id="testHazelcastInstanceBeanRefPut">
```

```
        <from uri="direct:testHazelcastInstanceBeanRefPut"/>
        <setHeader headerName="CamelHazelcastOperationType">
            <constant>put</constant>
        </setHeader>
        <to uri="hazelcast-map:testmap?hazelcastInstanceName=HZ.INSTANCE"/>
    </route>

    <route id="testHazelcastInstanceBeanRefGet">
        <from uri="direct:testHazelcastInstanceBeanRefGet" />
        <setHeader headerName="CamelHazelcastOperationType">
            <constant>get</constant>
        </setHeader>
        <to uri="hazelcast-map:testmap?hazelcastInstanceName=HZ.INSTANCE"/>
        <to uri="seda:out" />
    </route>
</camelContext>
```

122.2.2. インスタンス別

```
<bean id="hazelcastInstance" class="com.hazelcast.core.Hazelcast"
      factory-method="newHazelcastInstance" />
<bean id="hazelcastLifecycle" class="com.hazelcast.core.LifecycleService"
      factory-bean="hazelcastInstance" factory-method="getLifecycleService"
      destroy-method="shutdown" />

<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route id="testHazelcastInstanceBeanRefPut">
        <from uri="direct:testHazelcastInstanceBeanRefPut"/>
        <setHeader headerName="CamelHazelcastOperationType">
            <constant>put</constant>
        </setHeader>
        <to uri="hazelcast-map:testmap?hazelcastInstance=#hazelcastInstance"/>
    </route>

    <route id="testHazelcastInstanceBeanRefGet">
        <from uri="direct:testHazelcastInstanceBeanRefGet" />
        <setHeader headerName="CamelHazelcastOperationType">
            <constant>get</constant>
        </setHeader>
        <to uri="hazelcast-map:testmap?hazelcastInstance=#hazelcastInstance"/>
        <to uri="seda:out" />
    </route>
</camelContext>
```

122.3. HAZELCAST インスタンスを OSGI サービスとして公開

OSGI コンテナーで動作する場合で、同じコンテナーのすべてのバンドルで hazelcast のインスタンスを 1 つ使用します。キャッシュのニーズを使用してインスタンスを OSGI サービスとして公開し、バンドルは hazelcast エンドポイントでサービスを参照することです。

122.3.1. バンドル A がインスタンスを作成し、それを OSGI サービスとして公開します。

```
<bean id="config" class="com.hazelcast.config.FileSystemXmlConfig">
    <argument type="java.lang.String" value="${hazelcast.config}"/>
</bean>

<bean id="hazelcastInstance" class="com.hazelcast.core.Hazelcast" factory-
method="newHazelcastInstance">
    <argument type="com.hazelcast.config.Config" ref="config"/>
</bean>

<!-- publishing the hazelcastInstance as a service -->
<service ref="hazelcastInstance" interface="com.hazelcast.core.HazelcastInstance" />
```

122.3.2. Bundle B がインスタンスを使用する

```
<!-- referencing the hazelcastInstance as a service -->
<reference ref="hazelcastInstance" interface="com.hazelcast.core.HazelcastInstance" />

<camelContext xmlns="http://camel.apache.org/schema/blueprint">
    <route id="testHazelcastInstanceBeanRefPut">
        <from uri="direct:testHazelcastInstanceBeanRefPut"/>
        <setHeader headerName="CamelHazelcastOperationType">
            <constant>put</constant>
        </setHeader>
        <to uri="hazelcast-map:testmap?hazelcastInstance=#hazelcastInstance"/>
    </route>

    <route id="testHazelcastInstanceBeanRefGet">
        <from uri="direct:testHazelcastInstanceBeanRefGet" />
        <setHeader headerName="CamelHazelcastOperationType">
            <constant>get</constant>
        </setHeader>
        <to uri="hazelcast-map:testmap?hazelcastInstance=#hazelcastInstance"/>
        <to uri="seda:out" />
    </route>
</camelContext>
```

第123章 HAZELCAST ATOMIC NUMBER コンポーネント

Camel バージョン 2.7 で利用可能

Hazelcast atomic number コンポーネントは Camel Hazelcast Components の 1 つで、Hazelcast atomic 番号にアクセスできます。アトミック番号は、グリッドの幅広い数字(long)を提供するオブジェクトです。

このエンドポイントにはコンシューマーがありません！

123.1. オプション

Hazelcast Atomic Number コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Atomic Number エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-atomicvalue:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

123.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

123.1.2. クエリーパラメーター（10 パラメーター）：

Name	説明	デフォルト	Type
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
defaultOperation (producer)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (producer)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (producer)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

123.2. ATOMIC NUMBER PRODUCER - TO("HAZELCAST-ATOMICVALUE:FOO")

このプロデューサーの操作： * setvalue（指定の値で数値を設定する）* get * increase(+1)* decrease(-1)* destroy

リクエストメッセージのヘッダー変数：

Name	タイプ	説明
CamelHazelcastOperationType	**文字列**	有効な値は setvalue、get、increase、decrease、destroy です。

123.2.1. セット の例：

Java DSL の場合

```
from("direct:set")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.SET_VALUE))
.toF("hazelcast-%sfoo", HazelcastConstants.ATOMICNUMBER_PREFIX);
```

Spring DSL の場合：

```
<route>
   <from uri="direct:set" />
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
   <setHeader headerName="hazelcast.operation.type">
      <constant>setvalue</constant>
   </setHeader>
   <to uri="hazelcast-atomicvalue:foo" />
</route>
```

メッセージボディー内で設定する値を指定します（値は 10 です）：
template.sendBody("direct:set", 10);

123.2.2. get の場合の例：

Java DSL の場合

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.GET))
.toF("hazelcast-%sfoo", HazelcastConstants.ATOMICNUMBER_PREFIX);
```

Spring DSL の場合 :

```
<route>
    <from uri="direct:get" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
    <setHeader headerName="hazelcast.operation.type">
        <constant>get</constant>
    </setHeader>
    <to uri="hazelcast-atomicvalue:foo" />
</route>
```

You can get the number with long body = template.requestBody("direct:get", null, Long.class);.

123.2.3. インクリメント の例 :

Java DSL の場合

```
from("direct:increment")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.INCREMENT))
.toF("hazelcast-%sfoo", HazelcastConstants.ATOMICNUMBER_PREFIX);
```

Spring DSL の場合 :

```
<route>
    <from uri="direct:increment" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
    <setHeader headerName="hazelcast.operation.type">
        <constant>increment</constant>
    </setHeader>
    <to uri="hazelcast-atomicvalue:foo" />
</route>
```

実際の値（インクリメント後）はメッセージのボディー内に提供されます。

123.2.4. デクリメント の例 :

Java DSL の場合

```
from("direct:decrement")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.DECREMENT))
.toF("hazelcast-%sfoo", HazelcastConstants.ATOMICNUMBER_PREFIX);
```

Spring DSL の場合 :

```
<route>
    <from uri="direct:decrement" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
    <setHeader headerName="hazelcast.operation.type">
        <constant>decrement</constant>
    </setHeader>
    <to uri="hazelcast-atomicvalue:foo" />
</route>
```

実際の値（デクリメント後）は、メッセージボディー内に提供されます。

123.2.5. destroyの例

Java DSL の場合

```
from("direct:destroy")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.DESTROY))
.toF("hazelcast-%sfoo", HazelcastConstants.ATOMICNUMBER_PREFIX);
```

Spring DSL の場合 :

```
<route>
    <from uri="direct:destroy" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
    <setHeader headerName="hazelcast.operation.type">
        <constant>destroy</constant>
    </setHeader>
    <to uri="hazelcast-atomicvalue:foo" />
</route>
```

第124章 HAZELCAST インスタンスコンポーネント

Camel バージョン 2.7 で利用可能

Hazelcast インスタンスコンポーネントは Camel Hazelcast Components の 1 つであり、クラスター内のキャッシュインスタンスの結合/ターリーブイベントを使用できます。Hazelcast は 1 つの「サーバーノード」でも理にかなっていますが、クラスター環境では非常に強力です。

このエンドポイントにはプロデューサーがありません！

124.1. オプション

Hazelcast Instance コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Instance エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-instance:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

124.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

124.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
defaultOperation (consumer)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (consumer)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (consumer)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
pollingTimeout (consumer)	キューコンシューマーのポーリングタイムアウトをポーリングモードで定義	10000	Long
poolSize (consumer)	キューコンシューマーエグゼキューターのプールサイズの定義	1	int
queueConsumerMode (consumer)	キューコンシューマーモードの定義： Listen または Poll	listen	HazelcastQueueConsumer Mode
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

124.2. INSTANCE CONSUMER - FROM("HAZELCAST-INSTANCE:FOO")

> ***The instance consumer fires if a new cache instance will join or leave the cluster.***

以下に例を示します。

```
fromF("hazelcast-%sfoo", HazelcastConstants.INSTANCE_PREFIX)
.log("instance...")
.choice()

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDED))
        .log("...added")
        .to("mock:added")
```

```
    .otherwise()
        .log("...removed")
        .to("mock:removed");
```

各イベントは、メッセージヘッダー内に以下の情報を提供します。

レスポンスメッセージ内のヘッダー変数：

Name	タイプ	説明
CamelHazelcastListenerTime	**Long**	イベントの時間（ミリ秒単位）
CamelHazelcastListenerType	**文字列**	マップコンシューマーは「instancelistener」に設定します。
CamelHazelcastListenerAction	**文字列**	イベントの型 - ここで **追加** または **削除** します。
CamelHazelcastInstanceHost	**文字列**	インスタンスのホスト名
CamelHazelcastInstancePort	**整数**	インスタンスのポート番号

第125章 HAZELCAST LIST コンポーネント

Camel バージョン 2.7 で利用可能

Hazelcast List コンポーネントは Camel Hazelcast コンポーネントの 1 つで、Hazelcast 分散リストにアクセスできます。

125.1. オプション

Hazelcast List コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast List エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-list:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

125.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

125.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
defaultOperation (common)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	キューコンシューマーのポーリングタイムアウトをポーリングモードで定義	10000	Long
poolSize (consumer)	キューコンシューマーエグゼキューターのプールサイズの定義	1	int
queueConsumerMode (consumer)	キューコンシューマーモードの定義： Listen または Poll	listen	HazelcastQueueConsumer Mode
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

125.2. LIST PRODUCER - TO("HAZELCAST-LIST:FOO")

*list producer provides 7 operations: * addAll * set * get * removevalue * removeAll * clear*

125.2.1. 追加 の例 :

```
from("direct:add")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.ADD))
.toF("hazelcast-%sbar", HazelcastConstants.LIST_PREFIX);
```

125.2.2. get の場合の例 :

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.GET))
.toF("hazelcast-%sbar", HazelcastConstants.LIST_PREFIX)
.to("seda:out");
```

125.2.3. setvalue の例 :

```
from("direct:set")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.SET_VALUE))
.toF("hazelcast-%sbar", HazelcastConstants.LIST_PREFIX);
```

125.2.4. removevalue の例 :

```
from("direct:removevalue")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.REMOVE_VALUE))
.toF("hazelcast-%sbar", HazelcastConstants.LIST_PREFIX);
```

CamelHazelcastObjectIndex ヘッダーはインデックス化の目的で使用されることに注意してください。

125.3. LIST CONSUMER - FROM("HAZELCAST-LIST:FOO")

リストコンシューマーは 2 つの操作を提供します。 * add * remove

```
fromF("hazelcast-%smm", HazelcastConstants.LIST_PREFIX)
   .log("object...")
   .choice()

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDED))
        .log("...added")
                .to("mock:added")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.REMOVED))
        .log("...removed")
                .to("mock:removed")
           .otherwise()
                .log("fail!");
```

第126章 HAZELCAST マップコンポーネント

Camel バージョン 2.7 で利用可能

Hazelcast Map コンポーネントは Camel Hazelcast コンポーネントの 1 つで、Hazelcast 分散マップマップにアクセスできます。

126.1. オプション

Hazelcast Map コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Map エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-map:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

126.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

126.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
defaultOperation (common)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	キューコンシューマーのポーリングタイムアウトをポーリングモードで定義	10000	Long
poolSize (consumer)	キューコンシューマーエグゼキューターのプールサイズの定義	1	int
queueConsumerMode (consumer)	キューコンシューマーモードの定義： Listen または Poll	listen	HazelcastQueueConsumer Mode
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

126.2. MAP CACHE PRODUCER - TO("HAZELCAST-MAP:FOO")

値をマップに保存する場合は、マップキャッシュプロデューサーを使用できます。

map キャッシュプロデューサーは、CamelHazelcastOperationType ヘッダーによって指定される以下の操作を提供します。

- *put*
- *putIfAbsent*
- *get*
- *getAll*

- *keySet*
- *containsKey*
- *containsValue*
- *削除*
- *更新*
- *query*
- *明確な*
- *エビクト*
- *evictAll*

すべての操作は、「hazelcast.operation.type」ヘッダー変数内に提供されます。Java DSL では、org.apache.camel.component.hazelcast.HazelcastOperation からの定数を使用できます。

リクエストメッセージのヘッダー変数：

Name	タイプ	説明
CamelHazelcastOperationType	**文字列**	すでに説明されているようにしてあります。

Name	タイプ	説明
CamelHazelcastObjectId	**文字列**	保存するオブジェクト ID。キャッシュ内でオブジェクトを検索します（クエリー操作には必要ありません）。

put および putIfAbsent 操作はエビクションメカニズムを提供します。

Name	タイプ	説明
CamelHazelcastObjectTtlValue	**整数**	TTL の値。
CamelHazelcastObjectTtlUnit	**java.util.concurrent.TimeUnit**	時間単位の値（DAYS / HOURS / MINUTES / ...

以下を使用してサンプルを呼び出すことができます。

```
template.sendBodyAndHeader("direct:[put|get|update|delete|query|evict]", "my-foo",
HazelcastConstants.OBJECT_ID, "4711");
```

126.2.1. 配置 の例 :

Java DSL の場合

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUT))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX);
```

Spring DSL の場合 :

```
<route>
    <from uri="direct:put" />
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
```

```
    <setHeader headerName="hazelcast.operation.type">
      <constant>put</constant>
    </setHeader>
    <to uri="hazelcast-map:foo" />
</route>
```

エビクションを 使用 するサンプル：

Java DSL の場合

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUT))
.setHeader(HazelcastConstants.TTL_VALUE, constant(Long.valueOf(1)))
.setHeader(HazelcastConstants.TTL_UNIT, constant(TimeUnit.MINUTES))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX);
```

Spring DSL の場合：

```
<route>
    <from uri="direct:put" />
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
    <setHeader headerName="hazelcast.operation.type">
      <constant>put</constant>
    </setHeader>
    <setHeader headerName="HazelcastConstants.TTL_VALUE">
      <simple resultType="java.lang.Long">1</simple>
    </setHeader>
    <setHeader headerName="HazelcastConstants.TTL_UNIT">
      <simple resultType="java.util.concurrent.TimeUnit">TimeUnit.MINUTES</simple>
    </setHeader>
    <to uri="hazelcast-map:foo" />
</route>
```

126.2.2. get の場合の例：

Java DSL の場合

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.GET))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX)
.to("seda:out");
```

Spring DSL の場合：

```
<route>
   <from uri="direct:get" />
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
   <setHeader headerName="hazelcast.operation.type">
      <constant>get</constant>
   </setHeader>
   <to uri="hazelcast-map:foo" />
   <to uri="seda:out" />
</route>
```

126.2.3. 更新 の例 :

Java DSL の場合

```
from("direct:update")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.UPDATE))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX);
```

Spring DSL の場合 :

```
<route>
   <from uri="direct:update" />
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
   <setHeader headerName="hazelcast.operation.type">
      <constant>update</constant>
   </setHeader>
   <to uri="hazelcast-map:foo" />
</route>
```

126.2.4. 削除 用のサンプル :

Java DSL の場合

```
from("direct:delete")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.DELETE))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX);
```

Spring DSL の場合 :

```
<route>
   <from uri="direct:delete" />
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
```

```
    <setHeader headerName="hazelcast.operation.type">
      <constant>delete</constant>
    </setHeader>
    <to uri="hazelcast-map:foo" />
</route>
```

126.2.5. クエリーの例

Java DSL の場合

```
from("direct:query")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.QUERY))
.toF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX)
.to("seda:out");
```

Spring DSL の場合：

```
<route>
    <from uri="direct:query" />
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
    <setHeader headerName="hazelcast.operation.type">
      <constant>query</constant>
    </setHeader>
    <to uri="hazelcast-map:foo" />
    <to uri="seda:out" />
</route>
```

クエリー操作 Hazelcast は、分散マップをクエリーする構文のような SQL を提供します。

```
String q1 = "bar > 1000";
template.sendBodyAndHeader("direct:query", null, HazelcastConstants.QUERY, q1);
```

126.3. MAP CACHE CONSUMER - FROM("HAZELCAST-MAP:FOO")

Hazelcast は、データ収集でイベントリスナーを提供します。キャッシュが操作される際に通知する場合には、マップコンシューマーを使用できます。4 つのイベントがあります。、update、delete、および envict です。イベントタイプは、hazelcast.listener.action" ヘッダー変数に保存されます。マップコンシューマーは、これらの変数内にいくつかの追加情報を提供します。

レスポンスメッセージ内のヘッダー変数：

Name	タイプ	説明
CamelHazelcastListenerTime	**Long**	イベントの時間（ミリ秒単位）
CamelHazelcastListenerType	**文字列**	マップコンシューマーは「cachelistener」に設定します。
CamelHazelcastListenerAction	**文字列**	イベントのタイプ： ここでは、が **追加** され、**更新** され、envicted、および removed が削除されます。
CamelHazelcastObjectId	**文字列**	オブジェクトの oid
CamelHazelcastCacheName	**文字列**	キャッシュの名前 - 例：「foo」
CamelHazelcastCacheType	**文字列**	キャッシュのタイプ（ここではマップ）

オブジェクト値は、メッセージボディー内で put および update アクション内に保存されます。

以下に例を示します。

```
fromF("hazelcast-%sfoo", HazelcastConstants.MAP_PREFIX)
.log("object...")
.choice()

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDED))
        .log("...added")
```

```
        .to("mock:added")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ENVIC
TED))
        .log("...envicted")
        .to("mock:envicted")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.UPDAT
ED))
        .log("...updated")
        .to("mock:updated")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.REMO
VED))
        .log("...removed")
        .to("mock:removed")
    .otherwise()
        .log("fail!");
```

第127章 HAZELCAST MULTIMAP コンポーネント

Camel バージョン 2.7 で利用可能

Hazelcast Multimap コンポーネントは、Hazelcast 分散マルチマップにアクセスできる Camel Hazelcast コンポーネントの 1 つです。

127.1. オプション

Hazelcast Multimap コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Multimap エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-multimap:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

127.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

127.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
defaultOperation (common)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	キューコンシューマーのポーリングタイムアウトをポーリングモードで定義	10000	Long
poolSize (consumer)	キューコンシューマーエグゼキューターのプールサイズの定義	1	int
queueConsumerMode (consumer)	キューコンシューマーモードの定義： Listen または Poll	listen	HazelcastQueueConsumer Mode
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

127.2. MULTIMAP CACHE PRODUCER - TO("HAZELCAST-MULTIMAP:FOO")

マルチマップは、n の値を 1 つのキーに保存できるキャッシュです。multimap プロデューサーは、4 つの操作（put、get、removevalue、delete）を提供します。

リクエストメッセージのヘッダー変数：

Name	タイプ	説明
CamelHazelcastOperationType	**文字列**	有効な値は put、get、removevalue、delete **From Camel 2.16:** clear です。
CamelHazelcastObjectId	**文字列**	キャッシュ内にオブジェクト ID を保存/検索

127.2.1. 配置 の例：

Java DSL の場合

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUT))
.to(String.format("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX));
```

Spring DSL の場合 :

```
<route>
   <from uri="direct:put" />
   <log message="put.."/>
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
   <setHeader headerName="hazelcast.operation.type">
      <constant>put</constant>
   </setHeader>
   <to uri="hazelcast-multimap:foo" />
</route>
```

127.2.2. removevalue の例 :

Java DSL の場合

```
from("direct:removevalue")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.REMOVE_VALUE))
.toF("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX);
```

Spring DSL の場合 :

```
<route>
   <from uri="direct:removevalue" />
   <log message="removevalue..."/>
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
   <setHeader headerName="hazelcast.operation.type">
      <constant>removevalue</constant>
   </setHeader>
   <to uri="hazelcast-multimap:foo" />
</route>
```

値を削除するには、メッセージボディー内で削除する値を指定する必要があります。マルチマップオブジェクト \{key: "4711" values: { "my-foo", "my-bar" }} がある場合は、メッセージのボディーに "my-foo" の値を追加して "my-foo" の値を削除する必要があります。

127.2.3. get の場合の例 :

Java DSL の場合

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.GET))
.toF("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX)
.to("seda:out");
```

Spring DSL の場合 :

```
<route>
   <from uri="direct:get" />
   <log message="get.."/>
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
   <setHeader headerName="hazelcast.operation.type">
      <constant>get</constant>
   </setHeader>
   <to uri="hazelcast-multimap:foo" />
   <to uri="seda:out" />
</route>
```

127.2.4. 削除 用のサンプル :

Java DSL の場合

```
from("direct:delete")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.DELETE))
.toF("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX);
```

Spring DSL の場合 :

```
<route>
   <from uri="direct:delete" />
   <log message="delete.."/>
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
   <setHeader headerName="hazelcast.operation.type">
      <constant>delete</constant>
   </setHeader>
   <to uri="hazelcast-multimap:foo" />
</route>
```

以下を使用して、テストクラスで呼び出すことができます。

```
template.sendBodyAndHeader("direct:[put|get|removevalue|delete]", "my-foo",
HazelcastConstants.OBJECT_ID, "4711");
```

127.3. MULTIMAP CACHE CONSUMER - FROM("HAZELCAST-MULTIMAP:FOO")

マルチマップキャッシュでは、このコンポーネントはマップキャッシュコンシューマーと同じリスナー/変数を提供します（更新リスナーおよび enviction リスナーを除く）。唯一の違いは、URI 内のマルチマップ 接頭辞です。以下に例を示します。

```
fromF("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX)
.log("object...")
.choice()

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDED))
        .log("...added")
                .to("mock:added")

//.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ENVICTED))
        //      .log("...envicted")
        //      .to("mock:envicted")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.REMOVED))
                .log("...removed")
                .to("mock:removed")
        .otherwise()
                .log("fail!");
```

レスポンスメッセージ内のヘッダー変数：

Name	タイプ	説明
CamelHazelcastListenerTime	**Long**	イベントの時間（ミリ秒単位）
CamelHazelcastListenerType	**文字列**	マップコンシューマーは「cachelistener」に設定します。

Name	タイプ	説明
CamelHazelcastListenerAction	**文字列**	イベントの型 - ここで **追加** / **削除** （ストーティング **されたばかり**）
CamelHazelcastObjectId	**文字列**	オブジェクトの oid
CamelHazelcastCacheName	**文字列**	キャッシュの名前 - 例：「foo」
CamelHazelcastCacheType	**文字列**	キャッシュのタイプ（ここではマルチマップ）

第128章 HAZELCAST QUEUE コンポーネント

Camel バージョン 2.7 で利用可能

Hazelcast Queue コンポーネントは、Hazelcast 分散キューへのアクセスを可能にする Camel Hazelcast コンポーネントの 1 つです。

128.1. オプション

Hazelcast Queue コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Queue エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-queue:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

128.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

128.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
defaultOperation (common)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	キューコンシューマーのポーリングタイムアウトをポーリングモードで定義	10000	Long
poolSize (consumer)	キューコンシューマーエグゼキューターのプールサイズの定義	1	int
queueConsumerMode (consumer)	キューコンシューマーモードの定義： Listen または Poll	listen	HazelcastQueueConsumer Mode
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

128.2. QUEUE PRODUCER - TO("HAZELCAST-QUEUE:FOO")

キュープロデューサーは 10 操作を提供します。 add * put * poll * peek * offer * remove value * remaining capacity * remove all * remove if * drain to * take all * keep all*

128.2.1. 追加 の例 :

```
from("direct:add")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.ADD))
.toF("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.2. 配置 の例 :

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUT))
.toF("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.3. ポーリング の例 :

```
from("direct:poll")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.POLL))
.toF("hazelcast:%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.4. peek のサンプル：

```
from("direct:peek")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PEEK))
.toF("hazelcast:%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.5. 提供 するサンプル：

```
from("direct:offer")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.OFFER))
.toF("hazelcast:%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.6. removevalue の例：

```
from("direct:removevalue")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.REMOVE_VALUE))
.toF("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX);
```

128.2.7. 残りの容量 の例：

```
from("direct:remaining-capacity").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.REMAINING_CAPACITY)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.2.8. すべての削除 のサンプル：

```
from("direct:removeAll").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.REMOVE_ALL)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.2.9. 以下の 場合の削除 の例：

```
from("direct:removeIf").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.REMOVE_IF)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.2.10. ドレイン（解放）の 例：

```
from("direct:drainTo").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.DRAIN_TO)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.2.11. take の例：

```
from("direct:take").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.TAKE)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.2.12. すべてのサンプルを保持 します。

```
from("direct:retainAll").setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.RETAIN_ALL)).to(
String.format("hazelcast-%sbar", HazelcastConstants.QUEUE_PREFIX));
```

128.3. QUEUE CONSUMER - FROM("HAZELCAST-QUEUE:FOO")

キューコンシューマーは、2 つの異なるモードを提供します。

- Poll
- listen

ポーリング モードの例

```
fromF("hazelcast-%sfoo?queueConsumerMode=Poll",
HazelcastConstants.QUEUE_PREFIX)).to("mock:result");
```

これにより、コンシューマーはキューをポーリングし、タイムアウト後にキューの先頭または null を返します。

代わりに Listen モードでは、コンシューマーがキューのイベントをリッスンします。

Listen モードのキューコンシューマーは 2 つの操作を提供します。* add * remove

Listen モードの例

```
fromF("hazelcast-%smm", HazelcastConstants.QUEUE_PREFIX)
  .log("object...")
  .choice()

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDE
D))
        .log("...added")
      .to("mock:added")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.REMO
```

```
VED))
        .log("...removed")
        .to("mock:removed")
    .otherwise()
        .log("fail!");
```

第129章 HAZELCAST REPLICATED MAP コンポーネント

Camel バージョン 2.16 から利用可能

Hazelcast インスタンスコンポーネントは Camel Hazelcast Components の 1 つであり、クラスター内のキャッシュインスタンスの結合/ターリーブイベントを使用できます。レプリケートされたマップは、データパーティションのない、セキュアに一貫性のある分散キーと値のデータ構造です。

129.1. オプション

Hazelcast Replicated Map コンポーネントは以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Replicated Map エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-replicatedmap:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

129.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

129.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
defaultOperation (common)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	キューコンシューマーのポーリングタイムアウトをポーリングモードで定義	10000	Long
poolSize (consumer)	キューコンシューマーエグゼキューターのプールサイズの定義	1	int
queueConsumerMode (consumer)	キューコンシューマーモードの定義： Listen または Poll	listen	HazelcastQueueConsumer Mode
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

129.2. REPLICATEDMAP キャッシュプロデューサー

replicatedmap プロデューサーは 4 つの操作を提供します。* put * get * delete * clear

リクエストメッセージのヘッダー変数：

Name	タイプ	説明
CamelHazelcastOperationType	**文字列**	有効な値は put、get、removevalue、delete です。

Name	タイプ	説明
CamelHazelcastObjectId	**文字列**	キャッシュ内にオブジェクト ID を保存/検索

129.2.1. 配置 の例 :

Java DSL の場合

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUT))
.to(String.format("hazelcast-%sbar", HazelcastConstants.REPLICATEDMAP_PREFIX));
```

Spring DSL の場合 :

```
<route>
   <from uri="direct:put" />
   <log message="put.."/>
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
   <setHeader headerName="hazelcast.operation.type">
      <constant>put</constant>
   </setHeader>
   <to uri="hazelcast-replicatedmap:foo" />
</route>
```

129.2.2. get の場合の例 :

Java DSL の場合

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.GET))
.toF("hazelcast-%sbar", HazelcastConstants.REPLICATEDMAP_PREFIX)
.to("seda:out");
```

Spring DSL の場合 :

```
<route>
   <from uri="direct:get" />
   <log message="get.."/>
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
```

```
>
   <setHeader headerName="hazelcast.operation.type">
      <constant>get</constant>
   </setHeader>
   <to uri="hazelcast-replicatedmap:foo" />
   <to uri="seda:out" />
</route>
```

129.2.3. 削除 用のサンプル :

Java DSL の場合

```
from("direct:delete")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.DELETE))
.toF("hazelcast-%sbar", HazelcastConstants.REPLICATEDMAP_PREFIX);
```

Spring DSL の場合 :

```
<route>
   <from uri="direct:delete" />
   <log message="delete.."/>
      <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
   <setHeader headerName="hazelcast.operation.type">
      <constant>delete</constant>
   </setHeader>
   <to uri="hazelcast-replicatedmap:foo" />
</route>
```

以下を使用して、テストクラスで呼び出すことができます。

```
template.sendBodyAndHeader("direct:[put|get|delete|clear]", "my-foo",
HazelcastConstants.OBJECT_ID, "4711");
```

129.3. REPLICATEDMAP キャッシュコンシューマー

マルチマップキャッシュでは、このコンポーネントはマップキャッシュコンシューマーと同じリスナー/変数を提供します（更新リスナーおよび enviction リスナーを除く）。唯一の違いは、URI 内の マルチマップ 接頭辞です。以下に例を示します。

```
fromF("hazelcast-%sbar", HazelcastConstants.MULTIMAP_PREFIX)
.log("object...")
.choice()

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ADDE
```

```
D))
        .log("...added")
                .to("mock:added")

//.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.ENVI
CTED))
        //      .log("...envicted")
        //      .to("mock:envicted")

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.REMO
VED))
                .log("...removed")
                .to("mock:removed")
        .otherwise()
                .log("fail!");
```

レスポンスメッセージ内のヘッダー変数：

Name	タイプ	説明
CamelHazelcastListenerTime	**Long**	イベントの時間（ミリ秒単位）
CamelHazelcastListenerType	**文字列**	マップコンシューマーは「cachelistener」に設定します。
CamelHazelcastListenerAction	**文字列**	イベントの型 - ここで **追加 / 削除** （ストーティング **されたばかり**）
CamelHazelcastObjectId	**文字列**	オブジェクトの oid
CamelHazelcastCacheName	**文字列**	キャッシュの名前 - 例：「foo」

Name	タイプ	説明
CamelHazelcastCacheType	**文字列**	キャッシュのタイプ（ここでは replicatedmap）

第130章 HAZELCAST RINGBUFFER コンポーネント

Camel バージョン 2.16 から利用可能

Camel 2.16 の影響

Hazelcast リングバッファーコンポーネントは Camel Hazelcast Components の 1 つで、Hazelcast リングバッファーにアクセスできます。Ringbuffer は、データがリングのような構造に保存される分散データ構造です。これは、特定の容量を持つ円形の配列と考えることができます。

130.1. オプション

Hazelcast Ringbuffer コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Ringbuffer エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-ringbuffer:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

130.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

130.1.2. クエリーパラメーター（10 パラメーター）：

Name	説明	デフォルト	Type
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
defaultOperation (producer)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (producer)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (producer)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

130.2. リングバッファーキャッシュ プロデューサー

リングバッファープロデューサーは 5 つの操作を提供します。* readonceHead * readonceTail * remainingCapacity * capacity

リクエストメッセージのヘッダー変数：

Name	タイプ	説明
CamelHazelcastOperationType	**文字列**	有効な値は put、get、removevalue、delete です。
CamelHazelcastObjectId	**文字列**	キャッシュ内にオブジェクト ID を保存/検索

130.2.1. 配置 の例：

Java DSL の場合

```
from("direct:put")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.ADD))
.to(String.format("hazelcast-%sbar", HazelcastConstants.RINGBUFFER_PREFIX));
```

Spring DSL の場合：

```
<route>
    <from uri="direct:put" />
    <log message="put.."/>
        <!-- If using version 2.8 and above set headerName to "CamelHazelcastOperationType" --
>
    <setHeader headerName="hazelcast.operation.type">
        <constant>add</constant>
    </setHeader>
    <to uri="hazelcast-ringbuffer:foo" />
</route>
```

130.2.2. ヘッドから readonce のサンプル：

Java DSL の場合

```
from("direct:get")
.setHeader(HazelcastConstants.OPERATION,
constant(HazelcastOperation.READ_ONCE_HEAD))
.toF("hazelcast-%sbar", HazelcastConstants.RINGBUFFER_PREFIX)
.to("seda:out");
```

第131章 HAZELCAST SEDA コンポーネント

Camel バージョン 2.7 で利用可能

Hazelcast SEDA コンポーネントは Camel Hazelcast BlockingQueue にアクセスできる Camel Hazelcast コンポーネントの 1 つです。SEDA コンポーネントは、提供される残りのコンポーネントとは異なります。コア「SEDA」コンポーネントと同様に、非同期 SEDA アーキテクチャーをサポートするために work-queue を実装します。

131.1. オプション

Hazelcast SEDA コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast SEDA エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-seda:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

131.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

131.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
defaultOperation (common)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	キューコンシューマーのポーリングタイムアウトをポーリングモードで定義	10000	Long
poolSize (consumer)	キューコンシューマーエグゼキューターのプールサイズの定義	1	int
queueConsumerMode (consumer)	キューコンシューマーモードの定義： Listen または Poll	listen	HazelcastQueueConsumer Mode
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

131.2. SEDA PRODUCER - TO("HAZELCAST-SEDA:FOO")

SEDA プロデューサーは操作を提供しません。指定のキューにデータのみを送信します。

Java DSL の場合

```
from("direct:foo")
.to("hazelcast-seda:foo");
```

Spring DSL の場合：

```
<route>
  <from uri="direct:start" />
```

```
    <to uri="hazelcast-seda:foo" />
  </route>
```

131.3. SEDA コンシューマー - FROM("HAZELCAST-SEDA:FOO")

SEDA コンシューマーは操作を提供しません。指定のキューからのみデータを取得します。

Java DSL の場合

```
from("hazelcast-seda:foo")
.to("mock:result");
```

Spring DSL の場合 :

```
<route>
  <from uri="hazelcast-seda:foo" />
  <to uri="mock:result" />
</route>
```

第132章 HAZELCAST SET COMPONENT

Camel バージョン 2.7 で利用可能

Hazelcast Set コンポーネントは Camel Hazelcast コンポーネントの 1 つで、Hazelcast 分散セットにアクセスできます。

132.1. オプション

Hazelcast Set コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Set エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-set:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

132.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

132.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
defaultOperation (common)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	キューコンシューマーのポーリングタイムアウトをポーリングモードで定義	10000	Long
poolSize (consumer)	キューコンシューマーエグゼキューターのプールサイズの定義	1	int
queueConsumerMode (consumer)	キューコンシューマーモードの定義： Listen または Poll	listen	HazelcastQueueConsumer Mode
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

第133章 HAZELCAST TOPIC コンポーネント

Camel バージョン 2.15 から利用可能

Hazelcast Topic コンポーネントは Camel Hazelcast コンポーネントの 1 つで、Hazelcast 分散トピックにアクセスできます。

133.1. オプション

Hazelcast Topic コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
hazelcastInstance (advanced)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		HazelcastInstance
hazelcastMode (advanced)	使用すべきインスタンスの種類を示す hazelcast モード参照。モードを指定しない場合、ノードモードがデフォルトになります。	node	文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Hazelcast Topic エンドポイントは、URI 構文を使用して設定します。

```
hazelcast-topic:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

133.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

133.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
defaultOperation (common)	使用するデフォルトの操作を指定します（操作ヘッダーが指定されていない場合）。		HazelcastOperation
hazelcastInstance (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照。		HazelcastInstance
hazelcastInstanceName (common)	hazelcast エンドポイントに使用できる hazelcast インスタンス参照名。インスタンスの参照を指定しない場合、Camel は camel-hazelcast インスタンスのデフォルトの hazelcast インスタンスを使用します。		文字列
reliable (common)	エンドポイントが信頼できる Topic 構造を使用するかどうかを定義します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollingTimeout (consumer)	キューコンシューマーのポーリングタイムアウトをポーリングモードで定義	10000	Long
poolSize (consumer)	キューコンシューマーエグゼキューターのプールサイズの定義	1	int
queueConsumerMode (consumer)	キューコンシューマーモードの定義： Listen または Poll	listen	HazelcastQueueConsumer Mode
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
concurrentConsumers (seda)	SEDA キューからポーリングされる同時コンシューマーを使用するには、以下を行います。	1	int
onErrorDelay (seda)	エラー発生後にコンシューマーがポーリングを継続する前にミリ秒単位です。	1000	int
pollTimeout (seda)	SEDA キューから消費する際に使用するタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
トランザクション処理 (seda)	true に設定すると、コンシューマーはトランザクションモードで実行され、seda キューのメッセージはトランザクションがコミットされた場合にのみ削除されます。これは処理の完了時に行われます。	false	boolean
transferExchange (seda)	true に設定すると、Exchange 全体が転送されます。ヘッダーまたはボディーにシリアライズ可能なオブジェクトが含まれていない場合、それらは省略されます。	false	boolean

133.2. TOPIC PRODUCER - TO("HAZELCAST-TOPIC:FOO")

トピックプロデューサーは 1 つの操作（パブリッシュ）のみを提供します。

133.2.1. 公開 の例 :

```
from("direct:add")
.setHeader(HazelcastConstants.OPERATION, constant(HazelcastOperation.PUBLISH))
.toF("hazelcast-%sbar", HazelcastConstants.PUBLISH_OPERATION);
```

133.3. TOPIC CONSUMER - FROM("HAZELCAST-TOPIC:FOO")

トピックコンシューマーは 1 つの操作（受信）のみを提供します。このコンポーネントは、トピックの発生時に想定どおりに複数の消費をサポートすることが意図されているため、同じハザーキャストトピックに必要なだけのコンシューマーを自由に使用できます。

```
fromF("hazelcast-%sfoo", HazelcastConstants.TOPIC_PREFIX)
  .choice()

.when(header(HazelcastConstants.LISTENER_ACTION).isEqualTo(HazelcastConstants.RECEIVED))
```

```
    .log("...message received")
  .otherwise()
    .log("...this should never have happened")
```

第134章 HBASE コンポーネント

Camel バージョン 2.10 で利用可能

このコンポーネントは、Apache HBase にべき等リポジトリー、プロデューサー、およびコンシューマーを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hbase</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

134.1. APACHE HBASE OVERVIEW

HBase は、Google の Bigtable の後にモデル化されたオープンソースの分散型バージョン化された列指向ストアです(A Distributed Storage System for Structured Data)。Big Data への無作為な読み取り/書き込みアクセスが必要な場合には、HBase を使用できます。詳細は「Apache HBase」を参照してください。

134.2. CAMEL および HBASE

camel ルート内で datasotre を使用する場合、Camel メッセージがデータストアに保存される方法を指定するchalenge が常にあります。ドキュメントベースのストアでは、メッセージボディーをドキュメントに直接マッピングできるため、より簡単に使用できます。リレーショナルデータベースでは、ORM ソリューションを使用してプロパティーを列にマッピングできます。列ベースのストアでは、このようなマッピングを実行するための標準的な方法がないため、作業がより困難になります。

HBase は、さらに 2 つの課題を追加します。

- HBase は列をファミリーにグループ化するため、命名規則を使用してプロパティーを列にマッピングするだけでは不十分です。
- HBase には型の概念がありません。これは、すべてを byte[] として保存し、byte[] が String、Number、シリアライズされた Java オブジェクト、またはバイナリーデータのみを表

すかどうかを認識しないことを意味します。

このような課題に対応するために、camel-hbase はメッセージヘッダーを使用して、メッセージの HBase 列へのマッピングを指定します。また、HBase データをモデル化し、XML/json との間で簡単に変換できる camel-hbase 提供クラスを使用する機能も提供します。
最後に、ユーザーは独自のマッピングストラテジーを実装し、使用することができます。

マッピングストラテジー camel-hbase に関係なく、メッセージを org.apache.camel.component.hbase.model.HBaseData オブジェクトに変換し、そのオブジェクトを内部操作に使用します。

134.3. コンポーネントの設定

HBase コンポーネントは、カスタムの HBaseConfiguration オブジェクトをプロパティーとして指定するか、クラスパスにある HBase 関連のリソースに基づいて独自に HBase 設定オブジェクトを作成できます。

```
<bean id="hbase" class="org.apache.camel.component.hbase.HBaseComponent">
    <property name="configuration" ref="config"/>
</bean>
```

コンポーネントに設定オブジェクトが指定されていない場合、コンポーネントはこれを作成します。作成された設定は hbase-site.xml ファイルのクラスパスを検索します。ここから設定が取り消されます。HBase クライアントの設定方法は、「HBase クライアント設定および依存関係」を参照してください。

134.4. HBASE プロデューサー

上記のように、camel は HBase の produers エンドポイントを提供します。これにより、Camel ルートを使用して HBase からデータを保存、削除、取得、クエリーすることができます。

```
hbase://table[?options]
```

table はテーブル名に置き換えます。

サポートされる操作は以下のとおりです。

- *put*
- *Get*
- *Delete*
- *スキャン*

134.4.1. サポートされる URI オプション

HBase コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有設定の使用		設定
poolMaxSize (common)	HTable プールの各テーブルに対して保持する参照の最大数。デフォルト値は 10 です。	10	int
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

HBase エンドポイントは、URI 構文を使用して設定します。

```
hbase:tableName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

134.4.2. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
tableName	**必要な** テーブルの名前		文字列

134.4.3. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
cellMappingStrategyFactory (common)	セルをマッピングするカスタムの CellMappingStrategyFactory を使用するには、以下を実行します。		CellMappingStrategy Factory
フィルター （共通）	使用するフィルターの一覧。		リスト
mappingStrategyClassName (common)	カスタムマッピングストラテジー実装のクラス名。		文字列
mappingStrategyName (common)	Camel メッセージの HBase 列へのマッピングに使用するストラテジー。サポートされる値： header または body。		文字列
rowMapping (common)	マップから HBaseRow にキー/値をマッピングします。rowId - 行の ID キーがサポートされます。これは、通常 Exchange ごとの行への変更として使用が制限されています。rowType - 行 ID をカバーするタイプ。サポートされる操作： CamelHBaseScan. family: 列ファミリー。複数の列を参照するための数字接尾辞をサポートします。修飾子 - 列修飾子。複数の列を参照する数字の接尾辞をサポートします。値 - 値複数の列の valueType を参照する数字接尾辞をサポートします（値タイプ）。複数の列を参照する数字の接尾辞をサポートします。サポートされる操作： CamelHBaseGet および CamelHBaseScan		マップ
rowModel (common)	各行のモデル化方法を記述する org.apache.camel.component.hbase.model.HBaseRow のインスタンス		HBaseRow
userGroupInformation (common)	Kerberos を使用する HBase と通信する権限を定義します。		UserGroupInformation

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
maxMessagesPerPoll (consumer)	各ポーリングのポーリング制限としてメッセージの最大数を取得します。デフォルトは無制限ですが、0 または負の値を使用して無制限として無効にします。		int
操作 （コンシューマー）	実行する HBase 操作		文字列
Remove (consumer)	オプションが true の場合、Camel HBase Consumer は処理する行を削除します。	true	boolean
removeHandler (consumer)	行を削除する際に実行されるカスタムの HBaseRemoveHandler を使用するには、以下を行います。		HBaseRemoveHandler
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
maxResults （プロデューサー）	スキャンする行の最大数。	100	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

134.4.4. 操作を行います。

HBase は列ベースのストアで、特定行の特定の列にデータを保存できます。列はファミリーにグループ化されるため、列ファミリーとその列の修飾子を指定する必要があります。データを特定の列に

保存するには、列と行の両方を指定する必要があります。

camel ルートから HBase にデータを保存する最も簡単なシナリオは、メッセージボディーの一部を指定された HBase 列に保存することです。

```
        <route>
            <from uri="direct:in"/>
            <!-- Set the HBase Row -->
            <setHeader headerName="CamelHBaseRowId">
                <el>${in.body.id}</el>
            </setHeader>
            <!-- Set the HBase Value -->
            <setHeader headerName="CamelHBaseValue">
                <el>${in.body.value}</el>
            </setHeader>
            <to uri="hbase:mytable?
operation=CamelHBasePut&amp;family=myfamily&amp;qualifier=myqualifier"/>
        </route>
```

上記のルートは、メッセージボディーに id および value プロパティーを持つオブジェクトが含まれ、値の内容を id で指定された行の HBase 列 myfamily:myqualifier に保存することを前提としています。複数の列/値のペアを指定する必要がある場合は、追加の列マッピングを指定することもできます。2 番目のヘッダー以降は、RowId2、RowId3、RowId4 などの数字を使用する必要があります。1st ヘッダーのみの番号は 1 ではありません。

```
        <route>
            <from uri="direct:in"/>
            <!-- Set the HBase Row 1st column -->
            <setHeader headerName="CamelHBaseRowId">
                <el>${in.body.id}</el>
            </setHeader>
            <!-- Set the HBase Row 2nd column -->
            <setHeader headerName="CamelHBaseRowId2">
                <el>${in.body.id}</el>
            </setHeader>
            <!-- Set the HBase Value for 1st column -->
            <setHeader headerName="CamelHBaseValue">
                <el>${in.body.value}</el>
            </setHeader>
            <!-- Set the HBase Value for 2nd column -->
            <setHeader headerName="CamelHBaseValue2">
                <el>${in.body.othervalue}</el>
            </setHeader>
            <to uri="hbase:mytable?
operation=CamelHBasePut&amp;family=myfamily&amp;qualifier=myqualifier&amp;family2=myfamily&a
p;qualifier2=myqualifier2"/>
        </route>
```

uri オプション、メッセージヘッダー、またはその両方の組み合わせを使用することができることに

注意してください。定数を uri の一部として指定し、動的な値をヘッダーとして指定することが推奨されます。何かがヘッダーとして定義され、URI の一部として定義されている場合は、ヘッダーが使用されます。

134.4.5. 操作を取得します。

Get Operation は、指定された HBase 行から 1 つ以上の値を取得するために使用される操作です。取得する値を指定するには、URI の一部として指定するか、またはメッセージヘッダーとして指定することができます。

```
<route>
    <from uri="direct:in"/>
    <!-- Set the HBase Row of the Get -->
    <setHeader headerName="CamelHBaseRowId">
        <el>${in.body.id}</el>
    </setHeader>
    <to uri="hbase:mytable?
operation=CamelHBaseGet&amp;family=myfamily&amp;qualifier=myqualifier&amp;valueType=java.lang
Long"/>
    <to uri="log:out"/>
</route>
```

上記の例では、get 操作の結果は CamelHBaseValue という名前のヘッダーとして保存されます。

134.4.6. 操作を削除します。

camel-hbase を使用して HBase delete 操作を実行することもできます。delete 操作は行全体を削除します。指定する必要があるのは、メッセージヘッダーの一部として 1 つまたは複数の行になります。

```
<route>
    <from uri="direct:in"/>
    <!-- Set the HBase Row of the Get -->
    <setHeader headerName="CamelHBaseRowId">
        <el>${in.body.id}</el>
    </setHeader>
    <to uri="hbase:mytable?operation=CamelHBaseDelete"/>
</route>
```

134.4.7. スキャン操作。

スキャン操作は HBase のクエリーと同じです。scan 操作を使用して、複数の行を取得できます。結果の一部となる列を指定し、値をオブジェクトに変換する方法を指定するには、URI オプションまたはヘッダーを使用します。

-

```
        <route>
            <from uri="direct:in"/>
            <to uri="hbase:mytable?
operation=CamelHBaseScan&amp;family=myfamily&amp;qualifier=myqualifier&amp;valueType=java.la
g.Long&amp;rowType=java.lang.String"/>
            <to uri="log:out"/>
        </route>
```

この場合、結果を制限するためのフィルターの一覧も指定する必要があります。フィルターのリストを uri の一部として指定でき、Camel は すべて のフィルターを満たす行のみを返します。
メッセージに含まれる情報を認識するフィルターを設定するために、camel は ModelAwareFilter を定義します。これにより、フィルターはメッセージとマッピングストラテジーで定義されるモデルを考慮に入れることができます。
ModelAwareFilter camel-hbase を使用すると、選択したマッピングストラテジーが in メッセージに適用されます。これは、マッピングをモデル化するオブジェクトを作成し、そのオブジェクトを Filter に渡します。

たとえば、基準としてメッセージヘッダーとして使用してスキャンを実行するには、以下のように ModelAwareColumnMatchingFilter を使用できます。

```
        <route>
            <from uri="direct:scan"/>
            <!-- Set the Criteria -->
            <setHeader headerName="CamelHBaseFamily">
                <constant>name</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseQualifier">
                <constant>first</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseValue">
                <el>in.body.firstName</el>
            </setHeader>
            <setHeader headerName="CamelHBaseFamily2">
                <constant>name</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseQualifier2">
                <constant>last</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseValue2">
                <el>in.body.lastName</el>
            </setHeader>
            <!-- Set additional fields that you want to be return by skipping value -->
            <setHeader headerName="CamelHBaseFamily3">
                <constant>address</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseQualifier3">
                <constant>country</constant>
            </setHeader>
            <to uri="hbase:mytable?operation=CamelHBaseScan&amp;filters=#myFilterList"/>
        </route>

        <bean id="myFilters" class="java.util.ArrayList">
```

```
            <constructor-arg>
              <list>
                <bean
class="org.apache.camel.component.hbase.filters.ModelAwareColumnMatchingFilter"/>
              </list>
            </constructor-arg>
          </bean>
```

上記のルートは、pojo に properties firstName と lastName がメッセージボディーとして渡されることを想定し、これらのプロパティーを取得し、メッセージヘッダーの一部として追加します。デフォルトのマッピングストラテジーは、ヘッダーを HBase 列にマッピングし、そのモデルを ModelAwareColumnMatchingFilter に渡すモデルオブジェクトを作成します。このフィルターは、モデルに一致する列を含まない行を除外します。これは、たとえばクエリーと同様です。

134.5. HBASE CONSUMER

Camel HBase Consumer は、指定された HBase テーブルで繰り返しスキャンを実行し、メッセージの一部としてスキャン結果を返します。ヘッダーマッピング（デフォルト）またはボディーのマッピングのいずれかを指定できます。その後、メッセージボディーの一部として org.apache.camel.component.hbase.model.HBaseData を追加します。

```
hbase://table[?options]
```

返す列とそのタイプを uri オプションの一部として指定できます。

```
hbase:mutable?
family=name&qualifer=first&valueType=java.lang.String&family=address&qualifer=number&v
alueType2=java.lang.Integer&rowType=java.lang.Long
```

上記の例では、指定されたフィールドで構成されるモデルオブジェクトが作成され、スキャンの結果によりモデルオブジェクトに値が入力されます。最後に、マッピングストラテジーを使用して、このモデルを Camel メッセージにマッピングします。

134.6. HBASE IDEMPOTENT リポジトリー

camel-hbase コンポーネントは、各メッセージが一度だけ処理されるときに使用できるべき等リポジトリーも提供します。HBase idempotent リポジトリーは、テーブル、列ファミリー、および列修飾子で設定され、メッセージごとにそのテーブルを作成します。

```
HBaseConfiguration configuration = HBaseConfiguration.create();
HBaseIdempotentRepository repository = new HBaseIdempotentRepository(configuration,
tableName, family, qualifier);
```

```
from("direct:in")
  .idempotentConsumer(header("messageId"), repository)
  .to("log:out);
```

134.7. HBASE MAPPING

前述のように、デフォルトのマッピングストラテジーは ヘッダー と ボディー のマッピングです。以下に、各マッピングストラテジーがどのように機能するかの詳細例を示します。

134.7.1. HBase ヘッダーマッピングの例

ヘッダーマッピングはデフォルトのマッピングです。"myvalue" の値を HBase row "myrow" と列 "myfamily:mycolum" に配置するには、メッセージに以下のヘッダーを含める必要があります。

ヘッダー	値
Camel HBase RowId	myrow
Camel HBase Family	myfamily
Camel HBase Qualifier	myqualifier
Camel HBase Value	myvalue

異なる列や / または異なる行にさらに値を置くには、ヘッダーのインデックスで接尾辞が付けられた追加のヘッダーを指定します。以下に例を示します。

ヘッダー	値
Camel HBase RowId	myrow

ヘッダー	値
Camel HBase Family	myfamily
Camel HBase Qualifier	myqualifier
Camel HBase Value	myvalue
Camel HBase RowId2	myrow2
Camel HBase Family2	myfamily
Camel HBase Qualifier2	myqualifier
Camel HBase Value2	myvalue2

get や scan などの取得操作では、データの変換先となる型を各列に指定することもできます。試験の場合：

ヘッダー	値
Camel HBase Family	myfamily

ヘッダー	値
Camel HBase Qualifier	myqualifier
Camel HBase ValueType	Long

すべてのメッセージに対して定数として考慮される boilerplate ヘッダーを回避するために、以下のようにエンドポイント URI の一部として指定することもできます。

134.7.2. ボディーマッピングの例

ボディーマッピングストラテジーを使用するには、以下のように URI の一部としてオプション mappingStrategy を指定する必要があります。

```
hbase:mytable?mappingStrategyName=body
```

ボディーマッピングストラテジーを使用するには、ボディーに org.apache.camel.component.hbase.model.HBaseData のインスタンスを追加する必要があります。t を構築することができます。

```
HBaseData data = new HBaseData();
HBaseRow row = new HBaseRow();
row.setId("myRowId");
HBaseCell cell = new HBaseCell();
cell.setFamily("myfamily");
cell.setQualifier("myqualifier");
cell.setValue("myValue");
row.getCells().add(cell);
data.addRows().add(row);
```

上記のオブジェクトは put 操作でたとえば、id myRowId で行を作成または更新し、myvalue の値を myfamily:myqualifier 列に追加します。
ボディーマッピングストラテジーは、最初に非常にアプリされていない可能性があります。ヘッダーマッピングストラテジーよりも大きな利点は、HBaseData オブジェクトを簡単に xml/json に変換できることです。

134.8. 関連項目

- ***Polling Consumer***

- ***Apache HBase***

第135章 HDFS コンポーネント（非推奨）

Camel バージョン 2.8 から利用可能

hdfs コンポーネントを使用すると、HDFS ファイルシステムからメッセージを読み取り、書き込みできます。HDFS は、Hadoop の中核となる分散ファイルシステムです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hdfs</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

135.1. URI 形式

```
hdfs://hostname[:port][/path][?options]
```

URI には、?option=value&option=value&...の形式でクエリーオプションを追加できます。パスは以下の方法で処理されます。

1. **コンシューマーとしてファイルの場合、ファイルを読み取ります。ディレクトリーを表す場合は、設定されたパターンの条件を満たすパス下のすべてのファイルをスキャンします。そのディレクトリー下のすべてのファイルは同じタイプである必要があります。**

2. **プロデューサーとして、少なくとも 1 つの分割ストラテジーが定義されている場合、パスはディレクトリーとみなされ、そのディレクトリー下では設定済みの UuidGenerator を使用して名前が付けられた分割されたファイルごとに別のファイルを作成します。**

備考

hdfs から通常モードを使用する場合、ファイルはチャンクに分割され、チャンクごとにメッセージを生成します。chunkSize オプションを使用してチャンクのサイズを設定できます。hdfs から読み取り、file コンポーネントを使用して通常のファイルに書き込む場合は、fileMode=Append を使用して各チャンクをまとめて追加できます。

135.2. オプション

HDFS コンポーネントは、以下に挙げる 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
jAASConfiguration (common)	JAAS でセキュリティーに特定の設定を使用します。		設定
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

HDFS エンドポイントは、URI 構文を使用して設定します。

```
hdfs:hostName:port/path
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

135.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
hostname	使用する **必須** の HDFS ホスト		文字列
port	使用する HDFS ポート	8020	int
path	使用するディレクトリーパスが **必要です**。		文字列

135.2.2. クエリーパラメーター（38 パラメーター）：

Name	説明	デフォルト	Type
connectOnStartup (common)	プロデューサー/コンシューマーの開始時に HDFS ファイルシステムに接続するかどうか。false の場合、接続はオンデマンドで作成されます。45 x 20 秒の再配信がハードコーディングされているため、で接続を確立するのに最大 15 分かかることがあります。このオプションを false に設定すると、アプリケーションの起動が可能になり、最大 15 分間ブロックされません。	true	boolean
fileSystemType (common)	代わりに HDFS を使用せず、ローカルの java.io.File を使用しない場合は、LOCAL に設定します。	HDFS	HdfsFileSystemType
fileType (common)	使用するファイルタイプ。詳細は、さまざまなファイルタイプについてのドキュメントを参照してください。	NORMAL_FILE	HdfsFileType
keyType (common)	シーケンスファイルまたはマップファイルの場合のキーのタイプ。	NULL	WritableType
owner (common)	ファイルの所有者は、コンシューマーがファイルを取得するには、この所有者と一致する必要があります。そうでない場合は、ファイルはスキップされます。		文字列
valueType (common)	シーケンスファイルまたはマップファイルの場合のキーのタイプ	BYTES	WritableType
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
遅延 （コンシューマー）	ディレクトリースキャンの間隔（ミリ秒単位）。	1000	Long
initialDelay （コンシューマー）	コンシューマーの場合、ディレクトリーのスキャンを開始するまでの待機時間（ミリ秒単位）。		Long
パターン （コンシューマー）	ディレクトリーのスキャンに使用されるパターン	*	文字列

Name	説明	デフォルト	Type
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
append （プロデューサー）	既存ファイルに追加します。すべての HDFS ファイルシステムが append オプションをサポートしているわけではないことに注意してください。	false	boolean
上書き （プロデューサー）	同じ名前の既存ファイルを上書きするかどうか。	true	boolean
Blocksize （詳細）	HDFS ブロックのサイズ	67108864	Long
bufferSize (advanced)	HDFS によって使用されるバッファーサイズ	4096	int
checkIdleInterval (advanced)	アイドルチェッカーのバックグラウンドタスクを実行する頻度（ミリ秒単位）。このオプションは、Splitter ストラテジーが IDLE の場合にのみ使用されます。	500	int
chunkSize (advanced)	通常ファイルを読み取ると、チャンクごとにメッセージを生成するチャンクに分割されます。	4096	int
compressionCodec (advanced)	使用する圧縮コーデック	DEFAULT	HdfsCompressionCodec

Name	説明	デフォルト	Type
compressionType (advanced)	使用する圧縮タイプ（デフォルトは使用されていない）。	NONE	CompressionType
openedSuffix (advanced)	ファイルを読み取り/書き込み用に開くと、書き込みフェーズでファイルを読み込まないように、この接尾辞が付いたファイルの名前が変更されます。	開く	文字列
readSuffix (advanced)	ファイルが読み取られると、再度読み取られないように、この接尾辞が付いたファイルの名前が変更されます。	read	文字列
レプリケーション （詳細）	HDFS レプリケーションファクター	3	short
splitStrategy (advanced)	現在のバージョンでは、追加モードでファイルを開くことは、非常に信頼性がないために無効にされます。したがって、現時点では新しいファイルの作成のみが可能です。Camel HDFS エンドポイントは、この問題を解決しようとします。スプリットストラテジーオプションが定義されている場合、hdfs パスはディレクトリーとして使用され、設定済みの UuidGenerator を使用してファイルが作成されます。分割条件が満たされるたびに、新しいファイルが作成されます。splitStrategy オプションは、SplitStrategy=ST:value,ST:value... という構文で文字列として定義されます。ここで、splitStrategy=ST:value,ST:value,...		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int

Name	説明	デフォルト	Type
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

135.2.3. KeyType および ValueType

- ***NULL これは、キーまたは値がないことを意味します。***
- ***バイトを作成するための BYTE。java Byte クラスは BYTE にマップされます。***
- ***バイトシーケンスを書き込む BYTES。java ByteBuffer クラスをマッピングします。***
- ***Java 整数を書くための INT***

- *java float を書き込む FLOAT*
- *Java long を記述するための LONG*
- *Java double を書くための DOUBLE*
- *Java 文字列を記述するための TEXT*

BYTES は、その他のすべてで使用されます。たとえば、Camel ではファイルが InputStream として送信されると、int はシーケンスファイルまたはマップファイルでバイトのシーケンスとして記述されます。

135.3. 分割ストラテジー

現在のバージョンでは、追加モードでファイルを開くことは、非常に信頼性がないために無効にされます。したがって、現時点では新しいファイルの作成のみが可能です。Camel HDFS エンドポイントは、以下の方法でこの問題を解決するよう試みます。

- *分割ストラテジーオプションが定義されている場合、hdfs パスはディレクトリーとして使用され、設定済みの UuidGenerator を使用してファイルが作成されます。*
- *分割条件が満たされるたびに、新しいファイルが作成されます。*
splitStrategy オプションは、以下の構文を持つ文字列として定義されます。
*splitStrategy=<ST>:<value>,<ST>:<value>,**

ここで、<ST> は以下のいずれかになります。

- *BYTES は新規ファイルが作成され、書き込まれたバイト数が <value> を超える場合に古いファイルが閉じられます。*
- *MESSAGES は新しいファイルが作成され、書き込まれたメッセージの数が <value> を超える場合に古いメッセージが閉じられます。*
-

IDLE は新しいファイルが作成され、最後の <value> ミリ秒で書き込みが行われない場合に古いファイルが閉じられます。

備考

現在、このストラテジーでは IDLE 値を設定するか、または BYTES/MESSAGES 設定を使用するよう HdfsConstants.HDFS_CLOSE ヘッダーを false に設定する必要があります。それ以外では、ファイルは各メッセージで閉じられます。

以下に例を示します。

```
hdfs://localhost/tmp/simple-file?splitStrategy=IDLE:1000,BYTES:5
```

つまり、新しいファイルが 1 秒以上アイドル状態であったり、5 バイトを超える場合に作成されます。そのため、oop fs -ls /tmp/simple-file を実行すると、複数のファイル が作成されていることを確認できます。

135.4. メッセージヘッダー

このコンポーネントでは、以下のヘッダーがサポートされます。

135.4.1. プロデューサーのみ

ヘッダー	説明
CamelFileName	Camel 2.13: 書き込むファイルの名前（エンドポイントパスに対して相対的）を指定します。名前は **String** または Expression オブジェクトになります。分割ストラテジーを使用しない場合にのみ該当します。

135.5. ファイルストリームを閉じる制御

Camel 2.10.4 から利用可能

分割ストラテジー なし で HDFS プロデューサーを使用する場合、ファイル出力ストリームはデフォルトで書き込み後に閉じられます。ただし、ストリームを開いたままにし、後でのみストリームを明示的に閉じることができます。これを制御するために、ヘッダー HdfsConstants.HDFS_CLOSE (value

= "CamelHdfsClose")を使用できます。この値をブール値に設定すると、ストリームを閉じるかどうかを明示的に制御できます。

これは、ストリームが閉じられたときに制御できるさまざまなストラテジーがあるため、分割ストラテジーを使用する場合は適用されないことに注意してください。

135.6. OSGI でのこのコンポーネントの使用

このコンポーネントは OSGi 環境で完全に機能しますが、ユーザーからいくつかのアクションが必要になります。Hadoop は、リソースをロードするためにスレッドコンテキストクラスローダーを使用します。通常、スレッドコンテキストクラスローダーは、ルートが含まれるバンドルのバンドルクラスローダーになります。そのため、デフォルトの設定ファイルはバンドルクラスローダーから確認する必要があります。これに対応する一般的な方法は、バンドルルートで core-default.xml のコピーを保持することです。このファイルは hadoop-common.jar にあります。

第136章 HDFS2 コンポーネント

Camel バージョン 2.14 から利用可能

hdfs2 コンポーネントを使用すると、Hadoop 2.x を使用して HDFS ファイルシステムからメッセージを読み取り/書き込みできます。HDFS は、Hadoop の中核となる分散ファイルシステムです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hdfs2</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

136.1. URI 形式

```
hdfs2://hostname[:port][/path][?options]
```

URI には、?option=value&option=value&...の形式でクエリーオプションを追加できます。パスは以下の方法で処理されます。

1. コンシューマーとしてファイルの場合、ファイルを読み取ります。ディレクトリーを表す場合は、設定されたパターンの条件を満たすパス下のすべてのファイルをスキャンします。そのディレクトリー下のすべてのファイルは同じタイプである必要があります。

2. プロデューサーとして、少なくとも 1 つの分割ストラテジーが定義されている場合、パスはディレクトリーとみなされ、そのディレクトリー下では設定済みの UuidGenerator を使用して名前が付けられた分割されたファイルごとに別のファイルを作成します。

hdfs2 から通常モードを使用する場合、ファイルはチャンクに分割され、チャンクごとにメッセージを生成します。chunkSize オプションを使用してチャンクのサイズを設定できます。hdfs から読み取り、file コンポーネントを使用して通常のファイルに書き込む場合は、fileMode=Append を使用して各チャンクをまとめて追加できます。

136.2. オプション

HDFS2 コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
jAASConfiguration (common)	JAAS でセキュリティーに特定の設定を使用します。		設定
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

HDFS2 エンドポイントは、URI 構文を使用して設定します。

```
hdfs2:hostName:port/path
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

136.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
hostname	使用する **必須** の HDFS ホスト		文字列
port	使用する HDFS ポート	8020	int
path	使用するディレクトリーパスが **必要です**。		文字列

136.2.2. クエリーパラメーター（38 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
connectOnStartup (common)	プロデューサー/コンシューマーの開始時に HDFS ファイルシステムに接続するかどうか。false の場合、接続はオンデマンドで作成されます。45 x 20 秒の再配信がハードコーディングされているため、で接続を確立するのに最大 15 分かかることがあります。このオプションを false に設定すると、アプリケーションの起動が可能になり、最大 15 分間ブロックされません。	true	boolean
fileSystemType (common)	代わりに HDFS を使用せず、ローカルの java.io.File を使用しない場合は、LOCAL に設定します。	HDFS	HdfsFileSystemType
fileType (common)	使用するファイルタイプ。詳細は、さまざまなファイルタイプについてのドキュメントを参照してください。	NORMAL_FILE	HdfsFileType
keyType (common)	シーケンスファイルまたはマップファイルの場合のキーのタイプ。	NULL	WritableType
owner (common)	ファイルの所有者は、コンシューマーがファイルを取得するには、この所有者と一致する必要があります。そうでない場合は、ファイルはスキップされます。		文字列
valueType (common)	シーケンスファイルまたはマップファイルの場合のキーのタイプ	BYTES	WritableType
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
パターン （コンシューマー）	ディレクトリーのスキャンに使用されるパターン	*	文字列
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
append （プロデューサー）	既存ファイルに追加します。すべての HDFS ファイルシステムが append オプションをサポートしているわけではないことに注意してください。	false	boolean
上書き （プロデューサー）	同じ名前の既存ファイルを上書きするかどうか。	true	boolean
Blocksize （詳細）	HDFS ブロックのサイズ	67108864	Long
bufferSize (advanced)	HDFS によって使用されるバッファーサイズ	4096	int
checkIdleInterval (advanced)	アイドルチェッカーのバックグラウンドタスクを実行する頻度（ミリ秒単位）。このオプションは、Splitter ストラテジーが IDLE の場合にのみ使用されます。	500	int
chunkSize (advanced)	通常ファイルを読み取ると、チャンクごとにメッセージを生成するチャンクに分割されます。	4096	int
compressionCodec (advanced)	使用する圧縮コーデック	DEFAULT	HdfsCompressionCodec
compressionType (advanced)	使用する圧縮タイプ（デフォルトは使用されていない）。	NONE	CompressionType
openedSuffix (advanced)	ファイルを読み取り/書き込み用に開くと、書き込みフェーズでファイルを読み込まないように、この接尾辞が付いたファイルの名前が変更されます。	開く	文字列

Name	説明	デフォルト	Type
readSuffix (advanced)	ファイルが読み取られると、再度読み取られないように、この接尾辞が付いたファイルの名前が変更されます。	read	文字列
レプリケーション （詳細）	HDFS レプリケーションファクター	3	short
splitStrategy (advanced)	現在のバージョンでは、追加モードでファイルを開くことは、非常に信頼性がないために無効にされます。したがって、現時点では新しいファイルの作成のみが可能です。Camel HDFS エンドポイントは、この問題を解決しようとします。スプリットストラテジーオプションが定義されている場合、hdfs パスはディレクトリーとして使用され、設定済みの UuidGenerator を使用してファイルが作成されます。分割条件が満たされるたびに、新しいファイルが作成されます。splitStrategy オプションは、SplitStrategy=ST:value,ST:value... という構文で文字列として定義されます。ここで、splitStrategy=ST:value,ST:value,...		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean

Name	説明	デフォルト	Type
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

136.2.3. KeyType および ValueType

- ***NULL これは、キーまたは値がないことを意味します。***
- ***バイトを作成するための BYTE。java Byte クラスは BYTE にマップされます。***
- ***バイトシーケンスを書き込む BYTES。java ByteBuffer クラスをマッピングします。***
- ***Java 整数を書くための INT***

- *java float を書き込む FLOAT*
- *Java long を記述するための LONG*
- *Java double を書くための DOUBLE*
- *Java 文字列を記述するための TEXT*

BYTES は、その他のすべてで使用されます。たとえば、Camel ではファイルが InputStream として送信されると、int はシーケンスファイルまたはマップファイルでバイトのシーケンスとして記述されます。

136.3. 分割ストラテジー

現在のバージョンでは、追加モードでファイルを開くことは、非常に信頼性がないために無効にされます。したがって、現時点では新しいファイルの作成のみが可能です。Camel HDFS エンドポイントは、以下の方法でこの問題を解決するよう試みます。

- *分割ストラテジーオプションが定義されている場合、hdfs パスはディレクトリーとして使用され、設定済みの UuidGenerator を使用してファイルが作成されます。*
- *分割条件が満たされるたびに、新しいファイルが作成されます。 splitStrategy オプションは、splitStrategy=<ST>:<value>,<ST>:<value>,* 構文で文字列として定義されます。*

ここで、<ST> は以下のいずれかになります。

- *BYTES は新規ファイルが作成され、書き込まれたバイト数が <value> を超える場合に古いファイルが閉じられます。*
- *MESSAGES は新しいファイルが作成され、書き込まれたメッセージの数が <value> を超える場合に古いメッセージが閉じられます。*
-

IDLE は新しいファイルが作成され、最後の <value> ミリ秒で書き込みが行われない場合に古いファイルが閉じられます。

現在、このストラテジーでは IDLE 値を設定するか、または BYTES/MESSAGES 設定を使用するよう HdfsConstants.HDFS_CLOSE ヘッダーを false に設定する必要があります。それ以外では、ファイルは各メッセージで閉じられます。

以下に例を示します。

```
hdfs2://localhost/tmp/simple-file?splitStrategy=IDLE:1000,BYTES:5
```

つまり、新しいファイルが 1 秒以上アイドル状態であったり、5 バイトを超える場合に作成されます。そのため、oop fs -ls /tmp/simple-file を実行すると、複数のファイル が作成されていることを確認できます。

136.4. メッセージヘッダー

このコンポーネントでは、以下のヘッダーがサポートされます。

136.4.1. プロデューサーのみ

ヘッダー	説明
CamelFileName	Camel 2.13: 書き込むファイルの名前（エンドポイントパスに対して相対的）を指定します。名前は **String** または Expression オブジェクトになります。分割ストラテジーを使用しない場合にのみ該当します。

136.5. ファイルストリームを閉じる制御

分割ストラテジー なし で HDFS2 プロデューサーを使用する場合、ファイル出力ストリームはデフォルトで書き込み後に閉じられます。ただし、ストリームを開いたままにし、後でのみストリームを明示的に閉じることができます。これを制御するために、ヘッダー HdfsConstants.HDFS_CLOSE (value = "CamelHdfsClose")を使用できます。この値をブール値に設定すると、ストリームを閉じるかどうかを明示的に制御できます。

これは、ストリームが閉じられたときに制御できるさまざまなストラテジーがあるため、分割ストラテジーを使用する場合は適用されないことに注意してください。

136.6. OSGI でのこのコンポーネントの使用

このコンポーネントを、メカニズム 2.x がさまざまな org.apache.hadoop.fs.FileSystem 実装の検出に使用する OSGi 環境で実行すると、いくつかの質問があります。Hadoop 2.x は java.util.ServiceLoader を使用して、利用可能なファイルシステムタイプと実装を定義する /META-INF/services/org.apache.hadoop.fs.FileSystem ファイルを検索します。これらのリソースは、OSGi 内で実行する場合には利用できません。

camel-hdfs コンポーネントと同様に、デフォルトの設定ファイルはバンドルクラスローダーから確認する必要があります。これに対応する一般的な方法は、バンドルルートで core-default.xml （ hdfs-default.xmlなど）のコピーを保持することです。

136.6.1. 手動で定義されたルートでのこのコンポーネントの使用

2 つの選択肢があります。

1. ルートを定義するバンドルを使用して /META-INF/services/org.apache.hadoop.fs.FileSystem リソースをパッケージ化します。このリソースは、必要なすべての Hadoop 2.x ファイルシステムの実装を一覧表示する必要があります。

2. org.apache.hadoop.fs.FileSystem クラス内で内部の静的キャッシュを設定する boilerplate 初期化コードを提供します。

```
org.apache.hadoop.conf.Configuration conf = new org.apache.hadoop.conf.Configuration();
conf.setClass("fs.file.impl", org.apache.hadoop.fs.LocalFileSystem.class, FileSystem.class);
conf.setClass("fs.hdfs.impl", org.apache.hadoop.hdfs.DistributedFileSystem.class,
FileSystem.class);
...
FileSystem.get("file:///", conf);
FileSystem.get("hdfs://localhost:9000/", conf);
...
```

136.6.2. Blueprint コンテナーでのこのコンポーネントの使用

以下の 2 つのオプションを選択できます。

1. Blueprint 定義が含まれるバンドルを使用して /META-INF/services/org.apache.hadoop.fs.FileSystem リソースをパッケージ化します。

2. 以下を Blueprint 定義ファイルに追加します。

```
<bean id="hdfsOsgiHelper" class="org.apache.camel.component.hdfs2.HdfsOsgiHelper">
  <argument>
    <map>
      <entry key="file:///" value="org.apache.hadoop.fs.LocalFileSystem"  />
      <entry key="hdfs://localhost:9000/"
value="org.apache.hadoop.hdfs.DistributedFileSystem" />
      ...
    </map>
  </argument>
</bean>

<bean id="hdfs2" class="org.apache.camel.component.hdfs2.HdfsComponent" depends-
on="hdfsOsgiHelper" />
```

このようにして、2.x が URI スキームとファイルシステムの実装と正しいマッピングを行います。

第137章 HEADERSMAP

Camel 2.20 から利用可能

camel-headersmap は、ケースセンサルマップのより高速な実装で、実行時に Camel がプラグインして使用することで、Camel Message ヘッダーのパフォーマンスが大幅に向上します。

137.1. クラスパスからの自動検出

この実装を使用するには、camel-headersmap 依存関係をクラスパスに追加し、Camel がこれを起動時に自動検出し、以下のようにログに記録する必要があります。

```
Detected and using custom HeadersMapFactory:
org.apache.camel.component.headersmap.FastHeadersMapFactory@71e9ebae
```

spring-boot の場合、camel-headersmap-starter 依存関係を使用する必要があります。

137.2. 手動有効化

OSGi を使用したり、実装がクラスパスに追加されていない場合は、この .Title を有効にする必要があります。

```
CamelContext camel = ...

camel.setHeadersMapFactory(new FastHeadersMapFactory());
```

または XML DSL（spring または Blueprint XML ファイル）では、ファクトリーを < bean> として宣言できます。

```
<bean id="fastMapFactory"
class="org.apache.camel.component.headersmap.FastHeadersMapFactory"/>
```

そして、Camel は Bean を検出し、ログに記録されるファクトリーを使用する必要があります。

第138章 HESSIAN DATAFORMAT（非推奨）

Camel バージョン 2.17 から利用可能

Hessian は、Caucho の Hessian 形式を使用したマーシャリングおよびアンマーシャリングメッセージ用のデータフォーマットです。

Maven から Hessian Data Format を使用する場合は、以下の依存関係を pom.xml に追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hessian</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

138.1. オプション

Hessian データフォーマットは、以下に示す 4 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
whitelistEnabled	**true**	**ブール値**	ホワイトリスト機能が有効かどうかを定義します。
allowedUnmarshallObjects		**文字列**	アンマーシャリングを許可するオブジェクトを定義します。
deniedUnmarshallObjects		**文字列**	アンマーシャリングする拒否オブジェクトを定義します。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

138.2. JAVA DSL での HESSIAN データフォーマットの使用

```
from("direct:in")
    .marshal().hessian();
```

138.3. SPRING DSL での HESSIAN データフォーマットの使用

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:in"/>
        <marshal ref="hessian"/>
    </route>
</camelContext>
```

第139章 HIPCHAT コンポーネント

Camel バージョン 2.15 から利用可能

Hipchat コンポーネントは、Hipchat サービスからメッセージの生成および使用をサポートします。

前提条件

有効な Hipchat ユーザーアカウントが必要で、メッセージの生成/消費に使用できる 個人アクセストークン を取得する必要があります。

139.1. URI 形式

```
hipchat://[host][:port]?options
```

URI には、？options=value&option2=value&... という形式でクエリーオプションを追加できます。

139.2. URI オプション

Hipchat コンポーネントにはオプションがありません。

Hipchat エンドポイントは、URI 構文を使用して設定します。

```
hipchat:protocol:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

139.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
protocol	**必須**: hipchat サーバーのプロトコル（http など）。		文字列

Name	説明	デフォルト	Type
host	**必須**。api.hipchat.com などの hipchat サーバーのホスト		文字列
port	hipchat サーバーのポート。デフォルトは 80 です。	80	整数

139.2.2. クエリーパラメーター（22 パラメーター）：

Name	説明	デフォルト	Type
authToken (common)	OAuth 2 認証トークン		文字列
consumeUsers (common)	hiptchat サーバーからメッセージを消費する際のユーザー名。複数のユーザー名をコンマで区切ることができます。		文字列
httpClient (common)	API HTTP リクエスト中に使用されるレジストリーからの CloseableHttpClient 参照。	HttpClient ライブラリーからの CloseableHttpClient デフォルト	CloseableHttpClient
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean

Name	説明	デフォルト	Type
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

139.3. スケジュールされたポーリングコンシューマー

このコンポーネントは ScheduledPollConsumer を実装します。指定された 'consumeUsers' の最後のメッセージのみが取得され、Exchange ボディーとして送信されます。次のポーリングに新しいメッセージがない場合に同じメッセージを再度取得しない場合は、以下のようにべき等コンシューマーを追加できます。ScheduledPollConsumer のすべてのオプションは、コンシューマーの制御にも使用できます。

```
@Override
public void configure() throws Exception {
 String hipchatEndpointUri = "hipchat://?authToken=XXXX&consumeUsers=@Joe,@John";
 from(hipchatEndpointUri)
  .idempotentConsumer(
    simple("${in.header.HipchatMessageDate} ${in.header.HipchatFromUser}"),
```

```
    MemoryIdempotentRepository.memoryIdempotentRepository(200)
  )
  .to("mock:result");
 }
```

139.3.1. Hipchat コンシューマーによって設定されたメッセージヘッダー

ヘッダー	Constant	Type	説明
HipchatFromUser	HipchatConstants.FROM_USER	**文字列**	ボディーには、このユーザーから authToken の所有者に送信されたメッセージがあります。
HipchatMessageDate	HipchatConstants.MESSAGE_DATE	**文字列**	日付メッセージが送信された。形式は、Hipchat 応答 にある ISO-8601 です。

139.4. HIPCHAT プロデューサー

プロデューサーは、**Room** と **User** の両方にメッセージを送信することができます。エクスチェンジの本文はメッセージとして送信されます。使用例を以下に示します。適切なヘッダーを設定する必要があります。

```
@Override
 public void configure() throws Exception {
  String hipchatEndpointUri = "hipchat://?authToken=XXXX";
  from("direct:start")
   .to(hipchatEndpointUri)
   .to("mock:result");
 }
```

139.4.1. Hipchat プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Constant	Type	説明
HipchatToUser	HipchatConstants.TO_USER	**文字列**	メッセージの送信が必要な Hipchat ユーザー。

ヘッダー	Constant	Type	説明
HipchatToRoom	HipchatConstants.TO_ROOM	**文字列**	メッセージを送信する必要のある Hipchat 部屋。
HipchatMessageFormat	HipchatConstants.MESSAGE_FORMAT	文字列	有効な形式は 'text' または 'html' です。**デフォルト： 'text'**
HipchatMessageBackgroundColor	HipchatConstants.MESSAGE_BACKGROUND_COLOR	**文字列**	有効な色の値は「yellow」、「green」、「red」、「purple」、「gray」、「random」です。**デフォルト： 'yellow'(Room Only)**
HipchatTriggerNotification	HipchatConstants.TRIGGER_NOTIFY	**文字列**	有効な値は「true」または「false」です。このメッセージがユーザー通知をトリガーするかどうか（タブの色の変更、サウンドの再生、モバイル電話など）。**デフォルト： 'false'(Room Only)**

139.4.2. Hipchat プロデューサーによって設定されたメッセージヘッダー

ヘッダー	Constant	Type	説明

ヘッダー	Constant	Type	説明
HipchatToUserResponseStatus	HipchatConstants.TO_USER_RESPONSE_STATUS	StatusLine: ユーザーがメッセージを送信する際に受信される API 応答のステータス。	HipchatFromUserResponseStatus

139.4.3. Http Client の設定

HipChat コンポーネントは、独自の HttpClient 設定を許可します。これは、レジストリー（Spring Context など）で CloseableHttpClient への参照を定義してから、エンドポイント定義中にパラメーターを設定します（例： hip chat:http://api.hipchat.com?httpClient=#myHttpClient）。

```
CloseableHttpClient httpclient = HttpClients.custom()
    .setConnectionManager(connManager)
    .setDefaultCookieStore(cookieStore)
    .setDefaultCredentialsProvider(credentialsProvider)
    .setProxy(new HttpHost("myproxy", 8080))
    .setDefaultRequestConfig(defaultRequestConfig)
    .build();
```

Http Client 設定の詳細は、公式ドキュメント を参照してください。

139.4.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-hipchat</artifactId>
    <version>${camel-version}</version>
</dependency>
```

-

ここで、${camel-version} は Camel の実際のバージョン（2.15.0 以降）に置き換える必要があります。

第140章 HL7 DATAFORMAT

Camel バージョン 2.0 で利用可能

HL7 コンポーネントは、HAPI ライブラリー を使用した HL7 MLLP プロトコルおよび HL7 v2 メッセージ の使用に使用されます。

このコンポーネントは以下をサポートします。

- *Minaの HL7 MLLP コーデック*
- *Camel 2.15 以降の Netty4 の HL7 MLLP コーデック*
- *Type Converter from/to HAPI and String*
- *HAPI ライブラリーを使用した HL7 DataFormat*
- *camel-mina2 コンポーネントと統合されるため、より簡単に使用できます。*

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hl7</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

140.1. HL7 MLLP プロトコル

HL7 は、テキストベースの TCP ソケットベースのプロトコルである HL7 MLLP プロトコルでよく使用されます。このコンポーネントには、MLLP プロトコルに準拠する Mina および Netty4 Codec が同梱されています。そのため、TCP トランスポート層で HL7 リクエストを受け入れる HL7 リスナーを簡単に公開できます。HL7 リスナーサービスを公開するには、camel-mina2 または camel-netty4 コン

ポーネントは HL7MLLPCodec (mina2)または HL7MLLPNettyDecoder/HL7MLLPNettyEncoder (Netty4)で使用されます。

HL7 MLLP コーデックは、以下のように設定できます。

Name	デフォルト値	説明
startByte	**0x0b**	HL7 ペイロード全体の開始バイト。
endByte1	**0x1c**	HL7 ペイロードにまたがる最初の終了バイト。
endByte2	**0x0d**	HL7 ペイロードにまたがる 2 番目の終了バイト。
charset	JVM のデフォルト	codec に使用するエンコーディング（ 文字セット名）です。指定され ていない場合、Camel は JVM デフォルトの Charset を使用します。
produceString	**true**	**(Camel 2.14.1)true の場合、コーデックは定義された文字セットを使用して** 文字列を作成します。false の場合、コーデックはプレーンなバイトアレイをルートに送信し、HL7 Data Format が HL7 メッセージコンテンツから実際の文字セットを判断できるようにします。
convertLFtoCR	**false**	**\n** を **\r** （**0x0d**, 13 進法）に変換します。HAPI ライブラリーには**\r** を使用する必要があります。

140.1.1. Mina を使用した HL7 リスナーの公開

Spring XML ファイルでは、ポート 8888 で TCP を使用して HL7 要求をリッスンするように mina2 エンドポイントを設定します。

```
<endpoint id="hl7MinaListener" uri="mina2:tcp://localhost:8888?sync=true&amp;codec=#hl7codec"/>
```

sync=true は、このリスナーが同期されているため、呼び出し元に HL7 応答を返すことを示します。HL7 コーデックは codec=#hl7codec で設定されます。hl7codec は Spring Bean ID であるため、mygreatcodecforhl7 などの名前を付けることができます。codec も Spring XML ファイルに設定されます。

```
<bean id="hl7codec" class="org.apache.camel.component.hl7.HL7MLLPCodec">
    <property name="charset" value="iso-8859-1"/>
</bean>
```

エンドポイント hl7MinaLlistener はコンシューマーとしてルートで使用できます。これは Java DSL の例になります。

```
from("hl7MinaListener")
  .bean("patientLookupService");
```

これは HL7 をリッスンし、Snational LookupService という名前のサービスにルーティングする非常にシンプルなルートです。これは Spring XML で以下のとおりに設定された Spring Bean ID でもあります。

```
<bean id="patientLookupService"
class="com.mycompany.healthcare.service.PatientLookupService"/>
```

ビジネスロジックは、Camel に依存しない POJO クラスに実装できます。以下に例を示します。

```
import ca.uhn.hl7v2.HL7Exception;
import ca.uhn.hl7v2.model.Message;
import ca.uhn.hl7v2.model.v24.segment.QRD;

public class PatientLookupService {
    public Message lookupPatient(Message input) throws HL7Exception {
        QRD qrd = (QRD)input.get("QRD");
        String patientId = qrd.getWhoSubjectFilter(0).getIDNumber().getValue();

        // find patient data based on the patient id and create a HL7 model object with the
response
        Message response = ... create and set response data
        return response
    }
```

140.1.2. Netty を使用した HL7 リスナーの公開（Camel 2.15 以降で利用可能）

Spring XML ファイルでは、ポート 8888 で TCP を使用して HL7 要求をリッスンするように netty4 エンドポイントを設定します。

```
<endpoint id="hl7NettyListener" uri="netty4:tcp://localhost:8888?
sync=true&amp;encoder=#hl7encoder&amp;decoder=#hl7decoder"/>
```

sync=true は、このリスナーが同期されているため、呼び出し元に HL7 応答を返すことを示します。HL7 codec は encoder=#hl7encoder*and*decoder=#hl7decoder で設定されます。hl7encoder

および hl7decoder は単に Bean ID であるため、異なる名前を付けることができます。Bean は Spring XML ファイルで設定できます。

```
<bean id="hl7decoder" class="org.apache.camel.component.hl7.HL7MLLPNettyDecoderFactory"/>
<bean id="hl7encoder" class="org.apache.camel.component.hl7.HL7MLLPNettyEncoderFactory"/>
```

エンドポイント hl7NettyListener は、以下の Java DSL の例になります。

```
from("hl7NettyListener")
  .bean("patientLookupService");
```

140.2. JAVA.LANG.STRING または BYTE[] を使用した HL7 モデル

HL7 MLLP コーデックは、データ形式としてプレーンな String を使用します。Camel は Converter を使用して文字列から HAPI HL7 モデルオブジェクトに変換しますが、必要に応じてデータを自分で解析する場合は、プレーンな String オブジェクトを使用できます。

Camel 2.14.1 以降では、generate String プロパティーを false に設定すると、Ma および Netty コーデックの両方がプレーンな byte[] をデータ形式として使用することもできます。Type Converter は、HAPI HL7 モデルオブジェクトから byte[] を/から変換することもできます。

140.3. HAPI を使用した HL7V2 モデル

HL7v2 モデルは、HAPI ライブラリーの Java オブジェクトを使用します。このライブラリーを使用すると、主に HL7v2 で使用される EDI 形式(ER7)からエンコードおよびデコードできます。

以下の例は、患者 ID 0101701234 で患者を検索するリクエストです。

```
MSH|^~\\&|MYSENDER|MYRECEIVER|MYAPPLICATION||200612211200||QRY^A19|1234|P|2.4
QRD|200612211200|R|I|GetPatient|||1^RD|0101701234|DEM||
```

HL7 モデルを使用すると、ca.uhn.hl7v2.model.Message オブジェクトで使用できます。 たとえば、患者 ID を取得するには以下を行います。

```
Message msg = exchange.getIn().getBody(Message.class);
QRD qrd = (QRD)msg.get("QRD");
String patientId = qrd.getWhoSubjectFilter(0).getIDNumber().getValue();  // 0101701234
```

これは、バイト[]、String、またはその他の単純なオブジェクト形式と連携する必要がないため、

HL7 リスナーと組み合わせる場合に強力です。HAPI HL7v2 モデルオブジェクトのみを使用できます。メッセージタイプが事前に分かっている場合は、タイプセーフになります。

```
QRY_A19 msg = exchange.getIn().getBody(QRY_A19.class);
String patientId = msg.getQRD().getWhoSubjectFilter(0).getIDNumber().getValue();
```

140.4. HL7 DATAFORMAT

HL7 コンポーネントには HL7 データフォーマットが同梱されており、HL7 モデルオブジェクトをマーシャリングまたはアンマーシャリングするために使用できます。

HL7 データフォーマットは、以下に示す 2 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
validate	**true**	**ブール値**	HL7 メッセージを検証するかどうか。デフォルトは true です。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

- *marshal = from Message からバイトストリーム（HL7 MLLP コーデックを使用して応答する場合に使用できます）。*
- *unmarshal = バイトストリームからメッセージへの送信（HL7 MLLP からストリーミングデータを受信するときに使用可能）*

データフォーマットを使用するには、インスタンスをインスタンス化し、ルートビルダーでマーシャリングまたはアンマーシャリング操作を呼び出します。

```
DataFormat hl7 = new HL7DataFormat();

from("direct:hl7in")
  .marshal(hl7)
  .to("jms:queue:hl7out");
```

上記の例では、HL7 は HAPI Message オブジェクトからバイトストリームにマーシャリングされ、JMS キューに置かれます。
次の例は逆になります。

```
DataFormat hl7 = new HL7DataFormat();

from("jms:queue:hl7out")
  .unmarshal(hl7)
  .to("patientLookupService");
```

ここでは、バイトストリームを、患者の検索サービスに渡される HAPI Message オブジェクトにアンマーシャリングします。

140.4.1. pidgin messages

HAPI 2.0（ Camel 2.11で使用される）の時点で、HL7v2 モデルクラスは完全にシリアライズ可能です。そのため、HL7v2 メッセージを直接 JMS キューに置くことができます（例： marshal（） を呼び出しせずに）、再度キューから再度（ unmarshal（） を呼び出すことなく）読み取ることができます。

140.4.2. segment separators

Camel 2.11 の時点で、アンマーシャリング は \n を \r に変換して自動的にセグメントセパレーターを修正しません。
にこの変換が必要な場合、org.apache.camel.component.hl7.HL7#convertLFToCR はこの目的の便利な 式 を提供します。

140.4.3. charset

Camel 2.14.1 の時点で、マーシャリングおよびアンマーシャリング は MSH-18 フィールドで提供される文字セットを評価し ます。このフィールドが空の場合、デフォルトでは対応する Camel charset プロパティー/ヘッダーに含まれる文字セットが想定されます。HL7DataFormat クラスから継承する際に guess CharsetName メソッド を上書きすることで、このデフォルトの動作を変更できます。

よく知られているデータ形式に関する Camel には簡単な構文があります。HL7DataFormat オブジェクトのインスタンスを作成する必要はありません。

```
from("direct:hl7in")
  .marshal().hl7()
```

```
    .to("jms:queue:hl7out");

from("jms:queue:hl7out")
  .unmarshal().hl7()
  .to("patientLookupService");
```

140.5. メッセージヘッダー

unmarshal 操作は、これらのフィールドを Camel メッセージのヘッダーとして MSH セグメントに追加します。

キー	MSH フィールド	例
CamelHL7SendingApplication	MSH-3	MYSERVER
CamelHL7SendingFacility	MSH-4	MYSERVERAPP
CamelHL7ReceivingApplication	MSH-5	MYCLIENT
CamelHL7ReceivingFacility	MSH-6	MYCLIENTAPP
CamelHL7Timestamp	MSH-7	20071231235900
CamelHL7Security	MSH-8	null

キー	MSH フィールド	例
CamelHL7MessageType	**MSH-9-1**	**ADT**
CamelHL7TriggerEvent	**MSH-9-2**	**A01**
CamelHL7MessageControl	**MSH-10**	**1234**
CamelHL7ProcessingId	**MSH-11**	**%P**
CamelHL7VersionId	**MSH-12**	**2.4**
`CamelHL7Context	``	' **(Camel 2.14)** には、メッセージの解析に使用された HapiContext が含まれます。
CamelHL7Charset	**MSH-18**	(Camel 2.14.1) **UNICODE UTF-8**

CamelHL7Context 'are 'String types を除くすべてのヘッダー。ヘッダーの値がない場合、その値は null になります。

140.6. オプション

HL7 Data Format は、以下のオプションをサポートします。

オプション	デフォルト	説明
validate	true	HAPI Parser がデフォルトの検証ルールを使用してメッセージを検証すべきかどうか。**parser** または **hapiContext** オプションを使用して、必要な HAPI ValidationContextで初期化することが推奨されます。
parser	**ca.uhn.hl7v2.parser.GenericParser**	使用されるカスタムパーサー。**ca.uhn.hl7v2.parser.Parser** タイプである必要があります。**GenericParser** では、XML でエンコードされた HL7v2 メッセージの解析もできることに注意してください。
hapiContext	**ca.uhn.hl7v2.DefaultHapiContext**	Camel 2.14: カスタムパーサー、カスタム ValidationContext 等を定義するカスタム HAPI コンテキスト。これにより、HL7 解析およびレンダリングプロセスを完全に制御できます。

140.7. 依存関係

Camel ルートで HL7 を使用するには、上記の camel-hl7 の依存関係を追加して、このデータ形式を実装する必要があります。

HAPI ライブラリーは、HL7v2 メッセージバージョンごとに、ベースライブラリー と複数の構造ライブラリーに分割されます。

- *v2.1 structures library*
- *v2.2 構造ライブラリー*
- *v2.3 構造ライブラリー*
- *v2.3.1 structures library*
- *v2.4 structures library*

- *v2.5 structures library*
- *v2.5.1 structures library*
- *v2.6 構造ライブラリー*

デフォルトでは、camel-hl7 は HAPI ベースライブラリー のみを参照します。アプリケーションは、構造ライブラリー自体をインクルードします。たとえば、アプリケーションが HL7v2 メッセージバージョン 2.4 および 2.5 で機能する場合は、以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>ca.uhn.hapi</groupId>
  <artifactId>hapi-structures-v24</artifactId>
  <version>2.2</version>
  <!-- use the same version as your hapi-base version -->
</dependency>
<dependency>
  <groupId>ca.uhn.hapi</groupId>
  <artifactId>hapi-structures-v25</artifactId>
  <version>2.2</version>
  <!-- use the same version as your hapi-base version -->
</dependency>
```

または、ベースライブラリーを含む OSGi バンドルで、すべての構造ライブラリーと必要な依存関係（バンドルクラスパス上）を 中央 Maven リポジトリー からダウンロードできます。

```
<dependency>
  <groupId>ca.uhn.hapi</groupId>
  <artifactId>hapi-osgi-base</artifactId>
  <version>2.2</version>
</dependency>
```

140.8. TERSER 言語

HAPI は、一般的に使用される簡潔な場所の指定構文を使用してフィールドへのアクセスを提供する Terser クラスを提供します。Terser 言語は、この構文を使用してメッセージから値を抽出し、それらをフィルター、コンテンツベースのルーティングの式および述語として使用することができます。

sample:

```
import static org.apache.camel.component.hl7.HL7.terser;

  // extract patient ID from field QRD-8 in the QRY_A19 message above and put into message
header
  from("direct:test1")
    .setHeader("PATIENT_ID",terser("QRD-8(0)-1"))
    .to("mock:test1");

  // continue processing if extracted field equals a message header
  from("direct:test2")
    .filter(terser("QRD-8(0)-1").isEqualTo(header("PATIENT_ID"))
    .to("mock:test2");
```

140.9. HL7 VALIDATION PREDICATE

多くの場合、最初に HL7v2 メッセージを解析し、別のステップで HAPI ValidationContext に対して検証することが推奨されます。

sample:

```
import static org.apache.camel.component.hl7.HL7.messageConformsTo;
import ca.uhn.hl7v2.validation.impl.DefaultValidation;

  // Use standard or define your own validation rules
  ValidationContext defaultContext = new DefaultValidation();

  // Throws PredicateValidationException if message does not validate
  from("direct:test1")
    .validate(messageConformsTo(defaultContext))
    .to("mock:test1");
```

140.10. HAPICONTEXT(CAMEL 2.14)を使用した HL7 VALIDATION 述語

HAPI Context は常に ValidationContext （または ValidationRuleBuilder）で設定されるため、検証ルールは間接的にアクセスできます。さらに、HL7DataFormat が CamelHL7Context ヘッダーに設定された HAPI コンテキストを転送する際に、このコンテキストの検証ルールを簡単に再利用できます。

```
import static org.apache.camel.component.hl7.HL7.messageConformsTo;
import static org.apache.camel.component.hl7.HL7.messageConforms

  HapiContext hapiContext = new DefaultHapiContext();
  hapiContext.getParserConfiguration().setValidating(false); // don't validate during parsing

  // customize HapiContext some more ... e.g. enforce that PID-8 in ADT_A01 messages of
version 2.4 is not empty
  ValidationRuleBuilder builder = new ValidationRuleBuilder() {
```

```
    @Override
    protected void configure() {
      forVersion(Version.V24)
          .message("ADT", "A01")
          .terser("PID-8", not(empty()));
      }
    };
hapiContext.setValidationRuleBuilder(builder);

HL7DataFormat hl7 = new HL7DataFormat();
hl7.setHapiContext(hapiContext);

from("direct:test1")
  .unmarshal(hl7)              // uses the GenericParser returned from the HapiContext
  .validate(messageConforms())   // uses the validation rules returned from the HapiContext
                          // equivalent with .validate(messageConformsTo(hapiContext))
  // route continues from here
```

140.11. HL7 ACKNOWLEDGEMENT EXPRESSION

HL7v2 処理の一般的なタスクは、検証結果に基づいて受信 HL7v2 メッセージへの応答として確認メッセージを生成することです。ack 式により、これを適切に完了させることができます。

```
import static org.apache.camel.component.hl7.HL7.messageConformsTo;
import static org.apache.camel.component.hl7.HL7.ack;
import ca.uhn.hl7v2.validation.impl.DefaultValidation;

 // Use standard or define your own validation rules
  ValidationContext defaultContext = new DefaultValidation();

 from("direct:test1")
   .onException(Exception.class)
     .handled(true)
     .transform(ack()) // auto-generates negative ack because of exception in Exchange
     .end()
   .validate(messageConformsTo(defaultContext))
   // do something meaningful here

   // acknowledgement
   .transform(ack())
```

140.12. 追加のサンプル

以下の例では、応答を送信する HL7 リスナーに単純な String HL7 リクエストが送信されます。

次の例では、HL7 リスナーからの HL 7 リクエストはビジネスロジックにルーティングされます。

次に、RouteBuilder を使用する Camel ルートは、以下のようになります。

HL7 DataFormat を使用すると、Camel メッセージヘッダーに MSH セグメントのフィールドが追加されることに注意してください。ヘッダーは、上記の例のようにフィルタリングまたはコンテンツベースのルーティングに特に便利です。

第141章 HTTP コンポーネント（非推奨）

Camel バージョン 1.0 で利用可能

http: コンポーネントは、外部 HTTP リソースを使用するための HTTP ベースのエンドポイントを提供します（HTTP を使用して外部サーバーを呼び出すクライアントとして）。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-http</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

141.1. URI 形式

```
http:hostname[:port][/resourceUri][?param1=value1][&param2=value2]
```

デフォルトでは、HTTP にはポート 80 を使用し、HTTPS には 443 を使用します。

camel-http vs camel-jetty

HTTP コンポーネントで生成されるエンドポイントにのみ生成できます。そのため、Camel ルートへの入力として使用することはできません。HTTP サーバー経由で HTTP エンドポイントを camel ルートへの入力としてバインド/公開するには、Jetty コンポーネントまたは Servlet コンポーネントを使用します。

141.2. 例

POST を使用してボディーで URL を呼び出し、応答を out メッセージとして返します。ボディーが GET を使用して null 呼び出し URL で、応答を out メッセージとして返す場合

Java DSL

Spring DSL

```
from("direct:start")
  .to("http://myhost/mypath");
```

```
<from uri="direct:start"/>
<to uri="http://oldhost"/>
```

ヘッダーを追加することで、HTTP エンドポイント URI を上書きできます。Camel は http://newhost を呼び出します。これは REST URL などに非常に便利です。

Java DSL

```
from("direct:start")
  .setHeader(Exchange.HTTP_URI, simple("http://myserver/orders/${header.orderId}"))
  .to("http://dummyhost");
```

URI パラメーターはエンドポイント URI に直接設定することも、ヘッダーとして設定することもできます。

Java DSL

```
from("direct:start")
  .to("http://oldhost?order=123&detail=short");
from("direct:start")
  .setHeader(Exchange.HTTP_QUERY, constant("order=123&detail=short"))
  .to("http://oldhost");
```

HTTP 要求メソッドを POST に設定します。

Java DSL

Spring DSL

```
from("direct:start")
  .setHeader(Exchange.HTTP_METHOD, constant("POST"))
  .to("http://www.google.com");
```

```
<from uri="direct:start"/>
<setHeader headerName="CamelHttpMethod">
  <constant>POST</constant>
</setHeader>
<to uri="http://www.google.com"/>
<to uri="mock:results"/>
```

141.3. HTTP オプション

HTTP コンポーネントは、以下に示す 8 個のオプションをサポートします。

Name	説明	デフォルト	Type
httpClientConfigurer (advanced)	カスタム HttpClientConfigurer を使用して使用される HttpClient の設定を行うには、以下を行います。		HttpClientConfigurer
httpConnectionManager (advanced)	カスタムの HttpConnectionManager を使用して接続を管理するには		HttpConnectionManager
httpBinding (producer)	カスタムの HttpBinding を使用して Camel メッセージと HttpClient 間のマッピングを制御する場合。		HttpBinding
httpConfiguration (producer)	共有 HttpConfiguration をベース設定として使用します。		HttpConfiguration
allowJavaSerializedObject (producer)	リクエストが context-type=application/x-java-serialized-object を使用する場合の java シリアライゼーションを許可するかどうか。これはデフォルトでオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

HTTP エンドポイントは、URI 構文を使用して設定します。

```
http:httpUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

141.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
httpUri	***必須***。呼び出す HTTP エンドポイントの URL。		URI

141.3.2. クエリーパラメーター（38 パラメーター）：

Name	説明	デフォルト	Type
disableStreamCache (common)	Servlet からの raw 入力ストリームがキャッシュされているかどうかを決定します（Camel はストリームをメモリー内/オーバーフローでファイル、ストリームキャッシング）キャッシュに読み取ります。デフォルトでは、Camel は Servlet 入力ストリームをキャッシュして、複数回ロードし、Camel がストリームからすべてのデータを取得できるようにします。ただし、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。たとえば、ファイルまたは他の永続ストアに直接ストリーミングする場合などに、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。DefaultHttpBinding は、ストリームの読み取りを複数回サポートするために、このオプションが false の場合は、リクエスト入力ストリームをストリームキャッシュにコピーし、メッセージボディーに配置します。Servlet を使用してエンドポイントをブリッジ/プロキシーする場合、このオプションを有効にしてパフォーマンスを向上することを検討してください。メッセージペイロードを複数回読み取る必要がない場合は、このオプションを有効にします。http/http4 プロデューサーは、デフォルトでは応答本体ストリームをキャッシュします。このオプションを true に設定すると、プロデューサーは応答ボディーストリームをキャッシュしませんが、応答ストリームをメッセージボディーとして使用します。	false	boolean

Name	説明	デフォルト	Type
headerFilterStrategy (common)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
httpBinding (common)	カスタムの HttpBinding を使用して Camel メッセージと HttpClient 間のマッピングを制御する場合。		HttpBinding
bridgeEndpoint (producer)	オプションが true の場合、HttpProducer は Exchange.HTTP_URI ヘッダーを無視し、リクエストにエンドポイントの URI を使用します。また、ththExceptionOnFailure オプションを false に設定して、HttpProducer がすべての障害応答を返せるようにすることもできます。	false	boolean
チャンク （プロデューサー）	このオプションが false の場合、サーブレットは HTTP ストリーミングを無効にし、応答に content-length ヘッダーを設定します。	true	boolean
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを指定します。デフォルトでは connectionClose は false です。	false	boolean
copyHeaders (producer)	このオプションが true の場合、コピーストラテジーに従って IN エクスチェンジヘッダーが OUT エクスチェンジヘッダーにコピーされます。これを false に設定し、HTTP 応答からのヘッダーのみを含めることができます（IN ヘッダーは伝播しません）。	true	boolean
httpMethod (producer)	使用する HTTP メソッドを設定します。HttpMethod ヘッダーが設定されている場合は、このオプションをオーバーライドできません。		HttpMethods
ignoreResponseBody (producer)	このオプションが true の場合、http プロデューサーは応答本体を読み取りせず、入力ストリームをキャッシュします。	false	boolean
preserveHostHeader (producer)	オプションが true の場合、HttpProducer は Host ヘッダーを現在のエクスチェンジ Host ヘッダーに含まれる値に設定します。これは、ダウンストリームサーバーが受け取る Host ヘッダーを使用してアップストリームクライアントが呼び出した URL を反映させたいリバースプロキシーアプリケーションで有用です。これにより、Host ヘッダーを使用するアプリケーションがプロキシーされるサービスの正確な URL を生成することができます。	false	boolean

Name	説明	デフォルト	Type
throwExceptionOnFailure (producer)	リモートサーバーからの応答に失敗した場合に HttpOperationFailedException のスローを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
transferException (producer)	有効で Exchange がコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object のコンテンツタイプとして応答でシリアライズされたかどうか。プロデューサー側では、例外は HttpOperationFailedException ではなくデシリアライズされ、そのままスローされます。原因となる例外はシリアライズされている必要があります。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
cookieHandler (producer)	HTTP セッションを維持するためのクッキーハンドラーの設定		CookieHandler
okStatusCodeRange (producer)	正常な応答とみなされるステータスコード。値は含まれます。コンマで区切られた複数の範囲を定義できます（例： 200-204,209,301-304）。各範囲は、1 つの数字またはダッシュを含む from から でなければなりません。	200-299	文字列
urlRewrite (producer)	**非推奨** の org.apache.camel.component.http.UrlRewrite への参照。ブリッジ/プロキシーエンドポイントの実行時に URL を書き換えることができるようになりました。詳細は、http://camel.apache.org/urlrewrite.htmlを参照してください。		UrlRewrite
httpClientConfigurer (advanced)	認証メカニズムの設定など、プロデューサーやコンシューマーによって作成された新しい HttpClient インスタンスのカスタム設定ストラテジーを登録します。		HttpClientConfigurer
httpClientOptions (advanced)	Map のキー/値を使用して HttpClient を設定します。		マップ
httpConnectionManager (advanced)	カスタムの HttpConnectionManager を使用して接続を管理するには		HttpConnectionManager

Name	説明	デフォルト	Type
httpConnectionManager Options (advanced)	Map のキー/値を使用して HttpConnectionManager を設定します。		マップ
mapHttpMessageBody (advanced)	このオプションが true の場合、エクスチェンジの IN エクスチェンジボディーは HTTP ボディーにマッピングされます。false に設定すると HTTP マッピングが回避されます。	true	boolean
mapHttpMessageFormUrlEncodedBody (advanced)	このオプションが true の場合、エクスチェンジの IN Exchange Form Encoded body が HTTP にマッピングされます。これを false に設定すると、HTTP Form Encoded body マッピングを回避します。	true	boolean
mapHttpMessageHeaders (advanced)	このオプションが true の場合、エクスチェンジの IN エクスチェンジヘッダーは HTTP ヘッダーにマッピングされます。false に設定すると、HTTP ヘッダーのマッピングが回避されます。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
proxyAuthDomain (proxy)	NTML で使用するプロキシー認証ドメイン		文字列
proxyAuthHost (proxy)	プロキシー認証ホスト		文字列
proxyAuthMethod (proxy)	使用するプロキシー認証方法		文字列
proxyAuthPassword (proxy)	プロキシー認証パスワード		文字列
proxyAuthPort (proxy)	プロキシー認証ポート		int
proxyAuthScheme (proxy)	使用するプロキシー認証スキーム		文字列
proxyAuthUsername (proxy)	プロキシー認証ユーザー名		文字列
proxyHost (proxy)	使用するプロキシーホスト名		文字列

Name	説明	デフォルト	Type
proxyPort (proxy)	使用するプロキシーポート		int
authDomain (security)	NTML で使用する認証ドメイン		文字列
authHost (security)	NTML で使用する認証ホスト		文字列
authmethod （セキュリティー）	Basic、Digest、または NTLM の値のコンマ区切りリストとして使用できる認証方法。		文字列
authMethodPriority (security)	Basic、Digest、または NTLM のいずれかの使用を優先する認証方法。		文字列
authPassword (security)	認証パスワード		文字列
authUsername (security)	認証ユーザー名		文字列

141.4. メッセージヘッダー

Name	タイプ	説明
Exchange.HTTP_URI	**文字列**	呼び出す URI。エンドポイントに直接設定された既存の URI を上書きします。この URI は、呼び出す http サーバーの URI です。これは、セキュリティーなどのエンドポイントオプションを設定する Camel エンドポイント URI と同じです。このヘッダーは、http サーバーの URI のみをサポートしません。
Exchange.HTTP_METHOD	**文字列**	使用する HTTP メソッド / Verb(GET/POST/PUT/DELETE/HEAD/OPTIONS/TRACE)
Exchange.HTTP_PATH	**文字列**	リクエスト URI のパス。ヘッダーは、リクエスト URI を HTTP_URI でビルドするために使用されます。**Camel 2.3.0:** パスが「/」で始まる場合、http プロデューサーは Exchange.HTTP_BASE_URI ヘッダーまたは exchange.getFromEndpoint（）.getEndpointUri（）; に基づく相対パスの検索を試行します。
Exchange.HTTP_QUERY	**文字列**	URI パラメーターエンドポイントに直接設定された既存の URI パラメーターを上書きします。

Name	タイプ	説明
Exchange.HTTP_RESPONSE_CODE	**int**	外部サーバーからの HTTP 応答コード。OK の場合は 200 です。
Exchange.HTTP_CHARACTER_ENCODING	**文字列**	文字エンコーディング。
Exchange.CONTENT_TYPE	**文字列**	HTTP コンテンツタイプ。**テキスト/html** などのコンテンツタイプを提供するために IN メッセージと OUT メッセージの両方に設定されます。
Exchange.CONTENT_ENCODING	**文字列**	HTTP コンテンツのエンコーディング。**gzip** などのコンテンツエンコーディングを提供するために IN メッセージと OUT メッセージの両方に設定されます。
Exchange.HTTP_SERVLET_REQUEST	**HttpServletRequest**	**HttpServletRequest** オブジェクト。
Exchange.HTTP_SERVLET_RESPONSE	**HttpServletResponse**	**HttpServletResponse** オブジェクト。

Name	タイプ	説明
Exchange.HTTP_PROTOCOL_VERSION	**文字列**	Camel 2.5: このヘッダーを使用して http プロトコルバージョンを設定できます（例：）。"HTTP/1.0".ヘッダーを指定しない場合、HttpProducer はデフォルト値「HTTP/1.1」を使用します。

上記のヘッダー名は定数です。Spring DSL では、名前の代わりに定数の値を使用する必要があります。

141.5. メッセージボディー

Camel は外部サーバーからの HTTP 応答を OUT ボディーに保存します。IN メッセージからのヘッダーはすべて OUT メッセージにコピーされるため、ヘッダーはルーティング時に保持されます。さらに Camel は HTTP 応答ヘッダーと OUT メッセージヘッダーを追加します。

141.6. レスポンスコード

Camel は HTTP レスポンスコードに従って処理されます。

- *応答コードは 100..299 の範囲にあり、Camel は成功レスポンスとして認識します。*
- *応答コードは 300..399 の範囲にあり、Camel はこれをリダイレクト応答として認識し、情報を使用して HttpOperationFailedException を発生させます。*
- *応答コードは 400 以上で、Camel はこれを外部サーバーの失敗として認識し、情報を使用して HttpOperationFailedException を発生させます。*

throwExceptionOnFailure

オプション throwExceptionOnFailure を false に設定して、HttpOperationFailedException が失敗した応答コードに対してスローされないようにします。これにより、リモートサーバーから応答を取得できます。
以下に、これを説明する例を示します。

141.7. HTTPOPERATIONFAILEDEXCEPTION

この例外には、以下の情報が含まれます。

- HTTP ステータスコード
- HTTP ステータス行（ステータスコードのテキスト）
- リダイレクトを返した場合のリダイレクト場所のリダイレクト
- サーバーがボディーを応答として提供した場合、応答本体は java.lang.String として応答します。

141.8. 使用される HTTP メソッド

以下のアルゴリズムを使用して、どの HTTP メソッドを使用するかを決定します。
1エンドポイント設定として提供されるメソッド(httpMethod)を使用します。
2.ヘッダーで提供されるメソッド(Exchange.HTTP_METHOD)を使用します。
3.クエリー文字列がヘッダーで提供される場合の GET。
4.エンドポイントがクエリー文字列で設定されている場合の GET。
5.送信するデータがある場合（ボディーは nullではありません）。
6.そうでない場合は GET。

141.9. HTTPSERVLETREQUEST および HTTPSERVLETRESPONSE へのアクセス方法

これらの 2 つは、以下を使用して Camel 型コンバーターシステムを使用して取得できます。

```
HttpServletRequest request = exchange.getIn().getBody(HttpServletRequest.class);
HttpServletRequest response = exchange.getIn().getBody(HttpServletResponse.class);
```

141.10. クライアントタイムアウトの使用 - SO_TIMEOUT

このリンクのユニットテストを参照してください。

141.11. その他の例

141.11.1. プロキシーの設定

Java DSL

```
from("direct:start")
  .to("http://oldhost?proxyHost=www.myproxy.com&proxyPort=80");
```

proxyUsername および proxyPassword オプションを使用したプロキシー認証もサポートされます。

141.11.2. URI 外でのプロキシー設定の使用

Java DSL

Spring DSL

```
context.getProperties().put("http.proxyHost", "172.168.18.9");
context.getProperties().put("http.proxyPort" "8080");
```

```
<camelContext>
    <properties>
        <property key="http.proxyHost" value="172.168.18.9"/>
        <property key="http.proxyPort" value="8080"/>
    </properties>
</camelContext>
```

Endpoint のオプションは、コンテキストのオプションを上書きします。

141.12. 文字セットの設定

POST を使用してデータを送信する場合は、charsetを設定できます。

```
setProperty(Exchange.CHARSET_NAME, "iso-8859-1");
```

141.13. スケジュールされたポーリングの例

この例では、Google ホームページを 10 秒ごとにポーリングし、ページをファイル message.html に書き込みます。

```
from("timer://foo?fixedRate=true&delay=0&period=10000")
    .to("http://www.google.com")
    .setHeader(FileComponent.HEADER_FILE_NAME, "message.html").to("file:target/google");
```

141.14. 応答コードの取得

Exchange.HTTP_RESPONSE_CODE の Out メッセージヘッダーから値を取得することにより、HTTP コンポーネントから HTTP 応答コードを取得できます。

```
Exchange exchange = template.send("http://www.google.com/search", new Processor() {
        public void process(Exchange exchange) throws Exception {
            exchange.getIn().setHeader(Exchange.HTTP_QUERY,
constant("hl=en&q=activemq"));
        }
});
Message out = exchange.getOut();
int responseCode = out.getHeader(Exchange.HTTP_RESPONSE_CODE, Integer.class);
```

141.15. THROWEXCEPTIONONFAILURE=FALSE を使用した応答の取得

以下のルートでは、リモートの HTTP 呼び出しから返されたデータにエンリッチするメッセージをルーティングします。リモートサーバーからの応答が必要な場合は、throwExceptionOnFailure オプションを false に設定して、AggregationStrategy で応答を取得します。このコードは、HTTP ステータスコード 404 をシミュレートするユニットテストをベースとしているため、アサーションコードなどがあります。

141.16. クッキーの無効化

Cookie を無効にするには、この URI オプション
httpClient.cookiePolicy=ignoreCookiesを追加して HTTP クライアントがクッキーを無視するように設定できます。httpClient.cookiePolicy=ignoreCookies

141.17. 高度な使用方法

HTTP プロデューサーをより詳細に制御する必要がある場合は、さまざまなクラスを設定してカスタム動作を提供する HttpComponent を使用する必要があります。

141.17.1. Setting MaxConnectionsPerHost

HTTP コンポーネントには org.apache.commons.httpclient.HttpConnectionManager があり、指定のコンポーネントにさまざまなグローバル設定を設定できます。
グローバルの場合、コンポーネントが作成するすべてのエンドポイントに同じ

HttpConnectionManager があることを意味します。そのため、ホストごとに最大接続に異なる値を設定する場合は、通常使用するエンドポイント URI では なく、HTTP コンポーネントで定義する必要があります。これには、以下の特徴があります。

まず、Spring XML で http コンポーネントを定義します。はい、同じスキーム名 http を使用します。指定しないと、Camel はデフォルト設定でコンポーネントを自動検出して作成します。オプションを設定するために、これをオーバールールする必要があること。以下の例では、最大接続をデフォルトの 2 ではなく 5 に設定します。

また、これは通常ルートで行うのと同じように使用できます。

141.17.2. プリエンプション認証の使用

エンドユーザーは、HTTPS での認証に問題があると報告されました。この問題は、HTTPS サーバーが HTTP コード 401 Authorization Required を返さない際に解決されました。ソリューションは、httpClient.authenticationPreemptive=trueの URI オプションを設定することです。

141.17.3. リモートサーバーからの自己署名証明書の許可

Apache Commons HTTP API でこれを行う方法を概説するために、一部のコードを使用したメーリングリストからのこの リンク を参照してください。

141.17.4. HTTP クライアントの SSL 設定

JSSE 設定ユーティリティーの使用

Camel 2.8 より、HTTP4 コンポーネントは Camel JSSE 設定ユーティリティーを介して SSL/TLS 設定をサポートします。 このユーティリティーは、エンドポイントおよびコンポーネントレベルで記述し、設定する必要のあるコンポーネント固有のコードの量を大幅に削減します。 以下の例は、HTTP4 コンポーネントでユーティリティーを使用する方法を示しています。

このコンポーネントで使用される Apache HTTP クライアントのバージョンは、グローバルの「プロトコル」レジストリーから SSL/TLS 情報を解決します。 このコンポーネントは、Camel JSSE 設定ユーティリティーの使用をサポートするために、HTTP クライアントのプロトコルソケットファクトリーの実装 org.apache.camel.component.http.SSLContextParametersSecureProtocolSocketFactory を提供します。 以下の例は、プロトコルレジストリーを設定し、登録されたプロトコル情報をルートで使用する方法を示しています。

```
KeyStoreParameters ksp = new KeyStoreParameters();
```

```
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

ProtocolSocketFactory factory =
   new SSLContextParametersSecureProtocolSocketFactory(scp);

Protocol.registerProtocol("https",
      new Protocol(
      "https",
      factory,
      443));

from("direct:start")
      .to("https://mail.google.com/mail/").to("mock:results");
```

Apache HTTP クライアントを直接設定

基本的に、camel-http コンポーネントは Apache HTTP クライアントの上部に構築され、カスタム org.apache.camel.component.http.HttpClientConfigurer を実装して、完全な制御が必要な場合に http クライアントでいくつかの設定を行うことができます。

ただし、キーストアとトラストストアのみ を指定する場合は、Apache HTTP HttpClientConfigurer でこれを行うことができます。以下に例を示します。

```
Protocol authhttps = new Protocol("https", new AuthSSLProtocolSocketFactory(
  new URL("file:my.keystore"), "mypassword",
  new URL("file:my.truststore"), "mypassword"), 443);

Protocol.registerProtocol("https", authhttps);
```

次に、HttpClientConfigurer を実装するクラスを作成し、上記の例ごとにキーストアまたはトラストストアを提供する https プロトコルを登録する必要があります。その後、Camel ルートビルダークラスから、以下のようにフックすることができます。

```
HttpComponent httpComponent = getContext().getComponent("http", HttpComponent.class);
httpComponent.setHttpClientConfigurer(new MyHttpClientConfigurer());
```

Spring DSL を使用してこれを行う場合は、URI を使用して HttpClientConfigurer を指定できます。以下に例を示します。

```
<bean id="myHttpClientConfigurer"
 class="my.https.HttpClientConfigurer">
</bean>

<to uri="https://myhostname.com:443/myURL?
httpClientConfigurerRef=myHttpClientConfigurer"/>
```

HttpClientConfigurer を実装し、上記のようにキーストアとトラストストアを設定する限り、問題なく動作します。

141.18. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- Jetty

第142章 HTTP4 コンポーネント

Camel バージョン 2.3 の時点で利用可能

http4: コンポーネントは、外部 HTTP リソースを呼び出すための HTTP ベースのエンドポイントを提供します（HTTP を使用して外部サーバーを呼び出すクライアントとして）。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-http4</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

camel-http4 vs camel-http

camel-http4 は Apache HttpClient 4.x を使用し、camel-http は Apache HttpClient 3.x を使用します。

142.1. URI 形式

```
http4:hostname[:port][/resourceUri][?options]
```

デフォルトでは、HTTP にはポート 80 を使用し、HTTPS には 443 を使用します。

URI にクエリーオプションを追加するには、?option=value&option=value&…

camel-http4 vs camel-jetty

HTTP4 コンポーネントで生成されるエンドポイントにのみ生成できます。そのため、Camel ルートへの入力として使用することはできません。HTTP サーバー経由で HTTP エンドポイントを Camel ルートへの入力としてバインド/公開するには、代わりに Jetty コンポーネント を使用します。

142.2. HTTP4 コンポーネントのオプション

HTTP4 コンポーネントは、以下に示す 18 個のオプションをサポートします。

Name	説明	デフォルト	Type
httpClientConfigurer (advanced)	カスタム HttpClientConfigurer を使用して使用される HttpClient の設定を行うには、以下を行います。		HttpClientConfigurer
clientConnectionManager (advanced)	カスタムおよび共有 HttpClientConnectionManager を使用して接続を管理します。これが設定されている場合、これは常にこのコンポーネントによって作成されるすべてのエンドポイントに使用されます。		HttpClientConnection Manager
httpContext (advanced)	リクエストの実行時にカスタムの org.apache.http.protocol.HttpContext を使用します。		HttpContext
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定します。重要： HttpComponent ごとにサポートされるのは org.apache.camel.util.jsse.SSLContextParameters の 1 つのインスタンスのみです。2 つ以上のインスタンスを使用する必要がある場合は、必要なインスタンスごとに新しい HttpComponent を定義する必要があります。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
x509HostnameVerifier （セキュリティー）	DefaultHostnameVerifier または org.apache.http.conn.ssl.NoopHostnameVerifier などのカスタム X509HostnameVerifier を使用するには、以下を実行します。		HostnameVerifier
maxTotalConnections (advanced)	最大接続数。	200	int
connectionsPerRoute (advanced)	ルートごとの最大接続数。	20	int
connectionTimeToLive (advanced)	接続の有効期間中、時間単位はミリ秒で、常にデフォルト値を維持します。		Long

Name	説明	デフォルト	Type
cookieStore (producer)	カスタムの org.apache.http.client.CookieStore を使用します。デフォルトでは、インメモリーのみのクッキーストアである org.apache.http.impl.client.BasicCookieStore が使用されます。bridgeEndpoint=true の場合、Cookie ストアは noop cookie ストアに強制されます。これは Cookie がブリッジのみであるため（プロキシーとして機能するなど）、これを保存しません。		CookieStore
connectionRequest Timeout (timeout)	接続マネージャーから接続を要求する際に使用されるタイムアウト（ミリ秒単位）。タイムアウト値 0 は無限のタイムアウトとして解釈されます。タイムアウト値 0 は無限のタイムアウトとして解釈されます。負の値は undefined（システムのデフォルト）として解釈されます。デフォルト： コード -1	-1	int
connectTimeout (timeout)	接続が確立されるまでのタイムアウトをミリ秒単位で指定します。タイムアウト値 0 は無限のタイムアウトとして解釈されます。タイムアウト値 0 は無限のタイムアウトとして解釈されます。負の値は undefined（システムのデフォルト）として解釈されます。デフォルト： コード -1	-1	int
socketTimeout (timeout)	ソケットタイムアウト(SO_TIMEOUT)をミリ秒単位で定義します。これは、データの待機のタイムアウトや、2 つの連続したデータパケット間の最大非アクティブ期間になります。タイムアウト値 0 は無限のタイムアウトとして解釈されます。負の値は undefined（システムのデフォルト）として解釈されます。デフォルト： コード -1	-1	int
httpBinding (advanced)	カスタムの HttpBinding を使用して Camel メッセージと HttpClient 間のマッピングを制御する場合。		HttpBinding
httpConfiguration (advanced)	共有 HttpConfiguration をベース設定として使用します。		HttpConfiguration
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用する場合に java のシリアライズを許可するかどうか。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean

Name	説明	デフォルト	Type
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

HTTP4 エンドポイントは、URI 構文を使用して設定します。

```
http4:httpUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

142.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
httpUri	**必須**。呼び出す HTTP エンドポイントの URL。		URI

142.2.2. クエリーパラメーター（48 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
disableStreamCache (common)	Servlet からの raw 入力ストリームがキャッシュされているかどうかを決定します（Camel はストリームをメモリー内/オーバーフローでファイル、ストリームキャッシング）キャッシュに読み取ります。デフォルトでは、Camel は Servlet 入力ストリームをキャッシュして、複数回ロードし、Camel がストリームからすべてのデータを取得できるようにします。ただし、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。たとえば、ファイルまたは他の永続ストアに直接ストリーミングする場合などに、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。DefaultHttpBinding は、ストリームの読み取りを複数回サポートするために、このオプションが false の場合は、リクエスト入力ストリームをストリームキャッシュにコピーし、メッセージボディーに配置します。Servlet を使用してエンドポイントをブリッジ/プロキシーする場合、このオプションを有効にしてパフォーマンスを向上することを検討してください。メッセージペイロードを複数回読み取る必要がない場合は、このオプションを有効にします。http/http4 プロデューサーは、デフォルトでは応答本体ストリームをキャッシュします。このオプションを true に設定すると、プロデューサーは応答ボディーストリームをキャッシュしませんが、応答ストリームをメッセージボディーとして使用します。	false	boolean
headerFilterStrategy (common)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
httpBinding (common)	カスタムの HttpBinding を使用して Camel メッセージと HttpClient 間のマッピングを制御する場合。		HttpBinding
authenticationPreemptive (producer)	このオプションが true の場合、camel-http4 はプリエンプションの Basic 認証をサーバーに送信します。	false	boolean
bridgeEndpoint (producer)	オプションが true の場合、HttpProducer は Exchange.HTTP_URI ヘッダーを無視し、リクエストにエンドポイントの URI を使用します。また、ththExceptionOnFailure オプションを false に設定して、HttpProducer がすべての障害応答を返せるようにすることもできます。	false	boolean
チャンク （プロデューサー）	このオプションが false の場合、サーブレットは HTTP ストリーミングを無効にし、応答に content-length ヘッダーを設定します。	true	boolean

Name	説明	デフォルト	Type
clearExpiredCookies (producer)	HTTP 要求を送信する前に期限切れの cookie を消去するかどうか。これにより、Cookie ストアは、有効期限が切れると新しい Cookie を追加することで、増大し続けます。	true	boolean
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを指定します。デフォルトでは connectionClose は false です。	false	boolean
cookieStore (producer)	カスタムの CookieStore を使用します。デフォルトでは、インメモリーのみのクッキーストアである BasicCookieStore が使用されます。bridgeEndpoint=true の場合、Cookie ストアは noop cookie ストアに強制されます。これは Cookie がブリッジのみであるため（プロキシーとして機能するなど）、これを保存しません。cookieHandler が設定される場合、Cookie ストアは cookie 処理として noop cookie ストアに強制されます。CookieHandler によって実行されます。		CookieStore
copyHeaders (producer)	このオプションが true の場合、コピーストラテジーに従って IN エクスチェンジヘッダーが OUT エクスチェンジヘッダーにコピーされます。これを false に設定し、HTTP 応答からのヘッダーのみを含めることができます（IN ヘッダーは伝播しません）。	true	boolean
deleteWithBody (producer)	HTTP DELETE にメッセージボディーを含めるかどうか。デフォルトでは、HTTP DELETE には HTTP メッセージは含まれません。ただし、まれに、ユーザーにメッセージボディーを含める必要があることがあります。	false	boolean
httpMethod (producer)	使用する HTTP メソッドを設定します。HttpMethod ヘッダーが設定されている場合は、このオプションをオーバーライドできません。		HttpMethods
ignoreResponseBody (producer)	このオプションが true の場合、http プロデューサーは応答本体を読み取りせず、入力ストリームをキャッシュします。	false	boolean
preserveHostHeader (producer)	オプションが true の場合、HttpProducer は Host ヘッダーを現在のエクスチェンジ Host ヘッダーに含まれる値に設定します。これは、ダウンストリームサーバーが受け取る Host ヘッダーを使用してアップストリームクライアントが呼び出した URL を反映させたいリバースプロキシーアプリケーションで有用です。これにより、Host ヘッダーを使用するアプリケーションがプロキシーされるサービスの正確な URL を生成することができます。	false	boolean

Name	説明	デフォルト	Type
throwExceptionOnFailure (producer)	リモートサーバーからの応答に失敗した場合に HttpOperationFailedException のスローを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
transferException (producer)	有効で Exchange がコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object のコンテンツタイプとして応答でシリアライズされたかどうか。プロデューサー側では、例外は HttpOperationFailedException ではなくデシリアライズされ、そのままスローされます。原因となる例外はシリアライズされている必要があります。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
cookieHandler (producer)	HTTP セッションを維持するためのクッキーハンドラーの設定		CookieHandler
okStatusCodeRange (producer)	正常な応答とみなされるステータスコード。値は含まれます。コンマで区切られた複数の範囲を定義できます（例： 200-204,209,301-304）。各範囲は、1 つの数字またはダッシュを含む from から でなければなりません。	200-299	文字列
urlRewrite (producer)	**非推奨** の org.apache.camel.component.http.UrlRewrite への参照。ブリッジ/プロキシーエンドポイントの実行時に URL を書き換えることができるようになりました。詳細は、http://camel.apache.org/urlrewrite.htmlを参照してください。		UrlRewrite
clientBuilder (advanced)	このエンドポイントのプロデューサーまたはコンシューマーによって使用される新しい RequestConfig インスタンスで使用される http クライアント要求パラメーターへのアクセスを提供します。		HttpClientBuilder
clientConnectionManager (advanced)	カスタムの HttpClientConnectionManager を使用して接続を管理するには		HttpClientConnection Manager
connectionsPerRoute (advanced)	ルートごとの最大接続数。	20	int

Name	説明	デフォルト	Type
httpClient (advanced)	プロデューサーによって使用されるカスタム HttpClient を設定します。		HttpClient
httpClientConfigurer (advanced)	認証メカニズムの設定など、プロデューサーやコンシューマーによって作成された新しい HttpClient インスタンスのカスタム設定ストラテジーを登録します。		HttpClientConfigurer
httpClientOptions (advanced)	Map のキー/値を使用して HttpClient を設定します。		マップ
httpContext (advanced)	カスタム HttpContext インスタンスの使用		HttpContext
mapHttpMessageBody (advanced)	このオプションが true の場合、エクスチェンジの IN エクスチェンジボディーは HTTP ボディーにマッピングされます。false に設定すると HTTP マッピングが回避されます。	true	boolean
mapHttpMessageFormUrlEncodedBody (advanced)	このオプションが true の場合、エクスチェンジの IN Exchange Form Encoded body が HTTP にマッピングされます。これを false に設定すると、HTTP Form Encoded body マッピングを回避します。	true	boolean
mapHttpMessageHeaders (advanced)	このオプションが true の場合、エクスチェンジの IN エクスチェンジヘッダーは HTTP ヘッダーにマッピングされます。false に設定すると、HTTP ヘッダーのマッピングが回避されます。	true	boolean
maxTotalConnections (advanced)	最大接続数。	200	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
useSystemProperties (advanced)	設定のフォールバックとしてシステムプロパティーを使用する	false	boolean
proxyAuthDomain (proxy)	NTML で使用するプロキシー認証ドメイン		文字列
proxyAuthHost (proxy)	プロキシー認証ホスト		文字列
proxyAuthMethod (proxy)	使用するプロキシー認証方法		文字列

Name	説明	デフォルト	Type
proxyAuthPassword (proxy)	プロキシー認証パスワード		文字列
proxyAuthPort (proxy)	プロキシー認証ポート		int
proxyAuthScheme (proxy)	使用するプロキシー認証スキーム		文字列
proxyAuthUsername (proxy)	プロキシー認証ユーザー名		文字列
proxyHost (proxy)	使用するプロキシーホスト名		文字列
proxyPort (proxy)	使用するプロキシーポート		int
authDomain (security)	NTML で使用する認証ドメイン		文字列
authHost (security)	NTML で使用する認証ホスト		文字列
authmethod （セキュリティー）	Basic、Digest、または NTLM の値のコンマ区切りリストとして使用できる認証方法。		文字列
authMethodPriority (security)	Basic、Digest、または NTLM のいずれかの使用を優先する認証方法。		文字列
authPassword (security)	認証パスワード		文字列
authUsername (security)	認証ユーザー名		文字列
x509HostnameVerifier （セキュリティー）	DefaultHostnameVerifier または org.apache.http.conn.ssl.NoopHostnameVerifier などのカスタム X509HostnameVerifier を使用するには、以下を実行します。		HostnameVerifier

142.3. メッセージヘッダー

Name	タイプ	説明
Exchange.HTTP_URI	**文字列**	呼び出す URI。エンドポイントに直接設定された既存の URI を上書きします。この URI は、呼び出す http サーバーの URI です。これは、セキュリティーなどのエンドポイントオプションを設定する Camel エンドポイント URI と同じです。このヘッダーは、http サーバーの URI のみをサポートしません。
Exchange.HTTP_PATH	**文字列**	リクエスト URI のパス。ヘッダーは、リクエスト URI を HTTP_URI でビルドするために使用されます。
Exchange.HTTP_QUERY	**文字列**	URI パラメーターエンドポイントに直接設定された既存の URI パラメーターを上書きします。
Exchange.HTTP_RESPONSE_CODE	**int**	外部サーバーからの HTTP 応答コード。OK の場合は 200 です。
Exchange.HTTP_RESPONSE_TEXT	**文字列**	外部サーバーからの HTTP 応答テキスト。
Exchange.HTTP_CHARACTER_ENCODING	**文字列**	文字エンコーディング。
Exchange.CONTENT_TYPE	**文字列**	HTTP コンテンツタイプ。**テキスト/html** などのコンテンツタイプを提供するために IN メッセージと OUT メッセージの両方に設定されます。

Name	タイプ	説明
Exchange.CONTENT_ENCODING	**文字列**	HTTP コンテンツのエンコーディング。**gzip** などのコンテンツエンコーディングを提供するために IN メッセージと OUT メッセージの両方に設定されます。

142.4. メッセージボディー

Camel は外部サーバーからの HTTP 応答を OUT ボディーに保存します。IN メッセージからのヘッダーはすべて OUT メッセージにコピーされるため、ヘッダーはルーティング時に保持されます。さらに Camel は HTTP 応答ヘッダーと OUT メッセージヘッダーを追加します。

142.5. システムプロパティーの使用

useSystemProperties を true に設定すると、HTTP クライアントは以下のシステムプロパティーを検索し、それを使用します。

- *ssl.TrustManagerFactory.algorithm*
- *javax.net.ssl.trustStoreType*
- *javax.net.ssl.trustStore*
- *javax.net.ssl.trustStoreProvider*
- *javax.net.ssl.trustStorePassword*
- *java.home*

- *ssl.KeyManagerFactory.algorithm*
- *javax.net.ssl.keyStoreType*
- *javax.net.ssl.keyStore*
- *javax.net.ssl.keyStoreProvider*
- *javax.net.ssl.keyStorePassword*
- *http.proxyHost*
- *http.proxyPort*
- *http.nonProxyHosts*
- *http.keepAlive*
- *http.maxConnections*

142.6. レスポンスコード

Camel は HTTP レスポンスコードに従って処理されます。

- *応答コードは 100..299 の範囲にあり、Camel は成功レスポンスとして認識します。*
- *応答コードは 300..399 の範囲にあり、Camel はこれをリダイレクト応答として認識し、情報を使用して HttpOperationFailedException を発生させます。*
-

応答コードは 400 以上で、Camel はこれを外部サーバーの失敗として認識し、情報を使用して HttpOperationFailedException を発生させます。

throwExceptionOnFailure The option, throwExceptionOnFailure: オプションを false に設定して、HttpOperationFailedException が失敗したレスポンスコードに対してスローされないようにします。これにより、リモートサーバーから応答を取得できます。
以下に、これを説明する例を示します。

142.7. HTTPOPERATIONFAILEDEXCEPTION

この例外には、以下の情報が含まれます。

- HTTP ステータスコード
- HTTP ステータス行（ステータスコードのテキスト）
- リダイレクトを返した場合のリダイレクト場所のリダイレクト
- サーバーがボディーを応答として提供した場合、応答本体は java.lang.String として応答します。

142.8. 使用される HTTP メソッド

以下のアルゴリズムを使用して、どの HTTP メソッドを使用するかを決定します。
1エンドポイント設定として提供されるメソッド(httpMethod)を使用します。
2.ヘッダーで提供されるメソッド(Exchange.HTTP_METHOD)を使用します。
3.クエリー文字列がヘッダーで提供される場合の GET。
4.エンドポイントがクエリー文字列で設定されている場合の GET。
5.送信するデータがある場合（ボディーは nullではありません）。
6.そうでない場合は GET。

142.9. HTTPSERVLETREQUEST および HTTPSERVLETRESPONSE へのアクセス方法

Camel 型コンバーターシステムを使用して、これらの 2 つのアクセスを取得することができます。
camel-jetty または camel-cxf エンドポイントの後にのみプロセッサーからリクエストおよび応答を取得できます。

```
HttpServletRequest request = exchange.getIn().getBody(HttpServletRequest.class);
HttpServletRequest response = exchange.getIn().getBody(HttpServletResponse.class);
```

142.10. 呼び出し用の URI の設定

HTTP プロデューサーの URI はエンドポイント URI を直接形成できます。以下のルートでは、Camel は HTTP を使用して外部サーバー（ 古いホスト ）に呼び出します。

```
from("direct:start")
        .to("http4://oldhost");
```

同等の Spring サンプルは次のとおりです。

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <to uri="http4://oldhost"/>
  </route>
</camelContext>
```

HTTP エンドポイント URI を上書きするには、キー Exchange.HTTP_URI でヘッダーをメッセージに追加します。

```
from("direct:start")
  .setHeader(Exchange.HTTP_URI, constant("http://newhost"))
  .to("http4://oldhost");
```

上記の例では、エンドポイントは http4://oldhost で設定されていても、Camel は http://newhost を呼び出します。
http4 エンドポイントがブリッジモードで動作している場合、Exchange.HTTP_URI のメッセージヘッダーを無視します。

142.11. URI パラメーターの設定

http プロデューサーは、HTTP サーバーに送信される URI パラメーターをサポートします。URI パラメーターはエンドポイント URI に直接設定することも、メッセージの Exchange.HTTP_QUERY キーのあるヘッダーとして設定できます。

```
from("direct:start")
  .to("http4://oldhost?order=123&detail=short");
```

ヘッダーで提供されるオプション：

```
from("direct:start")
  .setHeader(Exchange.HTTP_QUERY, constant("order=123&detail=short"))
  .to("http4://oldhost");
```

142.12. HTTP メソッド(GET/PATCH/POST/PUT/DELETE/HEAD/OPTIONS/TRACE)を HTTP プロデューサーに設定する方法

http PATCH メソッドの使用

http PATCH メソッドは Camel 2.11.3 / 2.12.1. 以降でサポートされます。

HTTP4 コンポーネントは、メッセージヘッダーを設定して HTTP リクエストメソッドを設定する方法を提供します。以下は例です。

```
from("direct:start")
  .setHeader(Exchange.HTTP_METHOD,
constant(org.apache.camel.component.http4.HttpMethods.POST))
  .to("http4://www.google.com")
  .to("mock:results");
```

この方法は、文字列定数を使用してビットをより短くすることができます。

```
.setHeader("CamelHttpMethod", constant("POST"))
```

同等の Spring サンプルは次のとおりです。

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
    <setHeader headerName="CamelHttpMethod">
        <constant>POST</constant>
    </setHeader>
    <to uri="http4://www.google.com"/>
    <to uri="mock:results"/>
  </route>
</camelContext>
```

142.13. クライアントタイムアウトの使用 - SO_TIMEOUT

HttpSOTimeoutTest ユニットテストを参照してください。

Camel 2.13.0 以降: 更新された HttpSOTimeoutTest ユニットテストを参照してください。

142.14. プロキシーの設定

HTTP4 コンポーネントは、プロキシーを設定する方法を提供します。

```
from("direct:start")
  .to("http4://oldhost?proxyAuthHost=www.myproxy.com&proxyAuthPort=80");
```

proxyAuthUsername および proxyAuthPassword オプションを使用したプロキシー認証もサポートします。

142.14.1. URI 外でのプロキシー設定の使用

システムプロパティーの競合を回避するには、CamelContext または URI からのみプロキシー設定を設定できます。
Java DSL の場合

```
context.getProperties().put("http.proxyHost", "172.168.18.9");
context.getProperties().put("http.proxyPort" "8080");
```

Spring XML

```
<camelContext>
   <properties>
      <property key="http.proxyHost" value="172.168.18.9"/>
      <property key="http.proxyPort" value="8080"/>
  </properties>
</camelContext>
```

Camel は、最初に Java System または CamelContext プロパティーからの設定を設定し、指定されている場合はエンドポイントプロキシーオプションを設定します。
そのため、エンドポイントオプションでシステムプロパティーを上書きできます。

Camel 2.8 では、http.proxyScheme プロパティーもあり、使用するスキームを明示的に設定できます。

142.15. 文字セットの設定

POST を使用してデータを送信する場合は、Exchange プロパティーを使用し て文字セット を設定できます。

```
exchange.setProperty(Exchange.CHARSET_NAME, "ISO-8859-1");
```

142.15.1. スケジュールされたポーリングの例

この例では、Google ホームページを 10 秒ごとにポーリングし、ページをファイル message.html に書き込みます。

```
from("timer://foo?fixedRate=true&delay=0&period=10000")
  .to("http4://www.google.com")
  .setHeader(FileComponent.HEADER_FILE_NAME, "message.html")
  .to("file:target/google");
```

142.15.2. エンドポイント URI からの URI パラメーター

この例では、Web ブラウザーに入力した内容のみの完全な URI エンドポイントがあります。Web ブラウザーの場合と同様に、複数の URI パラメーターをセパレーターとして使用し、区切り文字として使用できます。Camel はここでは複雑ではありません。

```
// we query for Camel at the Google page
template.sendBody("http4://www.google.com/search?q=Camel", null);
```

142.15.3. メッセージの URI パラメーター

```
Map headers = new HashMap();
headers.put(Exchange.HTTP_QUERY, "q=Camel&lr=lang_en");
// we query for Camel and English language at Google
template.sendBody("http4://www.google.com/search", null, headers);
```

上記のヘッダー値では、先頭に ? を付け ず、パラメーターは通常 &amp ; char で区切ることができることに注意してください。

142.15.4. 応答コードの取得

HTTP4 コンポーネントから HTTP 応答コードを取得するには、Exchange.HTTP_RESPONSE_CODE の Out メッセージヘッダーから値を取得します。

```
Exchange exchange = template.send("http4://www.google.com/search", new Processor() {
  public void process(Exchange exchange) throws Exception {
    exchange.getIn().setHeader(Exchange.HTTP_QUERY, constant("hl=en&q=activemq"));
  }
});
Message out = exchange.getOut();
int responseCode = out.getHeader(Exchange.HTTP_RESPONSE_CODE, Integer.class);
```

142.16. クッキーの無効化

Cookie を無効にするには、この URI オプション httpClient.cookiePolicy=ignoreCookiesを追加して HTTP クライアントがクッキーを無視するように設定できます。 httpClient.cookiePolicy=ignoreCookies

142.17. 高度な使用方法

HTTP プロデューサーをより詳細に制御する必要がある場合は、さまざまなクラスを設定してカスタム動作を提供する HttpComponent を使用する必要があります。

142.17.1. HTTP クライアントの SSL 設定

JSSE 設定ユーティリティーの使用

Camel 2.8 より、HTTP4 コンポーネントは Camel JSSE 設定ユーティリティーを介して SSL/TLS 設定をサポートします。 このユーティリティーは、エンドポイントおよびコンポーネントレベルで記述し、設定する必要のあるコンポーネント固有のコードの量を大幅に削減します。 以下の例は、HTTP4 コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);
```

```
HttpComponent httpComponent = getContext().getComponent("https4",
HttpComponent.class);
httpComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
    <camel:keyManagers
       keyPassword="keyPassword">
      <camel:keyStore
         resource="/users/home/server/keystore.jks"
         password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="https4://127.0.0.1/mail/?sslContextParameters=#sslContextParameters"/>...
```

Apache HTTP クライアントを直接設定

基本的には、camel-http4 コンポーネントは Apache HttpClient の上部に構築されます。詳細は、SSL/TLS のカスタマイズ を参照するか、org.apache.camel.component.http4.HttpsServerTestSupport ユニットテストベースクラスを調べます。
また、カスタム org.apache.camel.component.http4.HttpClientConfigurer を実装して、完全な制御が必要な場合に http クライアントで設定を行うことができます。

ただし、キーストアとトラストストアのみ を指定する場合は、Apache HTTP HttpClientConfigurer でこれを行うことができます。以下に例を示します。

```
KeyStore keystore = ...;
KeyStore truststore = ...;

SchemeRegistry registry = new SchemeRegistry();
registry.register(new Scheme("https", 443, new SSLSocketFactory(keystore, "mypassword",
truststore)));
```

次に、HttpClientConfigurer を実装するクラスを作成し、上記の例ごとにキーストアまたはトラストストアを提供する https プロトコルを登録する必要があります。その後、Camel ルートビルダークラスから、以下のようにフックすることができます。

```
HttpComponent httpComponent = getContext().getComponent("http4", HttpComponent.class);
httpComponent.setHttpClientConfigurer(new MyHttpClientConfigurer());
```

Spring DSL を使用してこれを行う場合は、URI を使用して HttpClientConfigurer を指定できます。以下に例を示します。

```
<bean id="myHttpClientConfigurer"
 class="my.https.HttpClientConfigurer">
</bean>

<to uri="https4://myhostname.com:443/myURL?httpClientConfigurer=myHttpClientConfigurer"/>
```

HttpClientConfigurer を実装し、上記のようにキーストアとトラストストアを設定する限り、問題なく動作します。

HTTPS を使用した gotchas の認証

エンドユーザーは、HTTPS での認証に問題があると報告されました。この問題は最終的に、カスタムの設定された org.apache.http.protocol.HttpContext を指定して解決されました。

- 1.HttpContexts に(Spring)ファクトリーを作成します。

```
public class HttpContextFactory {

  private String httpHost = "localhost";
  private String httpPort = 9001;

  private BasicHttpContext httpContext = new BasicHttpContext();
  private BasicAuthCache authCache = new BasicAuthCache();
  private BasicScheme basicAuth = new BasicScheme();

  public HttpContext getObject() {
    authCache.put(new HttpHost(httpHost, httpPort), basicAuth);

    httpContext.setAttribute(ClientContext.AUTH_CACHE, authCache);

    return httpContext;
  }

  // getter and setter
}
```

- 2.Spring アプリケーションコンテキストファイルで HttpContext を宣言します。

```
<bean id="myHttpContext" factory-bean="httpContextFactory" factory-method="getObject"/>
```

- *3.http4 URL のコンテキストを参照します。*

```
<to uri="https4://myhostname.com:443/myURL?httpContext=myHttpContext"/>
```

異なる SSLContextParameters の使用

HTTP4 コンポーネントは、コンポーネントごとに org.apache.camel.util.jsse.SSLContextParameters の 1 つのインスタンスのみをサポートします。2 つ以上の異なるインスタンスを使用する必要がある場合は、以下のように複数の HTTP4 コンポーネントを設定する必要があります。2 つのコンポーネントがあり、それぞれ sslContextParameters プロパティーの独自のインスタンスを使用します。

```
<bean id="http4-foo" class="org.apache.camel.component.http4.HttpComponent">
  <property name="sslContextParameters" ref="sslContextParams1"/>
  <property name="x509HostnameVerifier" ref="hostnameVerifier"/>
</bean>

<bean id="http4-bar" class="org.apache.camel.component.http4.HttpComponent">
  <property name="sslContextParameters" ref="sslContextParams2"/>
  <property name="x509HostnameVerifier" ref="hostnameVerifier"/>
</bean>
```

第143章 HYSTRIX コンポーネント

Camel バージョン 2.18 から利用可能

hystrix コンポーネントは、Camel ルートで Netflix Hystrix サーキットブレーカーを統合します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hystrix</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

詳細は「Hystrix EIP」を参照してください。

第144章 ICAL DATAFORMAT

Camel バージョン 2.12 から利用可能

ICal dataformat は、iCalendar メッセージの使用に使用されます。

一般的な iCalendar メッセージは以下のようになります。

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Events Calendar//iCal4j 1.0//EN
CALSCALE:GREGORIAN
BEGIN:VEVENT
DTSTAMP:20130324T180000Z
DTSTART:20130401T170000
DTEND:20130401T210000
SUMMARY:Progress Meeting
TZID:America/New_York
UID:00000000
ATTENDEE;ROLE=REQ-PARTICIPANT;CN=Developer 1:mailto:dev1@mycompany.com
ATTENDEE;ROLE=OPT-PARTICIPANT;CN=Developer 2:mailto:dev2@mycompany.com
END:VEVENT
END:VCALENDAR
```

144.1. オプション

iCal dataformat は、以下に示す 2 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
validating	**false**	**ブール値**	検証するかどうか。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

144.2. 基本的な使用方法

上記のメッセージをアンマーシャリングおよびマーシャリングする場合、ルートは以下のようになり

ます。

```
from("direct:ical-unmarshal")
  .unmarshal("ical")
  .to("mock:unmarshaled")
  .marshal("ical")
  .to("mock:marshaled");
```

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-ical</artifactId>
    <version>x.x.x</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

144.3. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第145章 IEC 60870 CLIENT COMPONENT

Camel バージョン 2.20 で利用可能

IEC 60870-5-104 Client コンポーネントは、Eclipse NeoSCADA™ 実装を使用して IEC 60870 サーバーへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-iec60870</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

IEC 60870 Client コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
DefaultConnection Options (common)	デフォルトの接続オプション		ClientOptions
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

145.1. URI 形式

エンドポイントの URI 構文は以下のようになります。

```
iec60870-client:host:port/00-01-02-03-04
```

情報オブジェクトのアドレスは、上記の構文のパスにエンコードされます。常に 5 つのオクテットアドレス形式が使用されていることに注意してください。未使用の octets にはゼロで入力する必要があります。

145.2. URI オプション

IEC 60870 クライアントエンドポイントは、URI 構文を使用して設定します。

```
iec60870-client:uriPath
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

145.2.1. パスパラメーター（1 パラメーター）：

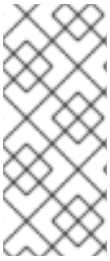
Name	説明	デフォルト	Type
uriPath	**必須**。オブジェクト情報アドレス		ObjectAddress

145.2.2. クエリーパラメーター（18 パラメーター）：

Name	説明	デフォルト	Type
dataModuleOptions (common)	データモジュールオプション		DataModuleOptions
protocolOptions (common)	プロトコルオプション		ProtocolOptions
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
acknowledgeWindow (connection)	パラメーター W - 承認ウィンドウ。	10	short
adsuAddressType (connection)	一般的な ASDU アドレスサイズ。SIZE_1 または SIZE_2 のいずれかを使用できます。		ASDUAddressType
causeOfTransmissionType (connection)	送信タイプの原因。SIZE_1 または SIZE_2 のいずれかを使用できます。		CauseOfTransmission Type
informationObjectAddress Type (connection)	情報アドレスサイズ。SIZE_1、SIZE_2、SIZE_3 のいずれかを使用できます。		InformationObjectAddressType
maxUnacknowledged (connection)	パラメーター K: 承認されていないメッセージの最大数。	15	short
timeout1 (connection)	T1 のタイムアウト（ミリ秒単位）。	15000	int
timeout2 （接続）	T2 のタイムアウト（ミリ秒単位）。	10000	int
timeout3 (connection)	T3 のタイムアウト（ミリ秒単位）。	20000	int
ignoreBackgroundScan (data)	バックグラウンドスキャン送信を無視するかどうか。	true	boolean
ignoreDaylightSavingTime (data)	DST を無視するか、または考慮するかどうか。	false	boolean
タイムゾーン（データ）	使用するタイムゾーン。Java タイムゾーン文字列を指定できます。	UTC	TimeZone
connectionId (id)	接続インスタンスをグループ化する識別子		文字列

URI のホストおよびポート部分で識別され、「id」グループのすべてのパラメーターで特定された場合には接続インスタンス。新しい接続 ID が検出されると、接続オプションが評価され、そのオプションを指定して接続インスタンスが作成されます。



注記

2 つの URI が同じ接続（ホスト、ポートなど）を指定するものの、異なる接続オプションを指定する場合、これらの接続オプションのどれを使用するかは未定義になります。

最後の接続オプションは、以下の順序で評価されます。

- *存在する場合は、connectionOptions パラメーターが使用されます。*
- *それ以外の場合は、以下の手順で defaultConnectionOptions インスタンスをコピーし、カスタマイズします。*
- *protocolOptions が存在する場合は適用します。*
- *dataModuleOptions が存在する場合は適用*
- *すべての明示的な接続パラメーターを適用します（例： timeZone）。*

第146章 IEC 60870 SERVER COMPONENT

Camel バージョン 2.20 で利用可能

IEC 60870-5-104 Server コンポーネントは、Eclipse NeoSCADA™ 実装を使用して IEC 60870 サーバーへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-iec60870</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

IEC 60870 サーバーコンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
DefaultConnection Options (common)	デフォルトの接続オプション		ServerOptions
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

146.1. URI 形式

エンドポイントの URI 構文は以下のようになります。

```
iec60870-server:host:port/00-01-02-03-04
```

情報オブジェクトのアドレスは、上記の構文のパスにエンコードされます。常に 5 つのオクテットアドレス形式が使用されていることに注意してください。未使用の octets にはゼロで入力する必要があります。

146.2. URI オプション

IEC 60870 サーバーエンドポイントは、URI 構文を使用して設定します。

```
iec60870-server:uriPath
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

146.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
uriPath	**必須**。オブジェクト情報アドレス		ObjectAddress

146.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
dataModuleOptions (common)	データモジュールオプション		DataModuleOptions
filterNonExecute (common)	execute ビットが設定されていないすべてのリクエストを除外します。	true	boolean
protocolOptions (common)	プロトコルオプション		ProtocolOptions
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
acknowledgeWindow (connection)	パラメーター W - 承認ウィンドウ。	10	short
adsuAddressType (connection)	一般的な ASDU アドレスサイズ。SIZE_1 または SIZE_2 のいずれかを使用できます。		ASDUAddressType
causeOfTransmissionType (connection)	送信タイプの原因。SIZE_1 または SIZE_2 のいずれかを使用できます。		CauseOfTransmission Type
informationObjectAddress Type (connection)	情報アドレスサイズ。SIZE_1、SIZE_2、SIZE_3 のいずれかを使用できます。		InformationObjectAddressType
maxUnacknowledged (connection)	パラメーター K: 承認されていないメッセージの最大数。	15	short
timeout1 (connection)	T1 のタイムアウト（ミリ秒単位）。	15000	int
timeout2 （接続）	T2 のタイムアウト（ミリ秒単位）。	10000	int
timeout3 (connection)	T3 のタイムアウト（ミリ秒単位）。	20000	int
ignoreBackgroundScan (data)	バックグラウンドスキャン送信を無視するかどうか。	true	boolean
ignoreDaylightSavingTime (data)	DST を無視するか、または考慮するかどうか。	false	boolean
タイムゾーン （データ）	使用するタイムゾーン。Java タイムゾーン文字列を指定できます。	UTC	TimeZone
connectionId (id)	接続インスタンスをグループ化する識別子		文字列

第147章 IGNITE CACHE コンポーネント

Camel バージョン 2.17 から利用可能

Ignite Cache エンドポイントは camel-ignite エンドポイントの 1 つで、Ignite Cache と対話できます。これは、プロデューサー（Ignite キャッシュでキャッシュ操作を呼び出す）とコンシューマー（継続的なクエリーから変更を消費する）の両方を提供します。

キャッシュ値は常にメッセージのボディーですが、キャッシュキーは常に IgniteConstants.IGNITE_CACHE_KEY メッセージヘッダーに保存されます。

エンドポイント URI で固定操作を設定していても、IgniteConstants.IGNITE_CACHE_OPERATION メッセージヘッダーを設定することで、交換ごとに異なる場合があります。

147.1. オプション

Ignite Cache コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
ignite (common)	Ignite インスタンスを設定します。		ignite
configurationResource (common)	設定の読み込み元となるリソースを設定します。URI、String(URI)、または InputStream を指定できます。		オブジェクト
igniteConfiguration (common)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Cache エンドポイントは、URI 構文を使用して設定します。

```
ignite-cache:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

147.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	***必須***。キャッシュ名。		文字列

147.1.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
propagateIncomingBodyIfNoReturnValue (common)	基礎となる Ignite 操作の戻り値タイプが void の場合には、受信ボディーを伝播するかどうかを設定します。	true	boolean
treatCollectionsAsCache オブジェクト （共通）	コレクションをキャッシュオブジェクトとして処理するか、または挿入/更新/コンピュートなどを行う項目のコレクションとして処理するかどうかを設定します。	false	boolean
autoUnsubscribe (consumer)	継続的なクエリーコンシューマーで自動サブスクライブが有効になっているかどうか。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
fireExistingQueryResults (consumer)	クエリーに一致する既存の結果を処理するかどうか。継続的なクエリーコンシューマーの初期化に使用されます。	false	boolean
oneExchangePerUpdate (consumer)	1 つのバッチで複数の更新が受信された場合でも、個別の Exchange で各更新をパッケージ化するかどうか。継続的なクエリーコンシューマーによってのみ使用されます。	true	boolean
pageSize (consumer)	ページサイズ。継続的なクエリーコンシューマーによってのみ使用されます。	1	int

Name	説明	デフォルト	Type
クエリー（コンシューマー）	実行する Query は、これを必要とする操作および Continuous Query Consumer についてのみ必要です。		Object>>
remoteFilter (consumer)	継続的なクエリーコンシューマーによってのみ使用されるリモートフィルター。		Object>
timeInterval (consumer)	継続的なクエリーコンシューマーの時間間隔。	0	Long
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cachePeekMode (producer)	CachePeekMode は、これを必要とする操作（リンク IgniteCacheOperationSIZE）にのみ必要です。	ALL	CachePeekMode
failIfInexistentCache (producer)	キャッシュが存在しない場合に初期化に失敗するかどうか。	false	boolean
操作（プロデューサー）	呼び出すキャッシュ操作。使用できる値は GET、PUT、REMOVE、SIZE、REBALANCE、QUERY、CLEAR です。		IgniteCacheOperation
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

147.1.3. 使用されるヘッダー

このエンドポイントは、以下のヘッダーを使用します。

ヘッダー名	Constant	想定されるタイプ	説明

ヘッダー名	Constant	想定されるタイプ	説明
CamelIgniteCacheKey	IgniteConstants.IGNITE_CACHE_KEY	文字列	メッセージボディーのエントリー値のキャッシュキー。
CamelIgniteCacheQuery	IgniteConstants.IGNITE_CACHE_QUERY	クエリー	QUERY 操作の呼び出し時に実行するクエリー（プロデューサー）。
CamelIgniteCacheOperation	IgniteConstants.IGNITE_CACHE_OPERATION	IgniteCacheOperation enum	キャッシュ操作を動的に変更して実行する(producer)ことができます。
CamelIgniteCachePeekMode	IgniteConstants.IGNITE_CACHE_PEEK_MODE	CachePeekMode enum	SIZE 操作の実行時にキャッシュ peek モードを動的に変更できます。
CamelIgniteCacheEventType	IgniteConstants.IGNITE_CACHE_EVENT_TYPE	int(EventType constants)	このヘッダーには、継続的なクエリーコンシューマーを使用する場合に受信したイベントタイプが含まれます。
CamelIgniteCacheName	IgniteConstants.IGNITE_CACHE_NAME	文字列	このヘッダーには、継続的なクエリーイベントが受信されたキャッシュ名（コンシューマー）があります。プロデューサー操作を実行するキャッシュを動的に変更することはできません。このために EIP を使用します（受信者リスト、動的ルーターなど）。
CamelIgniteCacheOldValue	IgniteConstants.IGNITE_CACHE_OLD_VALUE	オブジェクト	このヘッダーには、受信キャッシュイベント（コンシューマー）に渡されると古いキャッシュ値が含まれます。

第148章 IGNITE COMPUTE コンポーネント

Camel バージョン 2.17 から利用可能

Ignite Compute エンドポイントは camel-ignite エンドポイントの 1 つで、必要に応じて IgniteCallable、IgniteRunnable、IgniteClosure またはそれらのコレクションを渡して、クラスターでコンピュート操作 を実行できます。

このエンドポイントはプロデューサーのみをサポートします。

エンドポイント URI のホスト部分はシンボリックエンドポイント ID で、これはいずれの目的では使用されません。

エンドポイントは、IN メッセージのボディーに Compute ジョブとして渡されたオブジェクトの実行を試みます。実行タイプによっては、異なるペイロードタイプを想定します。

148.1. オプション

Ignite Compute コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
ignite (producer)	Ignite インスタンスを設定します。		ignite
configurationResource (producer)	設定の読み込み元となるリソースを設定します。URI、String(URI)、または InputStream を指定できます。		オブジェクト
igniteConfiguration (producer)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Compute エンドポイントは、URI 構文を使用して設定します。

ignite-compute:endpointId

以下の path パラメーターおよびクエリーパラメーターを使用します。

148.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
endpointId	**必要な** エンドポイント ID（不使用）。		文字列

148.1.2. クエリーパラメーター（8 パラメーター）：

Name	説明	デフォルト	Type
clusterGroupExpression (producer)	IgniteCompute インスタンスのクラスターグループを返す式。		ClusterGroupExpression
computeName (producer)	リンク IgniteComputewithName(String)で設定されるコンピュートジョブの名前。		文字列
executionType (producer)	実行するコンピュート操作が **必要です**。使用できる値は、CALL、BROADCAST、APPLY、EXECUTE、RUN、AFFINITY_CALL、AFFINITY_RUN です。コンポーネントは、操作に応じて異なるペイロードタイプを想定します。		IgniteComputeExecution Type
propagateIncomingBodyIfNoReturnValue (producer)	基礎となる Ignite 操作の戻り値タイプが void の場合には、受信ボディーを伝播するかどうかを設定します。	true	boolean
taskName (producer)	タスク名。リンク IgniteComputeExecutionTypeEXECUTE 実行タイプを使用している場合にのみ該当します。		文字列
timeoutMillis (producer)	トリガーされたジョブのタイムアウト間隔（ミリ秒単位）。リンク IgniteComputewithTimeout(long)で設定されます。		Long
treatCollectionsAsCache Objects (producer)	コレクションをキャッシュオブジェクトとして処理するか、または挿入/更新/コンピュートなどを行う項目のコレクションとして処理するかどうかを設定します。	false	boolean

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

148.1.3. 予想されるペイロードタイプ

各操作には、指定されたタイプが必要です。

操作	予想されるペイロード
CALL	IgniteCallable または単一の IgniteCallable のコレクション。
BROADCAST	IgniteCallable, IgniteRunnable, IgniteClosure.
APPLY	IgniteClosure.
実行	ComputeTask、Class<? は、taskName オプションが null でない場合にパラメーターを表すオブジェクトを拡張します。
実行	IgniteRunnables または単一の IgniteRunnables のコレクション。
AFFINITY_CALL	IgniteCallable.
AFFINITY_RUN	IgniteRunnable.

148.1.4. 使用されるヘッダー

このエンドポイントは、以下のヘッダーを使用します。

ヘッダー名	Constant	想定されるタイプ	説明
CamelIgniteComputeExecutionType	IgniteConstants.IGNITE_COMPUTE_EXECUTION_TYPE	IgniteComputeExecutionType enum	実行するコンピュート操作を動的に変更できます。

ヘッダー名	Constant	想定されるタイプ	説明
CamelIgniteComputeParameters	IgniteConstants.IGNITE_COMPUTE_PARAMS	オブジェクトのオブジェクトまたはコレクション。	APPLY、BROADCAST、および EXECUTE 操作のパラメーター。
CamelIgniteComputeReducer	IgniteConstants.IGNITE_COMPUTE_REDUCER	IgniteReducer	APPLY および CALL 操作を減らします。
CamelIgniteComputeAffinityCacheName	IgniteConstants.IGNITE_COMPUTE_AFFINITY_CACHE_NAME	文字列	AFFINITY_CALL および AFFINITY_RUN 操作のアフィニティーキャッシュ名。
CamelIgniteComputeAffinityKey	IgniteConstants.IGNITE_COMPUTE_AFFINITY_KEY	オブジェクト	AFFINITY_CALL および AFFINITY_RUN 操作のアフィニティーキー。

第149章 IGNITE EVENTS コンポーネント

Camel バージョン 2.17 から利用可能

Ignite Events エンドポイントは camel-ignite エンドポイントの 1 つで、ローカル イベントリスナーを作成して、Ignite クラスターからイベントを受信 できます。

このエンドポイントはコンシューマーのみをサポートします。このコンシューマーによって作成されたエクスチェンジは、受信したイベントオブジェクトを IN メッセージのボディーに配置します。

149.1. オプション

Ignite Events コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
ignite (consumer)	Ignite インスタンスを設定します。		ignite
configurationResource (consumer)	設定の読み込み元となるリソースを設定します。URI、String(URI)、または InputStream を指定できます。		オブジェクト
igniteConfiguration (consumer)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Events エンドポイントは、URI 構文を使用して設定します。

```
ignite-events:endpointId
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

149.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
endpointId	エンドポイント ID（不使用）。		文字列

149.1.2. クエリーパラメーター（8 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
clusterGroupExpression (consumer)	クラスターグループ式。		ClusterGroupExpression
イベント（コンシューマー）	ID が org.apache.ignite.events.EventType の異なる定数である Set として直接サブスクライブするイベント ID。	EventType.EVTS_ALL	Set<Integer>OrString
propagateIncomingBodyIfNoReturnValue (consumer)	基礎となる Ignite 操作の戻り値タイプが void の場合には、受信ボディーを伝播するかどうかを設定します。	true	boolean
treatCollectionsAsCache Objects (consumer)	コレクションをキャッシュオブジェクトとして処理するか、または挿入/更新/コンピュートなどを行う項目のコレクションとして処理するかどうかを設定します。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

第150章 OGIGNITE ID GENERATOR コンポーネント

Camel バージョン 2.17 から利用可能

Ignite ID Generator エンドポイントは camel-ignite エンドポイントの 1 つで、Ignite Atomic Sequences および ID Generators と対話できます。

このエンドポイントはプロデューサーのみをサポートします。

150.1. オプション

Ignite ID Generator コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
ignite (producer)	Ignite インスタンスを設定します。		ignite
configurationResource (producer)	設定の読み込み元となるリソースを設定します。URI、String(URI)、または InputStream を指定できます。		オブジェクト
igniteConfiguration (producer)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite ID Generator エンドポイントは、URI 構文を使用して設定します。

```
ignite-idgen:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

150.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	***必須***。シーケンス名。		文字列

150.1.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
batchSize (producer)	バッチサイズ。		整数
initialValue (producer)	初期値。	0	Long
操作 （プロデューサー）	Ignite ID Generator で呼び出す操作。IN メッセージの IgniteConstants.IGNITE_IDGEN_OPERATION ヘッダーに置き換えられました。使用できる値は ADD_AND_GET、GET、GET_AND_ADD、GET_AND_INCREMENT、INCREMENT_AND_GET です。		IgniteIdGenOperation
propagateIncomingBodyIfNoReturnValue (producer)	基礎となる Ignite 操作の戻り値タイプが void の場合には、受信ボディーを伝播するかどうかを設定します。	true	boolean
treatCollectionsAsCache Objects (producer)	コレクションをキャッシュオブジェクトとして処理するか、または挿入/更新/コンピュートなどを行う項目のコレクションとして処理するかどうかを設定します。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

第151章 IGNITE MESSAGING COMPONENT

Camel バージョン 2.17 から利用可能

Ignite Messaging エンドポイントは camel-ignite エンドポイントの 1 つで、Ignite トピック からメッセージを送受信できます。

このエンドポイントは、プロデューサー（メッセージの送信）およびコンシューマー（メッセージの受信）をサポートします。

151.1. オプション

Ignite Messaging コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
ignite (common)	Ignite インスタンスを設定します。		ignite
configurationResource (common)	設定の読み込み元となるリソースを設定します。URI、String(URI)、または InputStream を指定できます。		オブジェクト
igniteConfiguration (common)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Messaging エンドポイントは、URI 構文を使用して設定します。

```
ignite-messaging:topic
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

151.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
topic	**必要な** トピック名。		文字列

151.1.2. クエリーパラメーター（9 パラメーター）：

Name	説明	デフォルト	Type
propagateIncomingBodyIfNoReturnValue (common)	基礎となる Ignite 操作の戻り値タイプが void の場合には、受信ボディーを伝播するかどうかを設定します。	true	boolean
treatCollectionsAsCache オブジェクト （共通）	コレクションをキャッシュオブジェクトとして処理するか、または挿入/更新/コンピュートなどを行う項目のコレクションとして処理するかどうかを設定します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
clusterGroupExpression (producer)	クラスターグループ式。		ClusterGroupExpression
sendMode (producer)	使用する送信モード。使用できる値は UNORDERED、ORDERED です。	順序なし	IgniteMessagingSend Mode
タイムアウト （プロデューサー）	順序付けされたメッセージ使用時の送信操作のタイムアウト。		Long

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

151.1.3. 使用されるヘッダー

このエンドポイントは、以下のヘッダーを使用します。

ヘッダー名	Constant	想定されるタイプ	説明
CamelIgniteMessagingTopic	IgniteConstants.IGNITE_MESSAGING_TOPIC	文字列	トピックを動的に変更してメッセージを送信することができます（プロデューサー）。また、メッセージが受信されたトピック（コンシューマー）も行います。
CamelIgniteMessagingUUID	IgniteConstants.IGNITE_MESSAGING_UUID	UUID	このヘッダーには、メッセージが到達するとサブスクリプションの UUID が入力されます（コンシューマー）。

第152章 IGNITE QUEUES COMPONENT

Camel バージョン 2.17 から利用可能

Ignite Queue エンドポイントは camel-ignite エンドポイントの 1 つで、Ignite Queue データ構造と対話できます。

このエンドポイントはプロデューサーのみをサポートします。

152.1. オプション

Ignite Queues コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
ignite (producer)	Ignite インスタンスを設定します。		ignite
configurationResource (producer)	設定の読み込み元となるリソースを設定します。URI、String(URI)、または InputStream を指定できます。		オブジェクト
igniteConfiguration (producer)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Queues エンドポイントは、URI 構文を使用して設定します。

```
ignite-queue:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

152.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	***必須***。キュー名。		文字列

152.1.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
容量 （プロデューサー）	キューの容量。デフォルト： バインドなし。		int
設定 （プロデューサー）	コレクションの設定。デフォルト： 空の設定。また、configuration.xyz=123 オプションを使用して内部プロパティーを設定することもできます。		CollectionConfiguration
操作 （プロデューサー）	Ignite Queue で呼び出す操作。IN メッセージの IgniteConstants.IGNITE_QUEUE_OPERATION ヘッダーに置き換えられました。使用できる値は、CONTAINS、ADD、SIZE、REMOVE、ITERATOR、CLEAR、RETAIN_ALL、ARRAY、DRAIN、ELEMENT、PEEK、OFFER、POLL、TAKE、PUT です。		IgniteQueueOperation
propagateIncomingBodyIfNoReturnValue (producer)	基礎となる Ignite 操作の戻り値タイプが void の場合には、受信ボディーを伝播するかどうかを設定します。	true	boolean
timeoutMillis (producer)	キューのタイムアウト（ミリ秒単位）。デフォルト： タイムアウトなし。		Long
treatCollectionsAsCache Objects (producer)	コレクションをキャッシュオブジェクトとして処理するか、または挿入/更新/コンピュートなどを行う項目のコレクションとして処理するかどうかを設定します。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

152.1.3. 使用されるヘッダー

このエンドポイントは、以下のヘッダーを使用します。

ヘッダー名	Constant	想定されるタイプ	説明
CamelIgniteQueueOperation	IgniteConstants.IGNITE_QUEUE_OPERATION	IgniteQueueOperation enum	キュー操作を動的に変更できます。
CamelIgniteQueueMaxElements	IgniteConstants.IGNITE_QUEUE_MAX_ELEMENTS	整数または int	DRAIN 操作を呼び出す場合、ドレイン（解放）する項目の量。
CamelIgniteQueueTransferredCount	IgniteConstants.IGNITE_QUEUE_TRANSFERRED_COUNT	整数または int	DRAIN 操作の結果として転送される項目の量。
CamelIgniteQueueTimeoutMillis	IgniteConstants.IGNITE_QUEUE_TIMEOUT_MILLIS	long または long	OFFER または POLL 操作の呼び出し時に使用するタイムアウトをミリ秒単位で設定します。

第153章 IGNITE SETS COMPONENT

Camel バージョン 2.17 から利用可能

Ignite Sets エンドポイントは camel-ignite エンドポイントの 1 つで、Ignite Set データ構造と対話できます。

このエンドポイントはプロデューサーのみをサポートします。

153.1. オプション

Ignite Sets コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
ignite (producer)	Ignite インスタンスを設定します。		ignite
configurationResource (producer)	設定の読み込み元となるリソースを設定します。URI、String(URI)、または InputStream を指定できます。		オブジェクト
igniteConfiguration (producer)	ユーザーがプログラムによる IgniteConfiguration を設定できるようにします。		IgniteConfiguration
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Ignite Sets エンドポイントは、URI 構文を使用して設定します。

```
ignite-set:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

153.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	**必須** の名前。		文字列

153.1.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
設定 （プロデューサー）	コレクションの設定。デフォルト： 空の設定。また、configuration.xyz=123 オプションを使用して内部プロパティーを設定することもできます。		CollectionConfiguration
操作 （プロデューサー）	Ignite Set で呼び出す操作。IN メッセージの IgniteConstants.IGNITE_SETS_OPERATION ヘッダーに置き換えられました。使用できる値は、CONTAINS、ADD、SIZE、REMOVE、ITERATOR、CLEAR、RETAIN_ALL、ARRAY です。		IgniteSetOperation
propagateIncomingBodyIfNoReturnValue (producer)	基礎となる Ignite 操作の戻り値タイプが void の場合には、受信ボディーを伝播するかどうかを設定します。	true	boolean
treatCollectionsAsCache Objects (producer)	コレクションをキャッシュオブジェクトとして処理するか、または挿入/更新/コンピュートなどを行う項目のコレクションとして処理するかどうかを設定します。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

153.1.3. 使用されるヘッダー

このエンドポイントは、以下のヘッダーを使用します。

ヘッダー名	Constant	想定されるタイプ	説明
CamelIgniteSetsOperation	IgniteConstants.IGNITE_SETS_OPERATION	IgniteSetOperation enum	set 操作を動的に変更できます。

第154章 INFLUXDB コンポーネント

Camel バージョン 2.18 から利用可能

このコンポーネントでは、時系列データベースである InfluxDB https://influxdata.com/time-series-platform/influxdb/ と対話できます。このコンポーネントのネイティブボディータイプは Point（ネイティブ influxdb クラス）ですが、メッセージボディーとして Map<String, Object> も受け入れることができ、Point.class に変換されます。マップには InfluxDbConstants.MEASUREMENT_NAME をキーとして持つ要素が含まれる必要があることに注意してください。

当然ながら、独自のコンバーターを Data type to Point に登録したり、camel が提供する（アンマーシャリング）マーシャリングツールを使用したりできます。

Camel 2.18 以降の Influxdb には Java 8 が必要です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-influxdb</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

154.1. URI 形式

```
influxdb://beanName?[options]
```

154.2. URI オプション

プロデューサーを使用すると、ネイティブ java ドライバーを使用して、レジストリーに設定された influxdb にメッセージを送信できます。

InfluxDB コンポーネントにはオプションがありません。

InfluxDB エンドポイントは URI 構文を使用して設定します。

influxdb:connectionBean

以下の path パラメーターおよびクエリーパラメーターを使用します。

154.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
connectionBean	InfluxDB.class クラスの influx データベースへの **必須** 接続		文字列

154.2.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
batch （プロデューサー）	この操作がバッチ操作であるかどうかを定義します。	false	boolean
databaseName (producer)	時系列が保存されるデータベースの名前		文字列
操作 （プロデューサー）	この操作が挿入またはクエリーであるかどうかを定義します。	insert	文字列
クエリー （プロデューサー）	操作クエリーの場合にクエリーを定義します。		文字列
retentionPolicy (producer)	エンドポイントによって作成されるデータに対して保持ポリシーを定義する文字列	default	文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

154.3. メッセージヘッダー

Name	デフォルト値	Type	コンテキスト	説明

154.4. 例

以下は、ポイントを db（URI から db 名を起動する）に固有のキーに保存するルート例です。

```
from("direct:start")
        .setHeader(InfluxDbConstants.DBNAME_HEADER, constant("myTimeSeriesDB"))
        .to("influxdb://connectionBean);
```

```
from("direct:start")
        .to("influxdb://connectionBean?databaseName=myTimeSeriesDB");
```

詳細情報は、これらのリソース...

154.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第155章 IRC コンポーネント

Camel バージョン 1.1 で利用可能

irc コンポーネントは IRC (Internet Relay Chat)トランスポートを実装します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-irc</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

155.1. URI 形式

```
irc:nick@host[:port]/#room[?options]
irc:nick@host[:port]?channels=#channel1,#channel2,#channel3[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

155.2. オプション

IRC コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

IRC エンドポイントは、URI 構文を使用して設定されます。

irc:hostname:port

以下の path パラメーターおよびクエリーパラメーターを使用します。

155.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
hostname	IRC チャットサーバーに **必要な** ホスト名		文字列
port	IRC チャットサーバーのポート番号。ポートが設定されていない場合は、6667、6668、または 6669 のいずれかのデフォルトポートが使用されます。		int

155.2.2. クエリーパラメーター（24 パラメーター）：

Name	説明	デフォルト	Type
autoRejoin (common)	キック時に自動参加するかどうか	true	boolean
namesOnJoin (common)	参加後に NAMES コマンドをチャネルに送信します。応答上のリンクは true である必要があります。ヘッダーの値が irc.num = '353' の結果を処理するには、応答上のリンクが true である必要があります。	false	boolean
ニックネーム (common)	チャットで使用されるニックネーム。		文字列
Persistent (common)	**非推奨** となった永続メッセージの使用。	true	boolean
realname (common)	IRC ユーザーの実際の名前。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
colors （詳細）	サーバーが色コードをサポートするかどうか。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
onJoin (filter)	ユーザーの結合イベントを処理します。	true	boolean
onKick (filter)	キックイベントを処理します。	true	boolean
onMode (filter)	モード変更イベントを処理します。	true	boolean
onNick (filter)	ニックネーム変更イベントを処理します。	true	boolean
onPart (filter)	ユーザーパートイベントを処理します。	true	boolean
onPrivmsg (filter)	プライベートメッセージイベントを処理します。	true	boolean
onQuit (filter)	ユーザーの終了イベントを処理します。	true	boolean
onReply (filter)	コマンドまたは情報メッセージへの一般的な応答を処理するかどうか。	false	boolean
onTopic (filter)	トピック変更イベントを処理します。	true	boolean
nickPassword (security)	IRC サーバーのニックネームのパスワード。		文字列
パスワード （セキュリティー）	IRC サーバーパスワード。		文字列

Name	説明	デフォルト	Type
sslContextParameters (security)	SSL を使用したセキュリティーの設定に使用されます。レジストリーの org.apache.camel.util.jsse.SSLContextParameters への参照。この参照は、コンポーネントレベルで設定された SSLContextParameters を上書きします。この設定は trustManager オプションを上書きすることに注意してください。		SSLContextParameters
trustManager (security)	SSL サーバーの証明書を検証するために使用されるトラストマネージャー。		SSLTrustManager
ユーザー名 （セキュリティー）	IRC サーバーのユーザー名。		文字列

155.3. SSL サポート

155.3.1. JSSE 設定ユーティリティーの使用

Camel 2.9 の時点で、IRC コンポーネントは Camel JSSE 設定ユーティリティーを介して SSL/TLS 設定をサポートします。 このユーティリティーは、エンドポイントおよびコンポーネントレベルで記述し、設定する必要のあるコンポーネント固有のコードの量を大幅に削減します。 以下の例は、IRC コンポーネントでユーティリティーを使用する方法を示しています。

エンドポイントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/truststore.jks");
ksp.setPassword("keystorePassword");

TrustManagersParameters tmp = new TrustManagersParameters();
tmp.setKeyStore(ksp);

SSLContextParameters scp = new SSLContextParameters();
scp.setTrustManagers(tmp);

Registry registry = ...
registry.bind("sslContextParameters", scp);

...

from(...)
    .to("ircs://camel-prd-user@server:6669/#camel-test?nickname=camel-prd&password=password&sslContextParameters=#sslContextParameters");
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
      id="sslContextParameters">
    <camel:trustManagers>
      <camel:keyStore
          resource="/users/home/server/truststore.jks"
          password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="ircs://camel-prd-user@server:6669/#camel-test?nickname=camel-
prd&password=password&sslContextParameters=#sslContextParameters"/>...
```

155.3.2. レガシーの基本的な設定オプションの使用

また、以下のように SSL 対応の IRC サーバーに接続することもできます。

```
ircs:host[:port]/#room?username=user&password=pass
```

デフォルトでは、IRC トランスポートは SSLDefaultTrustManager を使用します。独自のカスタムトラストマネージャーを提供する必要がある場合は、以下のように trustManager パラメーターを使用します。

```
ircs:host[:port]/#room?
username=user&password=pass&trustManager=#referenceToMyTrustManagerBean
```

155.4. 鍵の使用

Camel 2.2 で利用可能

一部の irc 部屋では、そのチャネルに参加できるようにキーを提供する必要があります。キーは秘密語です。

たとえば、チャンネル 1 と 3 のみがキーを使用する 3 つのチャンネルに参加します。

```
irc:nick@irc.server.org?channels=#chan1,#chan2,#chan3&keys=chan1Key,,chan3key
```

155.5. チャンネルの一覧の取得

name OnJoin オプション を使用すると、コンポーネントがチャネルに参加した後に IRC-NAMES コマンドを呼び出すことができます。サーバーは irc.num = 353 で応答します。そのため、結果を処理するには onReply プロパティーが true である必要があります。さらに、名前を取得するために onReply エクスチェンジをフィルターする必要があります。

たとえば、チャネルのユーザー名が含まれるすべてのエクスチェンジを取得する場合：

```
from("ircs:nick@myserver:1234/#mychannelname?namesOnJoin=true&onReply=true")
 .choice()
  .when(header("irc.messageType").isEqualToIgnoreCase("REPLY"))
   .filter(header("irc.num").isEqualTo("353"))
   .to("mock:result").stop();
```

155.6. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第156章 JACKSONXML DATAFORMAT

Camel バージョン 2.16 から利用可能

Jackson XML は、XML Mapper エクステンションを持つ Jackson ライブラリー を使用して、XML ペイロードを Java オブジェクトにアンマーシャリングしたり、Java オブジェクトを XML ペイロードにマーシャリングしたりする Data Format です。

INFO: Jackson に精通している場合、この XML データフォーマットは JSON に対応するものと同じように動作するため、JSON シリアライゼーション/デシリアライゼーションのアノテーションを付けるクラスで使用できます。

このエクステンションは、JAXB の「Code first」アプローチ も見ます。

このデータ形式は、高速で効率的な XML プロセッサーである Woodstox （特に、プリ印刷などの機能）に依存します。

```
from("activemq:My.Queue").
  unmarshal().jacksonxml().
  to("mqseries:Another.Queue");
```

156.1. JACKSONXML オプション

JacksonXML データフォーマットは 15 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
xmlMapper		文字列	指定した id で既存の XmlMapper を検索し、使用します。
prettyPrint	false	ブール値	プリションでフォーマットされた出力を有効にします。デフォルトは false です。
unmarshalTypeName		文字列	無効化解除時に使用する java タイプのクラス名

Name	デフォルト	Java タイプ	説明
jsonView		class<?>	POJO を JSON にマーシャリングする場合は、JSON 出力から特定のフィールドを除外したい場合があります。Jackson を使用すると、JSON ビューを使用してこれを実行できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		文字列	pojo を JSON にマーシャリングし、pojo には null 値を持つフィールドがあります。また、これらの null 値を省略する場合は、このオプションを NOT_NULL に設定します。
allowJmsType	false	ブール値	JMS 仕様からの JMSType ヘッダーがアンマーシャリングに使用する FQN クラス名を指定するために JMS ユーザーに使用されます。
collectionTypeName		文字列	使用するレジストリーで検索するカスタムコレクションタイプを参照します。このオプションはほとんど使用されることはありませんが、デフォルトとして java.util.Collection とは異なるコレクションタイプを使用できます。
useList	false	ブール値	マップのリストまたは Pojo のリストへ無視する場合は、以下を行います。
enableJaxbAnnotationModule	false	ブール値	Jackson を使用する際に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson により JAXB アノテーションを使用できます。
moduleClassNames		文字列	FQN クラス名で String として指定されたカスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用します。複数のクラスをコンマで区切ることができます。
moduleRefs		文字列	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		文字列	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。
disableFeatures		文字列	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能セット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。

Name	デフォルト	Java タイプ	説明
allowUnmarshallType	**false**	**ブール値**	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する場合にのみ有効にしてください。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

156.1.1. Spring DSL での Jackson XML の使用

Spring DSL で Data Format を使用する場合は、最初にデータ形式を宣言する必要があります。これは DataFormats XML タグで実行されます。

```
<dataFormats>
  <!-- here we define a Xml data format with the id jack and that it should use the
TestPojo as the class type when
       doing unmarshal. The unmarshalTypeName is optional, if not provided Camel will
use a Map as the type -->
  <jacksonxml id="jack"
unmarshalTypeName="org.apache.camel.component.jacksonxml.TestPojo"/>
</dataFormats>
```

次に、ルートでこの ID を参照できます。

```
<route>
  <from uri="direct:back"/>
  <unmarshal ref="jack"/>
  <to uri="mock:reverse"/>
</route>
```

156.2. POJO フィールドのマーシャリングから除外

POJO を XML にマーシャリングする場合は、XML の出力から特定のフィールドを除外したい場合があります。Jackson を使用すると、 JSON ビュー を使用してこれを実行できます。まず、1 つ以上のマーカークラスを作成します。

@JsonView アノテーションを持つマーカークラスを使用して、特定のフィールドを包含/除外します。アノテーションはゲッターでも動作します。

最後に Camel JacksonXMLDataFormat を使用して、上記の POJO を XML にマーシャリングします。

結果の XML には weight フィールドがありません。

```
<pojo age="30" weight="70"/>
```

156.3. 'JACKSONXML'DATAFORMAT の JSONVIEW 属性を使用してフィールドを追加/除外

この属性の使用例として、以下の代わりに使用できます。

```
JacksonXMLDataFormat ageViewFormat = new JacksonXMLDataFormat(TestPojoView.class,
Views.Age.class);
from("direct:inPojoAgeView").
  marshal(ageViewFormat);
```

以下のように Java DSL 内で JSON ビュー を直接指定します。

```
from("direct:inPojoAgeView").
  marshal().jacksonxml(TestPojoView.class, Views.Age.class);
```

XML DSL の場合と同じです。

```
<from uri="direct:inPojoAgeView"/>
  <marshal>
    <jacksonxml unmarshalTypeName="org.apache.camel.component.jacksonxml.TestPojoView"
jsonView="org.apache.camel.component.jacksonxml.Views$Age"/>
  </marshal>
```

156.4. シリアル化の INCLUDE オプションの設定

pojo を XML にマーシャリングし、pojo には null 値を持つフィールドがあります。そして、これらの null 値を省略する場合は、pojo にアノテーションを設定する必要があります。

```
@JsonInclude(Include.NON_NULL)
public class MyPojo {
   ...
}
```

ただし、これには、pojo ソースコードにそのアノテーションを追加する必要があります。Camel JacksonXMLDataFormat は、以下のように include オプションを設定することもできます。

```
JacksonXMLDataFormat format = new JacksonXMLDataFormat();
format.setInclude("NON_NULL");
```

または、XML DSL からこれを設定します。

```
<dataFormats>
  <jacksonxml id="jacksonxml" include="NOT_NULL"/>
</dataFormats>
```

156.5. 動的なクラス名で XML から POJO へのアンマーシャリング

jackson を使用して XML を POJO にアンマーシャリングする場合は、アンマーシャリングするクラス名を示すメッセージにヘッダーを指定できるようになりました。
メッセージにヘッダーが存在する場合は CamelJacksonUnmarshalType がキーを持ち、Jackson は XML ペイロードをアンマーシャリングするために POJO クラスの FQN として使用します。

JMS エンドユーザーには、JMS 仕様に JMSType ヘッダーも示す JMSType ヘッダーがあります。JMSType のサポートを有効にするには、以下のように jackson データフォーマットでこれを有効にする必要があります。

```
JacksonDataFormat format = new JacksonDataFormat();
format.setAllowJmsType(true);
```

または、XML DSL からこれを設定します。

```
<dataFormats>
  <jacksonxml id="jacksonxml" allowJmsType="true"/>
</dataFormats>
```

156.6. XML から LIST<MAP> または LIST<POJO> へのアンマーシャリング

Jackson を使用して XML を map/pojo のリストにアンマーシャリングする場合は、useList="true" を設定するか、org.apache.camel.component.jacksonxml.ListJacksonXMLDataFormat を使用するよう指定できるようになりました。たとえば、Java では以下のように実行できます。

```
JacksonXMLDataFormat format = new ListJacksonXMLDataFormat();
// or
```

```
JacksonXMLDataFormat format = new JacksonXMLDataFormat();
format.useList();
// and you can specify the pojo class type also
format.setUnmarshalType(MyPojo.class);
```

XML DSL を使用する場合は、以下のように useList 属性を使用してリストを使用するように設定します。

```
<dataFormats>
  <jacksonxml id="jack" useList="true"/>
</dataFormats>
```

また、pojo タイプを指定することもできます。

```
<dataFormats>
  <jacksonxml id="jack" useList="true" unmarshalTypeName="com.foo.MyPojo"/>
</dataFormats>
```

156.7. カスタム JACKSON モジュールの使用

カスタム Jackson モジュールを使用するには、以下のように moduleClassNames オプションを使用してそれらのクラス名を指定します。

```
<dataFormats>
  <jacksonxml id="jack" useList="true" unmarshalTypeName="com.foo.MyPojo"
moduleClassNames="com.foo.MyModule,com.foo.MyOtherModule"/>
</dataFormats>
```

moduleClassNames を使用する場合、デフォルトのコンストラクターを使用して作成され、そのまま使用されるカスタム jackson モジュールは設定されません。カスタムモジュールでカスタム設定が必要な場合は、モジュールのインスタンスを作成して設定し、modulesRefs を使用して以下のようにモジュールを参照します。

```
<bean id="myJacksonModule" class="com.foo.MyModule">
  ... // configure the module as you want
</bean>

<dataFormats>
  <jacksonxml id="jacksonxml" useList="true" unmarshalTypeName="com.foo.MyPojo"
moduleRefs="myJacksonModule"/>
</dataFormats>
```

moduleRefs="myJacksonModule,myOtherModule" などのように、複数のモジュールをコンマで区切って指定できます。

156.8. JACKSON を使用した機能の有効化または無効化

Jackson には、ObjectMapper が使用する数多くの機能を有効化または無効化できます。たとえば、マーシャリング時に不明なプロパティーの失敗を無効にするには、disableFeatures を使用してこれを設定します。

```
<dataFormats>
    <jacksonxml id="jacksonxml" unmarshalTypeName="com.foo.MyPojo"
disableFeatures="FAIL_ON_UNKNOWN_PROPERTIES"/>
</dataFormats>
```

コンマで区切ると、複数の機能を無効にできます。機能の値は、以下の列挙クラスの Jackson の列挙名である必要があります。

- com.fasterxml.jackson.databind.SerializationFeature
- com.fasterxml.jackson.databind.DeserializationFeature
- com.fasterxml.jackson.databind.MapperFeature

機能を有効にするには、代わりに enableFeatures オプションを使用します。

Java コードから、camel-jackson モジュールのタイプセーフメソッドを使用できます。

```
JacksonDataFormat df = new JacksonDataFormat(MyPojo.class);
df.disableFeature(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES);
df.disableFeature(DeserializationFeature.FAIL_ON_NULL_FOR_PRIMITIVES);
```

156.9. JACKSON を使用した POJO へのマップ変換

Jackson ObjectMapper を使用すると、POJO オブジェクトにマッピングを変換できます。Jackson コンポーネントには、 java.util.Map インスタンスを非文字列、非プリミティブおよび非 Number オブジェクトに変換するのに使用できるデータコンバーターが同梱されています。

```
Map<String, Object> invoiceData = new HashMap<String, Object>();
invoiceData.put("netValue", 500);
```

```
producerTemplate.sendBody("direct:mapToInvoice", invoiceData);
...
// Later in the processor
Invoice invoice = exchange.getIn().getBody(Invoice.class);
```

Camel レジストリー内に単一の ObjectMapper インスタンスがある場合、変換を実行するためにコンバーターによって使用されます。それ以外の場合は、デフォルトのマッパーが使用されます。

156.10. フォーマットされた XML マーシャリング(PRETTY-PRINTING)

prettyPrint オプションを使用すると、マーシャリング中に適切にフォーマットされた XML を出力することができます。

```
<dataFormats>
    <jacksonxml id="jack" prettyPrint="true"/>
</dataFormats>
```

Java DSL では、以下のようになります。

```
from("direct:inPretty").marshal().jacksonxml(true);
```

unmarshalType、 jsonView などの他の設定と組み合わせて、pre Print オプションをサポートするオーバーロードされた jacksonxml（） DSL メソッドが 5 つ存在することに注意してください。

156.11. 依存関係

camel ルートで Jackson XML を使用するには、このデータ形式を実装する camel-jacksonxml の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jacksonxml</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第157章 JASYPT コンポーネント

Camel 2.5 で利用可能

j asypt は簡素化された暗号化ライブラリーで、暗号化と復号を容易にします。Camel は Jasypt と統合し、プロパティー ファイルの機密情報を暗号化できるようにします。暗号化した値は自動的に Camel によって復号化されます。クラスパスで camel-jasypt を削除すると、その暗号化された値は自動的に Camel によって復号化されます。これにより、人間の eyes が、ユーザー名とパスワードなどの機密情報を簡単に見つけ出すことができません。

Maven を使用している場合は、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jasypt</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Apache Karaf コンテナーを使用している場合は、以下の依存関係をこのコンポーネントの pom.xml に追加する必要があります。

```
<dependency>
  <groupId>org.apache.karaf.jaas</groupId>
  <artifactId>org.apache.karaf.jaas.jasypt</artifactId>
  <version>x.x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

157.1. ツール

Jasypt コンポーネントは、値を暗号化または復号化するコマンドラインツールをほとんど提供します。

コンソールでは、構文と、提供するオプションが出力されます。

```
Apache Camel Jasypt takes the following options

  -h or -help = Displays the help screen
  -c or -command <command> = Command either encrypt or decrypt
```

```
-p or -password <password> = Password to use
-i or -input <input> = Text to encrypt or decrypt
-a or -algorithm <algorithm> = Optional algorithm to use
```

たとえば、以下のパラメーターで実行した値tiger を暗号化するには、以下のコマンドを実行します。apache camel kit では、lib フォルダーに移動し、以下の java cmd を実行します。ここで、< CAMEL_HOME > は Camel ディストリビューションをダウンロードして抽出します。

```
$ cd <CAMEL_HOME>/lib
$ java -jar camel-jasypt-2.5.0.jar -c encrypt -p secret -i tiger
```

以下の結果を出力する

```
Encrypted text: qaEEacuW7BUti8LcMgyjKw==
```

これは、暗号化された表現 qaEEacuW7BUti8LcMgyjKw== は、シークレット が作成されたマスターパスワードが分かっている場合は、tiger に復号化できることを意味します。
ツールを再度実行すると、暗号化された値が別の結果を返します。値を復号すると、常に正しい元の値が返されます。

したがって、以下のパラメーターを使用してツールを実行してテストできます。

```
$ cd <CAMEL_HOME>/lib
$ java -jar camel-jasypt-2.5.0.jar -c decrypt -p secret -i qaEEacuW7BUti8LcMgyjKw==
```

以下のような結果が出力されます。

```
Decrypted text: tiger
```

その後、プロパティー ファイルで暗号化された値を使用します。パスワードの値が暗号化され、値には ENC(value here)の周りのトークンがあることに注意してください。

ヒント

jasypt ツールを実行する場合は、java.lang.NoClassDefFoundError: org/jasypt/encryption/pbe/StandardPBEStringEncryptor にまたがる場合は、クラスパスに jasypt7.1.jar を含める必要があります。jar をクラスパスに追加する例は、java -jar … として実行される場合は jasypt7.1.jar を $JAVA_HOME\jre\lib\ext にコピーすることがあります。後者の場合は、-cp を使用して jasypt7. 1.jar をクラスパスに追加できます。この場合、java -cp jasypt-1.9.2.jar:camel-jasypt-2.18.2.jar org.apache.camel.component.jasypt.Main -c encrypt -p secret -i tiger

157.2. URI オプション

以下のオプションは Jasypt コンポーネント専用です。

Name	デフォルト値	型	説明
password	**null**	**文字列**	復号化に使用するマスターパスワードを指定します。このオプションは必須です。詳細は、以下を参照してください。
algorithm	**null**	**文字列**	使用するオプションのアルゴリズムの名前。

157.3. マスターパスワードの保護

Jasypt が使用するマスターパスワードを指定して、値を復号化できるようにする必要があります。ただし、このマスターパスワードをオープンにすると、理想的なソリューションであるとは限りません。そのため、たとえば JVM システムプロパティーまたは OS 環境設定として指定できます。これを実行する場合、password オプションは、これを示す接頭辞をサポートします。sysenv: 指定のキーで OS システム環境を検索することを意味します。sys: JVM システムプロパティーを検索することを意味します。

たとえば、アプリケーションを起動する前にパスワードを入力することができます。

```
$ export CAMEL_ENCRYPTION_PASSWORD=secret
```

次に、起動スクリプトの実行など、アプリケーションを起動します。

アプリケーションが起動したら、環境の設定を解除することができます。

```
$ unset CAMEL_ENCRYPTION_PASSWORD
```

password オプションは、password=sysenv:CAMEL_ENCRYPTION_PASSWORD のように定義します。

157.4. JAVA DSL を使用した例

Java DSL では、Jacsypt を Jasypt Parser インスタンスとして設定し、以下のように Properties コンポーネントに設定する必要があります。

プロパティーファイル myproperties.properties には、以下のように暗号化された値が含まれます。パスワードの値が暗号化され、値には ENC(value here)の周りのトークンがあることに注意してください。

157.5. SPRING XML を使用した例

Spring XML では、以下に示す JasyptPropertiesParser を設定する必要があります。その後、Camel Properties コンポーネントは jasypt をプロパティーパーサーとして使用するよう指示されます。つまり、Jasypt はプロパティーで検索された値を復号化する機会を持っていることを意味します。

```
<!-- define the jasypt properties parser with the given password to be used -->
<bean id="jasypt" class="org.apache.camel.component.jasypt.JasyptPropertiesParser">
    <property name="password" value="secret"/>
</bean>

<!-- define the camel properties component -->
<bean id="properties" class="org.apache.camel.component.properties.PropertiesComponent">
    <!-- the properties file is in the classpath -->
    <property name="location"
value="classpath:org/apache/camel/component/jasypt/myproperties.properties"/>
    <!-- and let it leverage the jasypt parser -->
    <property name="propertiesParser" ref="jasypt"/>
</bean>
```

Properties コンポーネントは、以下に示す < camelContext > タグ内でインラインにすることもできます。propertiesParserRef 属性を使用して Jasypt を参照することに注意してください。

```
<!-- define the jasypt properties parser with the given password to be used -->
<bean id="jasypt" class="org.apache.camel.component.jasypt.JasyptPropertiesParser">
    <!-- password is mandatory, you can prefix it with sysenv: or sys: to indicate it should use
        an OS environment or JVM system property value, so you dont have the master
password defined here -->
    <property name="password" value="secret"/>
</bean>

<camelContext xmlns="http://camel.apache.org/schema/spring">
    <!-- define the camel properties placeholder, and let it leverage jasypt -->
    <propertyPlaceholder id="properties"

location="classpath:org/apache/camel/component/jasypt/myproperties.properties"
                        propertiesParserRef="jasypt"/>
    <route>
        <from uri="direct:start"/>
        <to uri="{{cool.result}}"/>
    </route>
</camelContext>
```

–

157.6. BLUEPRINT XML を使用した例

Blueprint XML では、以下に示す JasyptPropertiesParser を設定する必要があります。その後、Camel Properties コンポーネントは jasypt をプロパティーパーサーとして使用するよう指示されます。つまり、Jasypt はプロパティーで検索された値を復号化する機会を持っていることを意味します。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
         xsi:schemaLocation="
         http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

  <cm:property-placeholder id="myblue" persistent-id="mypersistent">
    <!-- list some properties for this test -->
    <cm:default-properties>
      <cm:property name="cool.result" value="mock:{{cool.password}}"/>
      <cm:property name="cool.password" value="ENC(bsW9uV37gQ0QHFu7KO03Ww==)"/>
    </cm:default-properties>
  </cm:property-placeholder>

  <!-- define the jasypt properties parser with the given password to be used -->
  <bean id="jasypt" class="org.apache.camel.component.jasypt.JasyptPropertiesParser">
    <property name="password" value="secret"/>
  </bean>

  <camelContext xmlns="http://camel.apache.org/schema/blueprint">
    <!-- define the camel properties placeholder, and let it leverage jasypt -->
    <propertyPlaceholder id="properties"
                         location="blueprint:myblue"
                         propertiesParserRef="jasypt"/>
    <route>
      <from uri="direct:start"/>
      <to uri="{{cool.result}}"/>
    </route>
  </camelContext>

</blueprint>
```

Properties コンポーネントは、以下に示す < camelContext > タグ内でインラインにすることもできます。propertiesParserRef 属性を使用して Jasypt を参照することに注意してください。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
         xsi:schemaLocation="
         http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">
```

```
    <!-- define the jasypt properties parser with the given password to be used -->
    <bean id="jasypt" class="org.apache.camel.component.jasypt.JasyptPropertiesParser">
        <property name="password" value="secret"/>
    </bean>

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
      <!-- define the camel properties placeholder, and let it leverage jasypt -->
      <propertyPlaceholder id="properties"
                           location="classpath:org/apache/camel/component/jasypt/myproperties.properties"
                           propertiesParserRef="jasypt"/>
        <route>
            <from uri="direct:start"/>
            <to uri="{{cool.result}}"/>
        </route>
    </camelContext>

</blueprint>
```

157.7. 関連項目

- セキュリティー
- プロパティー
- ActiveMQ - ActiveMQ で暗号化されたパスワード には、camel-jasypt コンポーネントと同様の機能があります。

第158章 JAXB DATAFORMAT

Camel バージョン 1.0 で利用可能

JAXB は Data Format で、Java 6 に含まれる JAXB2 XML マーシャリング標準を使用して XML ペイロードを Java オブジェクトにアンマーシャリングしたり、Java オブジェクトを XML ペイロードにマーシャリングしたりします。

158.1. オプション

JAXB データフォーマットは、以下に示す 18 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
contextPath		**文字列**	JAXB クラスが置かれているパッケージ名。
schema		**文字列**	既存のスキーマに対して検証する。接頭辞 classpath:、file:、または http: を使用して、リソースの解決方法を指定できます。','文字を使用して、複数のスキーマファイルを分離できます。
schemaSeverityLevel	**0**	**整数**	スキーマに対して検証する際に使用するスキーマの重大度レベルを設定します。このレベルは、JAXB が解析を継続しないようにトリガーする最低重大度エラーを決定します。デフォルト値の 0(warning)は、エラー（warning、error、または fatal error）が JAXB をトリガーして停止することを意味します。0=warning、1=error、2=fatal エラーの 3 つのレベルがあります。
prettyPrint	**false**	**ブール値**	プリションでフォーマットされた出力を有効にします。デフォルトは false です。
objectFactory	**false**	**ブール値**	マーシャリング中に ObjectFactory クラスを使用して POJO クラスを作成できるようにするかどうか。これは、JAXB アノテーションが付けられた POJO クラスと jaxb.index 記述子ファイルの提供のみに適用されます。
ignoreJAXBElement	**false**	**ブール値**	JAXBElement 要素を無視するかどうか。非常に特殊なユースケースで false のみに設定する必要があります。
mustBeJAXBElement	**false**	**ブール値**	marhsalling が JAXB アノテーションを持つ java オブジェクトである必要があるかどうか。そうでない場合は、失敗します。このオプションは false に設定すると、データがすでに XML 形式の場合などに緩和されます。

Name	デフォルト	Java タイプ	説明
filterNonXmlChars	false	ブール値	xml 以外の文字を無視し、空白のスペースに置き換えるには、以下を行います。
encoding		文字列	特定のエンコーディングを過剰に実行し、特定のエンコーディングを使用します。
fragment	false	ブール値	マーシャリング XML フラグメントツリーをオンにします。デフォルトでは JAXB は指定されたクラスで XmlRootElement アノテーションを検索して、XML ツリー全体で動作します。これは必ずしも便利ですが、生成されたコードに XmlRootElement アノテーションがない場合があり、時にはツリーの一部のみをアンマーシャリングする必要がある場合があります。この場合、部分的なアンマーシャリングを使用できます。この動作を有効にするには、プロパティー partClass を設定する必要があります。Camel はこのクラスを JAXB のアンマーシャリングに渡します。
partClass		文字列	フラグメント解析に使用されるクラスの名前。詳細は、「fragment オプション」を参照してください。
partNamespace		文字列	フラグメントの解析に使用する XML 名前空間。詳細は、「fragment オプション」を参照してください。
namespacePrefixRef		文字列	JAXB または SOAP を使用してマーシャリングする場合、JAXB 実装は ns2、ns3、ns4 などの名前空間接頭辞を自動的に割り当てます。このマッピングを制御するために、Camel では必要なマッピングが含まれるマップを参照できます。
xmlStreamWriterWrapper		文字列	カスタム xml ストリームライターを使用するには、以下を行います。
schemaLocation		文字列	スキーマの場所を定義します。
noNamespaceSchemaLocation		文字列	namespaceless スキーマの場所を定義します。
jaxbProviderProperties		文字列	カスタム java.util.Map を参照し、JAXB マーシャラーと使用するカスタム JAXB プロバイダープロパティーが含まれるレジストリーでルックアップします。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

158.2. JAVA DSL の使用

たとえば、以下は、JAXBContext を初期化するために、多数の Java パッケージ名で設定される jaxb の名前付き DataFormat を使用します。

```
DataFormat jaxb = new JaxbDataFormat("com.acme.model");

from("activemq:My.Queue").
  unmarshal(jaxb).
  to("mqseries:Another.Queue");
```

データフォーマットに名前付きの参照を使用する場合は、Spring XML ファイルなどを介してレジストリーで定義できます。

```
from("activemq:My.Queue").
  unmarshal("myJaxbDataType").
  to("mqseries:Another.Queue");
```

158.3. SPRING XML の使用

以下の例は、JAXB を使用して jaxb のデータ型を設定する方法を示しています。

この例は、データ型を一度だけ設定し、複数のルートに再利用する方法を示しています。

複数のコンテキストパス

このデータ形式は、複数のコンテキストパスで使用できます。コンテキストパスは区切り文字として指定することができます（例 : com.mycompany:com.mycompany2 ）。これは JAXB 実装によって処理され、RI 以外のベンダーを使用する場合は変更される可能性があります。

158.4. 部分的なマーシャリング/アンマーシャリング

この機能は Camel 2.2.0 の新機能です。
JAXB 2 はマーシャリングおよびアンマーシャリング XML ツリーフラグメントをサポートします。デフォルトでは JAXB は指定されたクラスで @XmlRootElement アノテーションを検索して、XML ツリー全体で動作します。これは必ずしも便利ですが、生成されたコードに @XmlRootElement アノテーションがない場合があり、時にはツリーの一部のみをアンマーシャリングする必要がある場合があります。
この場合、部分的なアンマーシャリングを使用できます。この動作を有効にするには、プロパティー partClass を設定する必要があります。Camel はこのクラスを JAXB のアンマーシャリングに渡します。JaxbConstants.JAXB_PART_CLASS がヘッダーの 1 つとして設定されている場合（partClass

プロパティーが DataFormat に設定されている場合でも）、DataFormat のプロパティーは最後に渡され、ヘッダーに設定されたプロパティーが使用されます。

マーシャリングには、宛先 namespace の QName を指定して partNamespace 属性を追加する必要があります。上記で確認できる Spring DSL の例JaxbConstants.JAXB_PART_NAMESPACE がヘッダーの 1 つとして設定されている場合（partNamespace プロパティーが DataFormat に設定されている場合でも）、DataFormat のプロパティーは最後にパスされ、ヘッダーに設定されたプロパティーが使用されます。JaxbConstants.JAXB_PART_NAMESPACE を介して partNamespace を設定する場合、その値 {[namespaceUri]}[localPart] を指定する必要があることに注意してください。

```
...
.setHeader(JaxbConstants.JAXB_PART_NAMESPACE, simple("
{http://www.camel.apache.org/jaxb/example/address/1}address"));
...
```

158.5. FRAGMENT

この機能は Camel 2.8.0 の新機能です。
JaxbDataFormat には新しいプロパティーフラグメントがあり、JAXB Marshaller に Marshaller.JAXB_FRAGMENT エンコーディングプロパティーを設定できます。JAXB Marshaller が XML 宣言を生成したくない場合は、このオプションを true に設定します。このプロパティーのデフォルト値は false です。

158.6. NONXML 文字を無視する

この機能は Camel 2.2.0 の新機能です。
JaxbDataFromat は NonXML 文字を無視することをサポートしています。filterNonXmlChars プロパティーを true に設定する必要があります。JacbDataFormat は NonXML 文字は " で、メッセージをマーシャリングまたはアンマーシャリングするときに " に置き換えられます。また、Exchange プロパティー Exchange.FILTER_NON_XML_CHARS を設定して実行することもできます。

	JDK 1.5	JDK 1.6+
使用中のフィルタリング	Stax API および実装	非対応
使用されていないフィルタリング	Stax API のみ	非対応

この機能は、Woodstox 3.2.9 および Sun JDK 1.6 StAX 実装でテストされています。

Camel 2.12.1
JaxbDataFormat では、ストリームを XML にマーシャリングするために使用される XMLStreamWriter をカスタマイズすることができるようになりました。この設定を使用すると、独自のストリームライターを追加して、非xml 文字を完全に削除、エスケープ、または置き換えることができます。

```
JaxbDataFormat customWriterFormat = new JaxbDataFormat("org.apache.camel.foo.bar");
customWriterFormat.setXmlStreamWriterWrapper(new TestXmlStreamWriter());
```

以下の例は、Spring DSL を使用して Camel の NonXML フィルターを有効にする方法を示しています。

```
<bean id="testXmlStreamWriterWrapper" class="org.apache.camel.jaxb.TestXmlStreamWriter"/>
<jaxb filterNonXmlChars="true"  contextPath="org.apache.camel.foo.bar"
xmlStreamWriterWrapper="#testXmlStreamWriterWrapper" />
```

158.7. OBJECTFACTORY の使用

XJC を使用してスキーマから java クラスを作成する場合は、JAXB コンテキストの ObjectFactory を取得します。ObjectFactory は JAXBElement を使用してスキーマと要素のインスタンス値の参照を保持するため、jaxbDataformat はデフォルトで JAXBElement を無視し、アンマーシャリングされたメッセージボディーを形成する JAXBElement オブジェクトの代わりに要素のインスタンス値を取得します。
JAXBElement オブジェクトがアンマーシャリングメッセージのボディーを形成する場合は、JacbDataFormat オブジェクトの ignoreJAXBElement プロパティーを false に設定する必要があります。

158.8. エンコーディングの設定

マーシャリング時に使用する encoding オプションを設定できます。JAXB Marshaller の Marshaller.JAXB_ENCODING エンコーディングプロパティー。
JAXB データフォーマットを宣言するときに使用するエンコーディングを設定できます。Exchange プロパティー Exchange.CHARSET_NAME にエンコーディングを指定することもできます。このプロパティーは、JAXB データフォーマットに設定されたエンコーディングをオーバールールします。

この Spring DSL では、iso-8859-1 をエンコーディングとして使用するよう定義しました。

158.9. 名前空間接頭辞のマッピングの制御

Camel 2.11 から利用可能

JAXB または SOAP を使用してマーシャリングする場合、JAXB 実装は ns2、ns3、ns4 などの名前空間接頭辞を自動的に割り当てます。このマッピングを制御するために、Camel では必要なマッピングが含まれるマップを参照できます。

マッピング機能は JAXB の実装に依存するため、このクラスには JAXB-RI 2.1 以上（SUN から）以上が必要です。これは、マッピング機能は JAXB の実装に依存するためです。

たとえば、Spring XML ではマッピングでマップを定義できます。以下のマッピングファイルで SOAP をプレフィックスとして使用するようマッピングします。カスタム名前空間 "http://www.mycompany.com/foo/2" は接頭辞を使用しませんが、カスタムの名前空間 "http://www.mycompany.com/foo/2" は一切使用していません。

```
<util:map id="myMap">
  <entry key="http://www.w3.org/2003/05/soap-envelope" value="soap"/>
  <!-- we dont want any prefix for our namespace -->
  <entry key="http://www.mycompany.com/foo/2" value=""/>
</util:map>
```

JAXB または SOAP で使用するには、以下のように namespacePrefixRef 属性を使用してこのマップを参照します。その後、Camel は レジストリー で、上で定義した内容である id "myMap" で java.util.Map を検索します。

```
<marshal>
  <soapjaxb version="1.2" contextPath="com.mycompany.foo" namespacePrefixRef="myMap"/>
</marshal>
```

158.10. スキーマ検証

Camel 2.11 から利用可能

JAXB データフォーマットは、XML からのマーシャリングおよびアンマーシャリングによる検証をサポートします。接頭辞 classpath:、file: または http: を使用して、リソースの解決方法を指定できます。',' 文字を使用して、複数のスキーマファイルを分離できます。

既知の問題

Camel 2.11.0 および 2.11.1 は、複数の 'Exchange' を並行して検証することで既知の問題があります。CAMEL-6630 を参照してください。これは Camel 2.11.2/2.12.0 で修正されています。

Java DSL を使用すると、以下の方法で設定できます。

```
JaxbDataFormat jaxbDataFormat = new JaxbDataFormat();
jaxbDataFormat.setContextPath(Person.class.getPackage().getName());
jaxbDataFormat.setSchema("classpath:person.xsd,classpath:address.xsd");
```

XML DSL を使用しても同じ操作を行うことができます。

```
<marshal>
    <jaxb id="jaxb" schema="classpath:person.xsd,classpath:address.xsd"/>
</marshal>
```

JDK に同梱される SchemaFactory はスレッドセーフではないため、Camel は適切に スキーマ Factory インスタンスを作成し、プールします。
ただし、スレッドセーフである SchemaFactory 実装がある場合は、JAXB データフォーマットをこれを使用するように設定できます。

```
JaxbDataFormat jaxbDataFormat = new JaxbDataFormat();
jaxbDataFormat.setSchemaFactory(thradSafeSchemaFactory);
```

158.11. スキーマの場所

Camel 2.14 から利用可能

JAXB Data Format は、XML をマーシャリングする際に SchemaLocation を指定することをサポートします。

Java DSL を使用すると、以下の方法で設定できます。

```
JaxbDataFormat jaxbDataFormat = new JaxbDataFormat();
jaxbDataFormat.setContextPath(Person.class.getPackage().getName());
jaxbDataFormat.setSchemaLocation("schema/person.xsd");
```

XML DSL を使用しても同じ操作を行うことができます。

```
<marshal>
    <jaxb id="jaxb" schemaLocation="schema/person.xsd"/>
</marshal>
```

158.12. すでに XML のデータをマーシャリングする

Camel 2.14.1 から利用可能

JAXB マーシャラーでは、メッセージボディーは JAXB と互換性があること、JAXB 要素の例、JAXB アノテーションを持つ java インスタンス、または JAXBElement を拡張する必要があります。メッセージボディーがすでに XML にある場合があります（例： String 型）。新しいオプション mustBeJAXBElement を false に設定し、このチェックを緩和するために JAXB マーシャラーは JAXBElements（javax.xml.bind.JAXBIntrospector#isElement を返す）のみのマーシャリングを試行します。また、これらの状況では、メッセージボディーをそのままマーシャリングするためのマーシャラーフォールバック。

158.13. 依存関係

camel ルートで JAXB を使用するには、このデータ形式を実装する camel-jaxb の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jaxb</artifactId>
  <version>x.x.x</version>
</dependency>
```

第159章 JBOSS DATA GRID COMPONENT

159.1. RED HAT JBOSS DATA GRID COMPONENT WITH APACHE CAMEL

Camel を Red Hat JBoss Data Grid と Red Hat JBoss Fuse で使用すると、接続性を追加するさまざまなトランスポートと API を提供することで、大規模なエンタープライズアプリケーションの統合を簡素化できます。

JBoss Data Grid は、Phcache を部分的に置き換える JBoss Fuse の Camel ルートでのキャッシュをサポートします。JBoss Data Grid は、埋め込みキャッシュ（ローカルまたはクラスター化）として、または Camel ルートのリモートキャッシュとしてサポートされます。

Apache Camel で Red Hat JBoss Data Grid の実行に関する詳細は、「Red Hat JBoss Data Grid の概要」を参照してください。

第160章 JCACHE コンポーネント

Camel バージョン 2.17 から利用可能

jcache コンポーネントを使用すると、JSR107/JCache をキャッシュ実装として使用してキャッシュ操作を実行できます。

160.1. URI 形式

```
jcache:cacheName[?options]
```

160.2. URI オプション

JCache エンドポイントは、URI 構文を使用して設定します。

```
jcache:cacheName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

160.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
cacheName	**必要な** キャッシュの名前		文字列

160.2.2. クエリーパラメーター（22 パラメーター）：

Name	説明	デフォルト	Type
cacheConfiguration (common)	キャッシュの設定		設定
cacheConfigurationProperties (common)	CacheManager を作成する javax.cache.spi.CachingProvider のプロパティー		プロパティー
cachingProvider (common)	javax.cache.spi.CachingProvider の完全修飾クラス名		文字列

Name	説明	デフォルト	Type
configurationUri (common)	CacheManager の実装固有の URI		文字列
managementEnabled (common)	管理収集が有効になっているかどうか。	false	boolean
readThrough (common)	読み込みスルーキャッシュを使用する必要がある場合は	false	boolean
statisticsEnabled (common)	統計収集が有効になっているかどうか。	false	boolean
storeByValue (common)	キャッシュが store-by-value または store-by-reference セマンティクスを使用する必要がある場合は	true	boolean
writeThrough (common)	ライトスルーキャッシュを使用する必要がある場合は	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
filteredEvents (consumer)	コンシューマーをフィルターする必要があるイベント。filteredEvents オプションを使用する場合、eventFilters は無視されます。		リスト
oldValueRequired (consumer)	イベントに古い値が必要な場合	false	boolean
同期 （コンシューマー）	イベントリスナーがスレッドをブロックしなければならない場合、イベントの原因	false	boolean
eventFilters (consumer)	The CacheEntryEventFilter.eventFilters オプションを使用している場合、filteredEvents は無視されます。		リスト

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
アクション （プロデューサー）	デフォルトではキャッシュ操作を使用して設定されます。メッセージヘッダーの操作がある場合は、ヘッダーからの操作が優先されます。		文字列
cacheLoaderFactory (advanced)	CacheLoader ファクトリー		CacheLoader>
cacheWriterFactory (advanced)	CacheWriter ファクトリー		CacheWriter>
createCacheIfNotExists (advanced)	キャッシュが存在する場合や事前設定できない場合は、キャッシュを作成する必要がある場合を設定します。	true	boolean
expiryPolicyFactory (advanced)	ExpiryPolicy ファクトリー		ExpiryPolicy>
lookupProviders （詳細）	camel-cache が OSGi などのランタイムで jcache api の実装の検索を試行するかどうかを設定します。	false	boolean

JCache コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
cachingProvider (common)	javax.cache.spi.CachingProvider の完全修飾クラス名		文字列
cacheConfiguration (common)	キャッシュの設定		設定
cacheConfiguration Properties (common)	CacheManager を作成する javax.cache.spi.CachingProvider のプロパティー		プロパティー

Name	説明	デフォルト	Type
configurationUri (common)	CacheManager の実装固有の URI		文字列
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

第161章 JCLOUDS コンポーネント

Camel バージョン 2.9 で利用可能

このコンポーネントにより、クラウドプロバイダーのキー/値エンジン(blobstores)および Compute サービスとの対話が可能になります。コンポーネントは jclouds を使用します。これは、Blobstore および Compute サービスの抽象化を提供するライブラリーです。

ComputeService は、クラウドでマシンを管理するタスクを簡素化します。たとえば、ComputeService を使用して 5 つのマシンを起動し、そのマシンにソフトウェアをインストールすることができます。
BlobStore は、Amazon S3 などのキーと値のペアプロバイダーの処理を簡素化します。たとえば、BlobStore にはコンテナーの単純な Map ビューを指定できます。

camel jclouds コンポーネントを使用すると、JcloudsBlobStoreEndpoint と JcloudsComputeEndpoint の 2 つのタイプのエンドポイントを指定するため、両方の抽象化を使用できます。Blobstore エンドポイントにはプロデューサーとコンシューマーの両方を設定できますが、プロデューサーは Compute エンドポイントでのみ設定できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jclouds</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

161.1. コンポーネントの設定

camel jclouds コンポーネントは、初期化中にコンポーネントに渡される限り、複数の jclouds blobstores および Compute サービスを利用します。コンポーネントは、リスト Blobstore および Compute サービスを受け入れます。以下の設定方法を説明します。

```
<bean id="jclouds" class="org.apache.camel.component.jclouds.JcloudsComponent">
  <property name="computeServices">
    <list>
      <ref bean="computeService"/>
    </list>
  </property>
  <property name="blobStores">
```

```
            <list>
                <ref bean="blobStore"/>
            </list>
        </property>
    </bean>

    <!-- Creating a blobstore from spring / blueprint xml -->
    <bean id="blobStoreContextFactory" class="org.jclouds.blobstore.BlobStoreContextFactory"/>

    <bean id="blobStoreContext" factory-bean="blobStoreContextFactory" factory-
method="createContext">
        <constructor-arg name="provider" value="PROVIDER_NAME"/>
        <constructor-arg name="identity"  value="IDENTITY"/>
        <constructor-arg name="credential" value="CREDENTIAL"/>
    </bean>

    <bean id="blobStore" factory-bean="blobStoreContext" factory-method="getBlobStore"/>

    <!-- Creating a compute service from spring / blueprint xml -->
    <bean id="computeServiceContextFactory"
class="org.jclouds.compute.ComputeServiceContextFactory"/>

    <bean id="computeServiceContext" factory-bean="computeServiceContextFactory" factory-
method="createContext">
        <constructor-arg name="provider" value="PROVIDER_NAME"/>
        <constructor-arg name="identity"  value="IDENTITY"/>
        <constructor-arg name="credential" value="CREDENTIAL"/>
    </bean>

    <bean id="computeService" factory-bean="computeServiceContext" factory-
method="getComputeService"/>
```

各種のコンポーネントで複数の Blobstores および Compute サービスを処理できることが確認できます。各エンドポイントによって使用される実際の実装は、URI 内でプロバイダーを渡すことで指定されます。

161.2. JCLOUDS オプション

```
jclouds:blobstore:[provider id][?options]
jclouds:compute:[provider id][?options]
```

プロバイダー ID は、ターゲットサービスを提供するクラウドプロバイダーの名前です（例： aws-s3 または aws_ec2）。

URI にクエリーオプションを追加するには、?option=value&option=value&...

161.3. BLOBSTORE URI オプション

JClouds コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
blobStores (common)	Blobstore の使用時に設定する必要のある指定の BlobStore を使用するには、以下を実行します。		リスト
computeServices (common)	compute を使用する際に設定する必要のある特定の ComputeService を使用するには、以下を実行します。		リスト
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JClouds エンドポイントは URI 構文を使用して設定されます。

```
jclouds:command:providerId
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

161.3.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
command	Blobstore や compute などの実行するコマンドが **必要です**。		JcloudsCommand
providerId	**必須**。ターゲットサービスを提供するクラウドプロバイダーの名前（例： aws-s3 または aws_ec2）。		文字列

161.3.2. クエリーパラメーター（15 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
blobName (blobstore)	Blob の名前。		文字列
コンテナー (blobstore)	blob コンテナーの名前。		文字列
directory (blobstore)	使用するオプションのディレクトリー名		文字列
グループ (compute)	新規に作成されたノードに割り当てられるグループ。値は実際のクラウドプロバイダーによって異なります。		文字列
hardwareId (compute)	ノードの作成に使用されるハードウェア。値は実際のクラウドプロバイダーによって異なります。		文字列
imageId (compute)	ノードの作成に使用される imageId。値は実際のクラウドプロバイダーによって異なります。		文字列
locationId (compute)	ノードの作成に使用される場所。値は実際のクラウドプロバイダーによって異なります。		文字列

Name	説明	デフォルト	Type
nodeId (compute)	スクリプトを実行するノードの ID または破棄されたノードの ID。		文字列
nodeState (compute)	ノードステータスで絞り込むには、実行中のノードのみを選択します。		文字列
操作 (compute)	blobstore に対して実行される操作のタイプを指定します。		文字列
user (compute)	スクリプトを実行するターゲットノード上のユーザー。		文字列

これらのオプションは、いくつでも指定できます。

```
jclouds:blobstore:aws-s3?
operation=CamelJcloudsGet&container=mycontainer&blobName=someblob
```

プロデューサーエンドポイントでは、適切なヘッダーをメッセージに渡すことで、上記の URI オプションをすべて上書きできます。

161.3.3. blobstore のメッセージヘッダー

ヘッダー	説明
CamelJcloudsOperation	Blob で実行される操作。有効なオプションは * PUT * GET です。
CamelJcloudsContainer	blob コンテナーの名前。
CamelJcloudsBlobName	Blob の名前。

161.4. BLOBSTORE USAGE SAMPLES

161.4.1. 例 1: Blob への配置

この例では、jclouds コンポーネントを使用して Blob 内にメッセージを保存する方法を説明します。

```
from("direct:start")
   .to("jclouds:blobstore:aws-s3" +
       "?operation=PUT" +
       "&container=mycontainer" +
       "&blobName=myblob");
```

上記の例では、メッセージ上のヘッダーを使用して任意の URI パラメーターを上書きできます。上記の例は、xml を使用してルートを定義する方法を示しています。

```
<route>
   <from uri="direct:start"/>
   <to uri="jclouds:blobstore:aws-s3?operation=PUT&container=mycontainer&blobName=myblob"/>
</route>
```

161.4.2. 例 2: Blob の取得/読み取り

この例では、jclouds コンポーネントを使用して Blob の contnet を読み取る方法を説明します。

```
from("direct:start")
   .to("jclouds:blobstore:aws-s3" +
       "?operation=GET" +
       "&container=mycontainer" +
       "&blobName=myblob");
```

上記の例では、メッセージ上のヘッダーを使用して任意の URI パラメーターを上書きできます。上記の例は、xml を使用してルートを定義する方法を示しています。

```
<route>
   <from uri="direct:start"/>
   <to uri="jclouds:blobstore:aws-s3?operation=PUT&container=mycontainer&blobName=myblob"/>
</route>
```

161.4.3. 例 3: Blob の使用

この例では、指定されたコンテナー下のすべての Blob を消費します。生成されたエクスチェンジには、Blob のペイロードがボディーとして含まれます。

```
    from("jclouds:blobstore:aws-s3" +
        "?container=mycontainer")
        .to("direct:next");
```

以下で示すように xml を使用して同じ目的を達成できます。

```
<route>
    <from uri="jclouds:blobstore:aws-s3?
operation=GET&container=mycontainer&blobName=myblob"/>
    <to uri="direct:next"/>
</route>
```

```
jclouds:compute:aws-ec2?
operation=CamelJcloudsCreateNode&imageId=AMI_XXXXX&locationId=eu-
west-1&group=mygroup
```

161.5. COMPUTE の使用状況についてのサンプル

以下の例は、java dsl および spring/blueprint xml での jclouds コンピュートプロデューサーの使用を示しています。

161.5.1. 例 1: 利用可能なイメージの一覧表示

```
    from("jclouds:compute:aws-ec2" +
        "&operation=CamelJCloudsListImages")
        .to("direct:next");
```

これにより、本文内にイメージの一覧が含まれるメッセージが作成されます。xml を使用して同じことを行うこともできます。

```
<route>
    <from uri="jclouds:compute:aws-ec2?operation=CamelJCloudsListImages"/>
    <to uri="direct:next"/>
</route>
```

161.5.2. 例 2: 新規ノードを作成します。

```
    from("direct:start").
    to("jclouds:compute:aws-ec2" +
        "?operation=CamelJcloudsCreateNode" +
        "&imageId=AMI_XXXXX" +
        "&locationId=XXXXX" +
        "&group=myGroup");
```

これにより、クラウドプロバイダーに新しいノードが作成されます。この場合の out メッセージは、新しく作成されたノードに関する情報（ip、hostname など）を含むメタデータのセットです。ここでは spring xml の使用と同じです。

```
<route>
    <from uri="direct:start"/>
    <to uri="jclouds:compute:aws-ec2?
operation=CamelJcloudsCreateNode&imageId=AMI_XXXXX&locationId=XXXXX&group=myGroup"/>
</route>
```

161.5.3. 例 3: 稼働中のノードでシェルスクリプトを実行する。

```
    from("direct:start").
    to("jclouds:compute:aws-ec2" +
        "?operation=CamelJcloudsRunScript" +
        "?nodeId=10" +
        "&user=ubuntu");
```

上記のサンプルは、メッセージのボディーを取得します。これには、実行されるシェルスクリプトが含まれることが予想されます。スクリプトを取得すると、指定したユーザーで実行できるようにノードに送信されます（この場合は ubuntu）。ターゲットノードは nodeId を使用して指定されます。nodeId は、ノードの作成時に取得することができます。これは、作成されるメタデータの一部であるか、または LIST_NODES 操作を実行して取得できます。

これに は、コンポーネントに渡す Compute サービスが必要で、適切な jclouds ssh 対応モジュール（jsch や sshj など）で初期化される必要があります。

ここでは spring xml の使用と同じです。

```
<route>
    <from uri="direct:start"/>
    <to uri="jclouds:compute:aws-ec2?operation=CamelJcloudsListNodes&?
nodeId=10&user=ubuntu"/>
</route>
```

161.5.4. 関連項目

jclouds に関する詳細は、こちらが便利なリソースのリストである。

jclouds Blobstore wiki

jclouds Compute wiki

第162章 JCR コンポーネント

Camel バージョン 1.3 で利用可能

jcr コンポーネントを使用すると、JCR 準拠のコンテンツリポジトリー（ Apache Jackrabbitなど）に/からノードを追加/読み取りでき、またはコンシューマーで EventListener を登録することができます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jcr</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

162.1. URI 形式

```
jcr://user:password@repository/path/to/node
```

コンシューマーの追加

Camel 2.10 以降では、JCR またはプロデューサーの EventListener としてコンシューマーを使用して、識別子でノードを読み取ることができます。

162.2. 用途

URI の repository 要素は、Camel コンテキストレジストリーで JCR Repository オブジェクトを検索するために使用されます。

162.2.1. JCR オプション

JCR コンポーネントにはオプションがありません。

JCR エンドポイントは URI 構文を使用して設定されます。

jcr:host/base

以下の path パラメーターおよびクエリーパラメーターを使用します。

162.2.2. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
host	使用される Camel レジストリーからルックアップするには javax.jcr.Repository の **必須** 名が必要です。		文字列
base	リポジトリーにアクセスする際にベースノードを取得します。		文字列

162.2.3. クエリーパラメーター（14 パラメーター）：

Name	説明	デフォルト	Type
deep (common)	isDeep が true の場合、関連付けられた親ノードが absPath か、またはそのサブグラフ内のイベントを受信します。	false	boolean
eventTypes (common)	EventTypes（javax.jcr.observation.Event.NODE_ADDED、javax.jcr.observation.Event.NODE_REMOVED など）としてエンコードされた 1 つ以上のイベントタイプの組み合わせ。		int
nodeTypeNames (common)	コンマ区切りの nodeTypeName リスト文字列が設定されると、一覧にあるノードタイプの 1 つ（またはノードタイプの 1 つのサブタイプ）を持つイベントのみを受信します。		文字列
noLocal (common)	noLocal が true の場合、リスナーが登録されたセッションによって生成されたイベントは無視されます。それ以外の場合は無視されます。	false	boolean
パスワード (common)	ログインのパスワード		文字列
sessionLiveCheckInterval (common)	各セッションのライブチェックまで待機する間隔（ミリ秒単位）。デフォルト値は 60000 ミリ秒です。	60000	Long

Name	説明	デフォルト	Type
sessionLiveCheckIntervalOn Start (common)	最初のセッションのライブチェックまで待機する間隔（ミリ秒単位）。デフォルト値は 3000 ミリ秒です。	3000	Long
username (common)	ログインのユーザー名		文字列
UUID （共通）	コンマ区切りの uuid リスト文字列が設定されている場合、関連付けられた親ノードの識別子がコンマ区切りの uuid 一覧にあるイベントのみを受信します。		文字列
workspaceName (common)	アクセスするワークスペース。指定されていない場合は、デフォルトのものが使用されます。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

JCR プロデューサーは、2.12.3 より前の Camel バージョンのメッセージヘッダーの代わりにメッセージプロパティーを使用していることに注意してください。詳細は、https://issues.apache.org/jira/browse/CAMEL-7067 を参照してください。

162.3. 例

以下のスニペットは、content リポジトリーの /home/test ノード の下に node という名前のノード

を作成します。追加のプロパティーはノードにも追加されています。my.contents.property には、送信されるメッセージのボディーが含まれる my.contents.property も追加されます。

```
from("direct:a").setHeader(JcrConstants.JCR_NODE_NAME, constant("node"))
    .setHeader("my.contents.property", body())
    .to("jcr://user:pass@repository/home/test");
```

以下のコードは、Event.NODE_ADDED および Event.NODE_REMOVED イベントのパス import-application/inbox の下に EventListener を登録し、イベントタイプ 1 および 2 の両方がマスクされ、すべての子をリッスンします。

```
<route>
    <from uri="jcr://user:pass@repository/import-application/inbox?eventTypes=3&deep=true" />
    <to uri="direct:execute-import-application" />
</route>
```

162.4. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第163章 JDBC コンポーネント

Camel バージョン 1.2 で利用可能

jdbc コンポーネントを使用すると、SQL クエリー(SELECT)および操作（INSERT、UPDATE など）がメッセージボディーに送信される JDBC 経由でデータベースにアクセスできます。このコンポーネントは、spring-jdbc を使用する SQL コンポーネントコンポーネントとは異なり、標準の JDBC API を使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jdbc</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

このコンポーネントを使用してプロデューサーエンドポイントを定義するため、from（）ステートメントで JDBC コンポーネントを使用することはできません。

163.1. URI 形式

```
jdbc:dataSourceName[?options]
```

このコンポーネントはプロデューサーエンドポイントのみをサポートします。

URI にクエリーオプションを追加するには、?option=value&option=value&...

163.2. オプション

JDBC コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
dataSource (producer)	レジストリーからデータソースを検索する代わりに DataSource インスタンスを使用します。		DataSource

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JDBC エンドポイントは、URI 構文を使用して設定されます。

```
jdbc:dataSourceName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

163.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
dataSourceName	**レジストリー** で検索する DataSource の名前。		文字列

163.2.2. クエリーパラメーター（13 パラメーター）：

Name	説明	デフォルト	Type
allowNamedParameters (producer)	クエリーで名前付きパラメーターを使用できるかどうか。	true	boolean
outputClass (producer)	outputType=SelectOne または SelectList 時に変換として使用する完全なパッケージおよびクラス名を指定します。		文字列
outputType (producer)	プロデューサーが使用する出力を決定します。	SelectList	JdbcOutputType
パラメーター（プロデューサー）	java.sql.Statement へのオプションのパラメーターです。たとえば、maxRows や fetchSize などを設定します。		マップ
readSize (producer)	ポーリングクエリーで読み取りできる行のデフォルトの最大数。デフォルト値は 0 です。		int

Name	説明	デフォルト	Type
resetAutoCommit (producer)	Camel は JDBC コネクションの autoCommit を false に設定し、ステートメントを実行した後に変更をコミットし、resetAutoCommit が true の場合、接続の autoCommit フラグをリセットします。JDBC コネクションが autoCommit フラグのリセットをサポートしていない場合は、resetAutoCommit フラグを false に設定することで、Camel は autoCommit フラグをリセットしません。XA トランザクションと併用する場合は、トランザクションマネージャーがこの tx のコミットを担当するよう false に設定する必要があります。	true	boolean
トランザクション処理 （プロデューサー）	トランザクションが使用中のかどうか。	false	boolean
useGetBytesForBlob (producer)	BLOB 列を文字列データではなくバイトとして読み取る。これは、BLOB 列をバイトとして読み取る必要がある Oracle などの特定のデータベースで必要になることがあります。	false	boolean
useHeadersAsParameters (producer)	名前付きパラメーターを指定した prepareStatementStrategy を使用するには、このオプションを true に設定します。これにより、名前付きプレースホルダーでクエリーを定義でき、クエリープレースホルダーの動的な値でヘッダーを使用できます。	false	boolean
useJDBC4ColumnNameAndLabelSemantics (producer)	列名を取得するときに JDBC 4 または JDBC 3.0 以前のセマンティックを使用するかどうかを設定します。JDBC 4.0 は columnLabel を使用して列名を取得し、JDBC 3.0 は両方の columnName または columnLabel を使用します。しかし、JDBC ドライバーの動作が異なるため、このコンポーネントの使用に問題がある場合は、このオプションを使用して JDBC ドライバーの問題を回避できます。	true	boolean
beanRowMapper (advanced)	outputClass の使用時にカスタムの org.apache.camel.component.jdbc.BeanRowMapper を使用するには、以下を行います。デフォルトの実装では、行名を小文字を下げ、アンダースコアとダッシュをスキップします。たとえば、CUST_ID は custId としてマッピングされます。		BeanRowMapper
prepareStatementStrategy (advanced)	プラグインがカスタムの org.apache.camel.component.jdbc.JdbcPrepareStatementStrategy を使用して、クエリーおよび準備済みステートメントの準備を制御できます。		JdbcPrepareStatement Strategy

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

163.3. 結果

デフォルトでは、結果は ArrayList<HashMap<String, Object>> として OUT ボディーで返されます。List オブジェクトには行の一覧が含まれ、Map オブジェクトには、String キーを列名とする各行が含まれます。outputType オプションを使用して結果を制御できます。

注記： このコンポーネントは ResultSetMetaData を取得し、マップのキーとして列名を返すことができます。

163.3.1. メッセージヘッダー

ヘッダー	説明
CamelJdbcRowCount	クエリーが **SELECT** の場合は、この OUT ヘッダーで行数のクエリーを行います。
CamelJdbcUpdateCount	クエリーが **UPDATE** の場合、更新数がこの OUT ヘッダーで返されます。
CamelGeneratedKeysRows	Camel 2.10: 生成された kets が含まれる行。
CamelGeneratedKeysRowCount	Camel 2.10: 生成されたキーが含まれるヘッダーの行数。

ヘッダー	説明
CamelJdbcColumnNames	Camel 2.11.1: ResultSet から **java.util.Set** 型として列名。
CamelJdbcParametes	Camel 2.12: **useHeadersAsParameters** が有効な場合に使用するヘッダーを持つ **java.util.Map**。

163.4. 生成される鍵

Camel 2.10 で利用可能

SQL INSERT を使用してデータを挿入すると、automatic が自動生成された鍵をサポートする可能性があります。JDBC プロデューサーに対して、ヘッダーで生成されたキーを返すように指示できます。
これには、ヘッダー CamelRetrieveGeneratedKeys=true を設定します。次に、生成されたキーは上記の表に記載されているキーと共にヘッダーとして提供されます。

この ユニットテスト で詳細を確認できます。

生成されたキーを使用しても、名前付きパラメーターと併用することはできません。

163.5. 名前付きパラメーターの使用

Camel 2.12 から利用可能

以下の指定のルートでは、プロジェクトテーブルからすべてのプロジェクトを取得します。SQL クエリーにはパラメーター :?lic と :?min の 2 つのパラメーターがあることに注意してください。
その後、Camel はこれらのパラメーターをメッセージヘッダーから検索します。上記の例では、名前付きパラメーターに定数値
を使用して 2 つのヘッダーを設定しています。

```
from("direct:projects")
  .setHeader("lic", constant("ASF"))
```

```
    .setHeader("min", constant(123))
    .setBody("select * from projects where license = :?lic and id > :?min order by id")
    .to("jdbc:myDataSource?useHeadersAsParameters=true")
```

ヘッダーの値を java.util.Map に保存し、CamelJdbcParameters キーを使用してヘッダーにマップを保存することもできます。

163.6. サンプル

以下の例では、顧客テーブルから行を取得します。

最初に、データソースを testdb として Camel レジストリーに登録します。

次に、JDBC コンポーネントにルーティングするルートを設定し、SQL が実行されます。前の手順でバインドされた testdb データソースを参照する方法に注意してください。

または、以下のように Spring で DataSource を作成することもできます。

エンドポイントを作成し、SQL クエリーを IN メッセージのボディーに追加してからエクスチェンジを送信します。クエリーの結果は OUT ボディーで返されます。

一度に ResultSet ではなく 1 行で作業したい場合は、次のような Splitter EIP を使用する必要があります。

Camel 2.13.x 以前

Camel 2.14.x 以降

```
from("direct:hello")
// here we split the data from the testdb into new messages one by one
// so the mock endpoint will receive a message per row in the table
// the StreamList option allows to stream the result of the query without creating a List of rows
// and notice we also enable streaming mode on the splitter
.to("jdbc:testdb?outputType=StreamList")
  .split(body()).streaming()
  .to("mock:result");
```

163.7. 例： 毎分データベースのポーリング

JDBC コンポーネントを使用してデータベースをポーリングする場合は、Timer や Quartz などのポーリングスケジューラーと組み合わせる必要があります。以下の例では、60 秒ごとにデータベースからデータを取得します。

```
from("timer://foo?period=60000").setBody(constant("select * from
customer")).to("jdbc:testdb").to("activemq:queue:customers");
```

163.8. サンプル： データソース間のデータ移動

一般的なユースケースは、データをクエリーし、処理して別のデータソース（ETL 操作）に移動することです。以下の例では、1 時間ごとにソーステーブルから新しい顧客レコードを取得し、それらをフィルター/変換し、送信先テーブルに移動します。

```
from("timer://MoveNewCustomersEveryHour?period=3600000")
    .setBody(constant("select * from customer where create_time > (sysdate-1/24)"))
    .to("jdbc:testdb")
    .split(body())
        .process(new MyCustomerProcessor()) //filter/transform results as needed
        .setBody(simple("insert into processed_customer
values('${body[ID]}','${body[NAME]}')"))
        .to("jdbc:testdb");
```

163.9. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- SQL

第164章 JETTY 9 コンポーネント

Camel バージョン 1.2 で利用可能

警告

プロデューサーは非推奨となり、使用されません。jetty をコンシューマーとして使用することのみを推奨します（例： jetty）。

jetty コンポーネントは、HTTP リクエストの使用および生成のための HTTP ベースのエンドポイントを提供します。つまり、Jetty コンポーネントは単純な Web サーバーとして機能します。
Jetty を http クライアントとして使用することもできます。つまり、Camel とプロデューサーとして使用することもできます。

ストリーム

コードはユニットテストの一部であるため、 アサート コールが表示されます。Jetty はストリームベースであるため、受信する入力はストリームとして Camel に送信されます。つまり、は 1 回 のみストリームのコンテンツを読み取ることができます。
メッセージボディーが空であるか、または Exchange.HTTP_RESPONSE_CODE データに複数回アクセスする必要がある場合（マルチキャストまたは再配信エラー処理を行うなど）、Stream キャッシュを使用するか、メッセージボディーを複数回再読み取りできる String に変換する必要があります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jetty</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

164.1. URI 形式

```
jetty:http://hostname[:port][/resourceUri][?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&...

164.2. オプション

Jetty 9 コンポーネントは、以下に示す 33 オプションをサポートします。

Name	説明	デフォルト	Type
sslKeyPassword (security)	キーストア内の証明書のキーエントリーにアクセスするために使用されるキーパスワード（キーストアコマンドの -keypass オプションに提供されるパスワードと同じもの）。		文字列
sslPassword (security)	キーストアファイルへのアクセスに必要な ssl パスワード（キーストアコマンドの -storepass オプションに提供されるパスワードと同じ）。		文字列
キーストア （セキュリティー）	キーエントリーに Jetty サーバーの独自の X.509 証明書が含まれる Java キーストアファイルの場所を指定します。		文字列
errorHandler (advanced)	このオプションは、Jetty サーバーが使用する ErrorHandler を設定するために使用されます。		ErrorHandler
sslSocketConnectors (security)	ポート番号固有の SSL コネクターごとに含まれるマップ。		マップ
socketConnectors (security)	ポート番号固有の HTTP コネクターごとに含まれるマップ。sslSocketConnectors と同じ原則を使用します。		マップ
httpClientMinThreads (producer)	HttpClient スレッドプールで最小スレッド数の値を設定します。最小サイズと最大サイズの両方を設定する必要があることに注意してください。		整数
httpClientMaxThreads (producer)	HttpClient スレッドプールの最大スレッド数に値を設定します。最小サイズと最大サイズの両方を設定する必要があることに注意してください。		整数
minThreads （コンシューマー）	サーバースレッドプールの最小スレッド数の値を設定します。最小サイズと最大サイズの両方を設定する必要があることに注意してください。		整数

Name	説明	デフォルト	Type
maxThreads （コンシューマー）	サーバースレッドプールの最大スレッド数の値を設定するには、以下を行います。最小サイズと最大サイズの両方を設定する必要があることに注意してください。		整数
threadPool (consumer)	サーバーにカスタムスレッドプールを使用するには、以下を行います。このオプションは、特別な場合のみ使用してください。		ThreadPool
enableJmx (common)	このオプションが true の場合、このエンドポイントに対して Jetty JMX サポートが有効になります。	false	boolean
jettyHttpBinding (advanced)	プロデューサーに対して応答を記述する方法をカスタマイズするために使用されるカスタム org.apache.camel.component.jetty.JettyHttpBinding を使用します。		JettyHttpBinding
httpBinding (advanced)	使用しない場合は、代わりに JettyHttpBinding を使用してください。		HttpBinding
httpConfiguration (advanced)	Jetty コンポーネントは HttpConfiguration を使用しません。		HttpConfiguration
mbContainer (advanced)	JMX が有効になっている場合、既存の設定済みの org.eclipse.jetty.jmx.MBeanContainer を使用するには、Jetty が mbeans の登録に使用します。		MBeanContainer
sslSocketConnector Properties （セキュリティー）	一般的な SSL コネクタープロパティーが含まれるマップ。		マップ
SocketConnector Properties （セキュリティー）	一般的な HTTP コネクタープロパティーが含まれるマップ。sslSocketConnectorProperties と同じ原則を使用します。		マップ
continuationTimeout (consumer)	Jetty をコンシューマー（サーバー）として使用する際のタイムアウトをミリ秒単位で設定できます。デフォルトでは Jetty は 30000 を使用します。= 0 の値を使用すると期限切れになりません。タイムアウトが発生すると、リクエストは期限切れになり、Jetty は http エラー 503 をクライアントに返します。このオプションは、Jetty を非同期ルーティングエンジンで使用する場合にのみ使用します。	30000	Long
useContinuation (consumer)	Jetty サーバーに Jetty 継続を使用するかどうか。	true	boolean

Name	説明	デフォルト	Type
sslContextParameters (security)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用の有効化	false	boolean
responseBufferSize (common)	Jetty コネクターで応答バッファーサイズのカスタム値を設定できます。		整数
requestBufferSize (common)	Jetty コネクターでリクエストバッファーサイズのカスタム値を設定できます。		整数
requestHeaderSize (common)	Jetty コネクターで要求ヘッダーサイズのカスタム値を設定できます。		整数
responseHeaderSize (common)	Jetty コネクターで応答ヘッダーサイズのカスタム値を設定できます。		整数
proxyHost (proxy)	http プロキシーを使用してホスト名を設定するには、以下を行います。		文字列
proxyPort (proxy)	http プロキシーを使用してポート番号を設定します。		整数
useXForwardedFor Header (common)	HttpServletRequest.getRemoteAddr で X-Forwarded-For ヘッダーを使用します。	false	boolean
sendServerVersion (consumer)	オプションが true の場合、jetty サーバーはリクエストを送信するクライアントに date ヘッダーを送信します。注記： 他の camel-jetty エンドポイントが同じポートを共有していることを確認してください。そうしないと、このオプションは想定どおりに機能しない可能性があります。	true	boolean
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用する場合に java のシリアライズを許可するかどうか。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Jetty 9 エンドポイントは、URI 構文を使用して設定します。

```
jetty:httpUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

164.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
httpUri	**必須**。呼び出す HTTP エンドポイントの URL。		URI

164.2.2. クエリーパラメーター（54 パラメーター）：

Name	説明	デフォルト	Type
チャンク （共通）	このオプションが false の場合、サーブレットは HTTP ストリーミングを無効にし、応答に content-length ヘッダーを設定します。	true	boolean

Name	説明	デフォルト	Type
disableStreamCache (common)	Servlet からの raw 入力ストリームがキャッシュされているかどうかを決定します（Camel はストリームをメモリー内/オーバーフローでファイル、ストリームキャッシング）キャッシュに読み取ります。デフォルトでは、Camel は Servlet 入力ストリームをキャッシュして、複数回ロードし、Camel がストリームからすべてのデータを取得できるようにします。ただし、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。たとえば、ファイルまたは他の永続ストアに直接ストリーミングする場合などに、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。DefaultHttpBinding は、ストリームの読み取りを複数回サポートするために、このオプションが false の場合は、リクエスト入力ストリームをストリームキャッシュにコピーし、メッセージボディーに配置します。Servlet を使用してエンドポイントをブリッジ/プロキシーする場合、このオプションを有効にしてパフォーマンスを向上することを検討してください。メッセージペイロードを複数回読み取る必要がない場合は、このオプションを有効にします。http/http4 プロデューサーは、デフォルトでは応答本体ストリームをキャッシュします。このオプションを true に設定すると、プロデューサーは応答ボディーストリームをキャッシュしませんが、応答ストリームをメッセージボディーとして使用します。	false	boolean
enableMultipartFilter (common)	Jetty org.eclipse.jetty.servlets.MultiPartFilter が有効かどうか。複数のパートリクエストもプロキシー/ブリッジされるようにするには、エンドポイントのブリッジ時にこの値を false に設定する必要があります。	false	boolean
headerFilterStrategy (common)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
transferException (common)	有効で Exchange がコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object のコンテンツタイプとして応答でシリアライズされたかどうか。プロデューサー側では、例外は HttpOperationFailedException ではなくデシリアライズされ、そのままスローされます。原因となる例外はシリアライズされている必要があります。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean

Name	説明	デフォルト	Type
httpBinding (common)	カスタムの HttpBinding を使用して Camel メッセージと HttpClient 間のマッピングを制御する場合。		HttpBinding
非同期 （コンシューマー）	非同期モードで動作するようにコンシューマーを設定する	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
continuationTimeout (consumer)	Jetty をコンシューマー（サーバー）として使用する際のタイムアウトをミリ秒単位で設定できます。デフォルトでは Jetty は 30000 を使用します。= 0 の値を使用すると期限切れになりません。タイムアウトが発生すると、リクエストは期限切れになり、Jetty は http エラー 503 をクライアントに返します。このオプションは、Jetty を非同期ルーティングエンジンで使用する場合にのみ使用します。	30000	Long
enableCORS (consumer)	オプションが true の場合、Jetty サーバーは CORS をそのままサポートする CrossOriginFilter を設定します。	false	boolean
enableJmx (consumer)	このオプションが true の場合、このエンドポイントに対して Jetty JMX サポートが有効になります。詳細は、「Jetty JMX サポート」を参照してください。	false	boolean
httpMethodRestrict (consumer)	HttpMethod が一致する場合にのみ消費（GET/POST/PUT など）を許可するために使用されます。複数のメソッドはカンマで区切って指定できます。		文字列
matchOnUriPrefix (consumer)	完全一致がない場合、コンシューマーが URI プレフィックスに一致することでターゲットコンシューマーの検索を試行するかどうか。	false	boolean
responseBufferSize (consumer)	javax.servlet.ServletResponse でカスタムバッファーサイズを使用します。		整数

Name	説明	デフォルト	Type
sendDateHeader (consumer)	オプションが true の場合、jetty サーバーはリクエストを送信するクライアントに date ヘッダーを送信します。注記： 他の camel-jetty エンドポイントが同じポートを共有していることを確認してください。そうしないと、このオプションは想定どおりに機能しない可能性があります。	false	boolean
sendServerVersion (consumer)	オプションが true の場合、jetty は、リクエストを送信するクライアントに jetty バージョン情報が含まれるサーバーヘッダーを送信します。注記： 他の camel-jetty エンドポイントが同じポートを共有していることを確認してください。そうしないと、このオプションは想定どおりに機能しない可能性があります。	true	boolean
sessionSupport (consumer)	Jetty のサーバー側でセッションマネージャーを有効にするかどうかを指定します。	false	boolean
useContinuation (consumer)	Jetty サーバーに Jetty 継続を使用するかどうか。		ブール値
eagerCheckContentAvailable (consumer)	content-length ヘッダーが 0 の場合に、HTTP リクエストにコンテンツがあるかどうかをアクティブにチェックするかどうか。これは、HTTP クライアントがストリーミングデータを送信しない場合に有効にすることができます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
filterInitParameters (consumer)	フィルター init パラメーターの設定。これらのパラメーターは、jetty サーバーを起動する前にフィルターリストに適用されます。		マップ
filtersRef (consumer)	リストに配置され、レジストリーで見つけることができるカスタムフィルターの使用を許可します。複数の値はコンマで区切ることができます。		文字列

Name	説明	デフォルト	Type
ハンドラー（コンシューマー）	レジストリーで検索するためのハンドラーインスタンスのカンマ区切りのセットを指定します。これらのハンドラーは Jetty サーブレットコンテキストに追加されます（セキュリティーの追加など）。重要：同じポート番号を使用して異なる Jetty エンドポイントで異なるハンドラーを使用することはできません。ハンドラーはポート番号に関連付けられます。異なるハンドラーが必要な場合は、異なるポート番号を使用します。		文字列
httpBindingRef (consumer)	リモートサーバーからの応答に失敗した場合に HttpOperationFailedException のスローを無効にするオプション。**これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。**		文字列
multipartFilter (consumer)	カスタムのマルチパートフィルターの使用を許可します。注記： multipartFilterRef を設定すると、enableMultipartFilter の値が true に強制的に設定されます。		フィルター
multipartFilterRef (consumer)	**非推奨**: カスタムのマルチパートフィルターの使用を許可します。注記： multipartFilterRef を設定すると、enableMultipartFilter の値が true に強制的に設定されます。		文字列
optionsEnabled (consumer)	このサーブレットコンシューマーに HTTP OPTIONS を有効にするかどうかを指定します。デフォルトでは OPTIONS はオフになっています。	false	boolean
traceEnabled (consumer)	このサーブレットコンシューマーに対して HTTP TRACE を有効にするかどうかを指定します。デフォルトでは、TRACE はオフになっています。	false	boolean
bridgeEndpoint (producer)	オプションが true の場合、HttpProducer は Exchange.HTTP_URI ヘッダーを無視し、リクエストにエンドポイントの URI を使用します。また、ththExceptionOnFailure オプションを false に設定して、HttpProducer がすべての障害応答を返せるようにすることもできます。	false	boolean
connectionClose (producer)	Connection Close ヘッダーを HTTP 要求に追加する必要があるかどうかを指定します。デフォルトでは connectionClose は false です。	false	boolean
cookieHandler (producer)	HTTP セッションを維持するためのクッキーハンドラーの設定		CookieHandler

Name	説明	デフォルト	Type
copyHeaders (producer)	このオプションが true の場合、コピーストラテジーに従って IN エクスチェンジヘッダーが OUT エクスチェンジヘッダーにコピーされます。これを false に設定し、HTTP 応答からのヘッダーのみを含めることができます（IN ヘッダーは伝播しません）。	true	boolean
httpClientMaxThreads (producer)	HttpClient スレッドプールの最大スレッド数に値を設定します。この設定は、コンポーネントレベルに設定された設定を上書きします。最小サイズと最大サイズの両方を設定する必要があることに注意してください。デフォルトを Jettys スレッドプールで使用される max 254 スレッドに設定しないと、Jettys スレッドプールで使用される最大 254 スレッドに設定されます。	254	整数
httpClientMinThreads (producer)	HttpClient スレッドプールで最小スレッド数の値を設定します。この設定は、コンポーネントレベルに設定された設定を上書きします。最小サイズと最大サイズの両方を設定する必要があることに注意してください。デフォルトに設定しないと、Jettys スレッドプールで使用される 8 つのスレッドがデフォルトに設定されます。	8	整数
httpMethod (producer)	使用する HTTP メソッドを設定します。HttpMethod ヘッダーが設定されている場合は、このオプションをオーバーライドできません。		HttpMethods
ignoreResponseBody (producer)	このオプションが true の場合、http プロデューサーは応答本体を読み取りせず、入力ストリームをキャッシュします。	false	boolean
preserveHostHeader (producer)	オプションが true の場合、HttpProducer は Host ヘッダーを現在のエクスチェンジ Host ヘッダーに含まれる値に設定します。これは、ダウンストリームサーバーが受け取る Host ヘッダーを使用してアップストリームクライアントが呼び出した URL を反映させたいリバースプロキシーアプリケーションで有用です。これにより、Host ヘッダーを使用するアプリケーションがプロキシーされるサービスの正確な URL を生成することができます。	false	boolean
throwExceptionOnFailure (producer)	リモートサーバーからの応答に失敗した場合に HttpOperationFailedException のスローを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean

Name	説明	デフォルト	Type
httpClient (producer)	このエンドポイントによって作成されたすべてのプロデューサーに使用する共有 HttpClient を設定します。デフォルトでは、各プロデューサーは新しい http クライアントを使用し、共有しません。重要：使用されていないときにクライアントの停止など、共有クライアントのライフサイクルを処理するようにしてください。Camel はクライアントで start メソッドを呼び出して、このエンドポイントがプロデューサーの作成時に起動されるようにします。このオプションは、特別な場合のみ使用してください。		HttpClient
httpClientParameters (producer)	Jetty の HttpClient の設定。たとえば、httpClient.idleTimeout=30000 を設定すると、アイドルタイムアウトが 30 秒に設定されます。また、httpClient.timeout=30000 は、長時間実行されるリクエスト/レスポンス呼び出しが長い場合にタイムアウトするタイムアウトを 30 秒に設定します。		マップ
jettyBinding (producer)	プロデューサーの応答の記述方法をカスタマイズするために使用されるカスタム JettyHttpBinding を使用するには、以下を行います。		JettyHttpBinding
jettyBindingRef (producer)	**非推奨**: プロデューサーの応答の記述方法をカスタマイズするために使用されるカスタム JettyHttpBinding を使用してください。		文字列
okStatusCodeRange (producer)	正常な応答とみなされるステータスコード。値は含まれます。コンマで区切られた複数の範囲を定義できます（例： 200-204,209,301-304）。各範囲は、1 つの数字またはダッシュを含む from から でなければなりません。	200-299	文字列
urlRewrite (producer)	**非推奨** の org.apache.camel.component.http.UrlRewrite への参照。ブリッジ/プロキシーエンドポイントの実行時に URL を書き換えることができるようになりました。詳細は、http://camel.apache.org/urlrewrite.htmlを参照してください。		UrlRewrite
mapHttpMessageBody (advanced)	このオプションが true の場合、エクスチェンジの IN エクスチェンジボディーは HTTP ボディーにマッピングされます。false に設定すると HTTP マッピングが回避されます。	true	boolean

Name	説明	デフォルト	Type
mapHttpMessageFormUrlEncodedBody (advanced)	このオプションが true の場合、エクスチェンジの IN Exchange Form Encoded body が HTTP にマッピングされます。これを false に設定すると、HTTP Form Encoded body マッピングを回避します。	true	boolean
mapHttpMessageHeaders (advanced)	このオプションが true の場合、エクスチェンジの IN エクスチェンジヘッダーは HTTP ヘッダーにマッピングされます。false に設定すると、HTTP ヘッダーのマッピングが回避されます。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
proxyAuthScheme (proxy)	使用するプロキシー認証スキーム		文字列
proxyHost (proxy)	使用するプロキシーホスト名		文字列
proxyPort (proxy)	使用するプロキシーポート		int
authHost (security)	NTML で使用する認証ホスト		文字列
sslContextParameters (security)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters

164.3. メッセージヘッダー

Camel は HTTP コンポーネントと同じメッセージヘッダーを使用します。Camel 2.2 から、(Exchange.HTTP_CHUNKED,CamelHttpChunked)ヘッダーも使用して camel-jetty コンシューマーで chuched エンコーディングを有効または無効にします。

Camel は すべて の request.parameter および request.headers も設定します。たとえば、URL http://myserver/myserver?orderid=123 を持つクライアントリクエストでは、エクスチェンジに value 123 の orderid という名前のヘッダーが含まれます。

Camel 2.0 以降では、メッセージヘッダーから Get Method からだけでなく、他の HTTP メソッドからも request.parameter を取得できます。

164.4. 用途

Jetty コンポーネントは、コンシューマーおよびプロデューサーエンドポイントの両方をサポートします。他の HTTP エンドポイントを生成する別のオプションとして、HTTP コンポーネントを使用することができます。

164.5. プロデューサーの例

警告

プロデューサーは非推奨となり、使用されません。jetty をコンシューマーとして使用することのみを推奨します（例： jetty）。

以下は、HTTP リクエストを既存の HTTP エンドポイントに送信する基本的な例です。

in Java DSL

```
from("direct:start").to("jetty://http://www.google.com");
```

または Spring XML で行う

```
<route>
  <from uri="direct:start"/>
  <to uri="jetty://http://www.google.com"/>
<route>
```

164.6. コンシューマーの例

以下の例では、http://localhost:8080/myapp/myservice で HTTP サービスを公開するルートを定義します。

ローカルの使用

URL で localhost を指定すると、Camel はローカル TCP/IP ネットワークインターフェースでのみエンドポイントを公開するため、操作するマシン外からはアクセスできません。

特定のネットワークインターフェースで Jetty エンドポイントを公開する必要がある場合は、このインターフェースの数値の IP アドレスをホストとして使用する必要があります。すべてのネットワークインターフェースで Jetty エンドポイントを公開する必要がある場合は、0.0.0.0 アドレスを使用する必要があります。

URI プレフィックス全体をリッスンするには、「How do I let Jetty match wildcards」を参照してください。

HTTP によってルートを公開し、すでにサーブレットがある場合は、代わりに Servlet Transport を参照する必要があります。

このビジネスロジックは、HTTP リクエストのコンテンツにアクセスして応答を返す MyBookService クラスに実装されます。
注記： コードはユニットテストの一部であるため、アサート 呼び出しはこの例では表示されます。

以下の例は、URI パラメーター、1 つ、エンドポイント、mock: one、およびその他のすべてのリクエストを mock: other にルーティングするコンテンツベースのルートを示しています。

そのため、クライアントが HTTP リクエスト http://serverUri?one=hello を送信すると、Jetty コンポーネントは HTTP リクエストパラメーター 1 つ をエクスチェンジの in.header にコピーします。次に、Simple 言語を使用して、このヘッダーが含まれるエクスチェンジを特定のエンドポイントおよび他のエンドポイントにルーティングすることができます。Simple （OGNLなど）よりも強力な言語を使用した場合は、パラメーター値をテストし、ヘッダー値に基づいてルーティングを行うこともできます。

164.7. セッションサポート

セッションサポートオプション sessionSupport を使用して HttpSession オブジェクトを有効にし、エクスチェンジの処理中にセッションオブジェクトにアクセスできます。たとえば、以下のルートはセッションを有効にします。

```
<route>
  <from uri="jetty:http://0.0.0.0/myapp/myservice/?sessionSupport=true"/>
  <processRef ref="myCode"/>
<route>
```

myCode Processor は Spring Bean 要素によってインスタンス化できます。

```
<bean id="myCode"class="com.mycompany.MyCodeProcessor"/>
```

プロセッサーの実装は、以下のように HttpSession にアクセスできます。

```
public void process(Exchange exchange) throws Exception {
    HttpSession session = exchange.getIn(HttpMessage.class).getRequest().getSession();
    ...
}
```

164.8. SSL サポート(HTTPS)

JSSE 設定ユーティリティーの使用

Camel 2.8 より、Jetty コンポーネントは Camel JSSE 設定ユーティリティーを介して SSL/TLS 設定をサポートします。 このユーティリティーは、エンドポイントおよびコンポーネントレベルで記述し、設定する必要のあるコンポーネント固有のコードの量を大幅に削減します。 以下の例は、Jetty コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

JettyComponent jettyComponent = getContext().getComponent("jetty",
JettyComponent.class);
jettyComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
    <camel:keyManagers
       keyPassword="keyPassword">
      <camel:keyStore
         resource="/users/home/server/keystore.jks"
```

```
        password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="jetty:https://127.0.0.1/mail/?sslContextParameters=#sslContextParameters"/>
...
```

Jetty の直接的な設定

Jetty は、追加設定なしで SSL サポートを提供します。Jetty を SSL モードで実行できるようにするには、https:// prefix--for を使用して URI をフォーマットします。

```
<from uri="jetty:https://0.0.0.0/myapp/myservice/"/>
```

Jetty は、正しい SSL 証明書をロードするため、キーストアのロード場所と、使用するパスワードを知っておく必要があります。以下の JVM システムプロパティーを設定します。

until Camel 2.2

- jetty.ssl.keystore は、キー エントリー に Jetty サーバー自体 X.509 証明書が含まれる Java キーストアファイルの場所を指定します。キーエントリーは、X.509 証明書（パブリック鍵）とそれに関連する秘密鍵も格納します。
- jetty.ssl.password キーストアファイルへのアクセスに必要なストアパスワード（キー ストア コマンドの -storepass オプションに提供されるパスワードと同じ）
- jetty.ssl.keypassword キーストアの証明書のキーエントリーへのアクセスに使用されるキーパスワード（キー ストア コマンドの -keypass オプションに提供されるパスワードと同じもの）。

Camel 2.3 以降

- org.eclipse.jetty.ssl.keystore は、キー エントリー に Jetty サーバーの独自の X.509 証明書が含まれる Java キーストアファイルの場所を指定します。キーエントリーは、X.509 証明書（パブリック鍵）とそれに関連する秘密鍵も格納します。
- org.eclipse.jetty.ssl.password キーストアファイルへのアクセスに必要なストアパスワー

ド（キー ストア コマンドの **-storepass** オプションに指定されたパスワードと同じもの）。

- **org.eclipse.jetty.ssl.keypassword** キーパスワードは、キーストア内の証明書のキーエントリーにアクセスするために使用されます（キー ストア コマンドの **-keypass** オプションに提供されるパスワードと同じです）。

Jetty エンドポイントで SSL を設定する方法は、「Jetty Site: http://docs.codehaus.org/display/JETTY/How+to+configure+SSL」を参照してください。

一部の SSL プロパティーは Camel によって直接公開されませんが、Camel は基礎となる SslSocketConnector を公開します。さまざまな Camel バージョンには若干の違いがあります。

Camel 2.2 まで

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
  <property name="sslSocketConnectors">
    <map>
      <entry key="8043">
        <bean class="org.mortbay.jetty.security.SslSocketConnector">
          <property name="password"value="..."/>
          <property name="keyPassword"value="..."/>
          <property name="keystore"value="..."/>
          <property name="needClientAuth"value="..."/>
          <property name="truststore"value="..."/>
        </bean>
      </entry>
    </map>
  </property>
</bean>
```

Camel 2.3, 2.4

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
  <property name="sslSocketConnectors">
    <map>
      <entry key="8043">
        <bean class="org.eclipse.jetty.server.ssl.SslSocketConnector">
          <property name="password"value="..."/>
          <property name="keyPassword"value="..."/>
          <property name="keystore"value="..."/>
          <property name="needClientAuth"value="..."/>
          <property name="truststore"value="..."/>
        </bean>
      </entry>
```

```
        </map>
    </property>
</bean>
```

** Camel 2.5 から SslSelectChannelConnector を使用するよう切り替えます **

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
    <property name="sslSocketConnectors">
        <map>
            <entry key="8043">
                <bean class="org.eclipse.jetty.server.ssl.SslSelectChannelConnector">
                    <property name="password"value="..."/>
                    <property name="keyPassword"value="..."/>
                    <property name="keystore"value="..."/>
                    <property name="needClientAuth"value="..."/>
                    <property name="truststore"value="..."/>
                </bean>
            </entry>
        </map>
    </property>
</bean>
```

上記のマップのキーとして使用する値は、リッスンする Jetty を設定するポートです。

164.8.1. 一般的な SSL プロパティーの設定

Camel 2.5 で利用可能

（上記のように）ポート番号固有の SSL ソケットコネクターごとに、すべての SSL ソケットコネクターに適用される一般的なプロパティーを設定できるようになりました（これは、ポート番号をエントリーとして上記のように明示的に設定されていません）。

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
    <property name="sslSocketConnectorProperties">
        <map>
            <entry key="password"value="..."/>
            <entry key="keyPassword"value="..."/>
            <entry key="keystore"value="..."/>
            <entry key="needClientAuth"value="..."/>
            <entry key="truststore"value="..."/>
        </map>
    </property>
</bean>
```

164.8.2. X509Certificate への参照を取得する方法

jetty は、以下のようにコードからアクセスできる HttpServletRequest に証明書への参照を保存します。

```
HttpServletRequest req = exchange.getIn().getBody(HttpServletRequest.class);
X509Certificate cert = (X509Certificate)
req.getAttribute("javax.servlet.request.X509Certificate")
```

164.8.3. 一般的な HTTP プロパティーの設定

Camel 2.5 で利用可能

（上記のように）ポート番号固有の HTTP ソケットコネクターごとに、すべての HTTP ソケットコネクターに適用される一般的なプロパティーを設定できるようになりました（これは、ポート番号をエントリーとして上記のように明示的に設定されていません）。

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
  <property name="socketConnectorProperties">
    <map>
      <entry key="acceptors" value="4"/>
      <entry key="maxIdleTime" value="300000"/>
    </map>
  </property>
</bean>
```

164.8.4. HttpServletRequest.getRemoteAddr（）を使用した X-Forwarded-For ヘッダーの取得

HTTP リクエストが Apache サーバーによって処理され、mod_proxy で jetty に転送された場合、元のクライアント IP アドレスは X-Forwarded-For ヘッダーにあり、HttpServletRequest.getRemoteAddr（）は Apache プロキシーのアドレスを返します。

Jetty には転送されたプロパティーがあり、X-Forwarded-For から値を取得し、HttpServletRequest remoteAddr プロパティーに配置します。 このプロパティーはエンドポイント設定から直接は利用できませんが、socketConnectors プロパティーを使用して簡単に追加できます。

```
<bean id="jetty" class="org.apache.camel.component.jetty.JettyHttpComponent">
  <property name="socketConnectors">
    <map>
      <entry key="8080">
        <bean class="org.eclipse.jetty.server.nio.SelectChannelConnector">
          <property name="forwarded" value="true"/>
        </bean>
      </entry>
```

```
        </map>
    </property>
</bean>
```

これは、既存の Apache サーバーがドメインの TLS 接続を処理し、それらを内部にアプリケーションサーバーにプロキシーする場合に役立ちます。

164.9. HTTP ステータスコードを返すデフォルトの動作

HTTP ステータスコードのデフォルト動作は、応答の記述方法を処理し、HTTP ステータスコードを設定する org.apache.camel.component.http.DefaultHttpBinding クラスによって定義されます。

エクスチェンジが正常に処理された場合は、200 HTTP ステータスコードが返されます。
エクスチェンジが例外で失敗した場合には、500 HTTP ステータスコードが返され、stacktrace がボディーで返されます。返す HTTP ステータスコードを指定する場合は、OUT メッセージの Exchange.HTTP_RESPONSE_CODE ヘッダーにコードを設定します。

164.10. CUSTOMIZING HTTPBINDING

デフォルトでは、Camel は org.apache.camel.component.http.DefaultHttpBinding を使用して応答の書き込み方法を処理します。必要に応じて、独自の HttpBinding クラスを実装するか、または DefaultHttpBinding を拡張し、適切なメソッドを上書きすることでこの動作をカスタマイズできます。

以下の例は、例外が返される方法を変更するために DefaultHttpBinding をカスタマイズする方法を示しています。

その後、バインディングのインスタンスを作成して、以下のように Spring レジストリーに登録することができます。

```
<bean id="mybinding"class="com.mycompany.MyHttpBinding"/>
```

ルートを定義するときにこのバインディングを参照できます。

```
<route><from uri="jetty:http://0.0.0.0:8080/myapp/myservice?httpBindingRef=mybinding"/><to
uri="bean:doSomething"/></route>
```

164.11. JETTY ハンドラーおよびセキュリティー設定

エンドポイントで Jetty ハンドラーの一覧を設定できます。これは、高度な Jetty セキュリティー機能を有効にするのに役立ちます。これらのハンドラーは、以下のように Spring XML で設定されます。

```
<-- Jetty Security handling -->
<bean id="userRealm" class="org.mortbay.jetty.plus.jaas.JAASUserRealm">
    <property name="name" value="tracker-users"/>
    <property name="loginModuleName" value="ldaploginmodule"/>
</bean>

<bean id="constraint" class="org.mortbay.jetty.security.Constraint">
    <property name="name" value="BASIC"/>
    <property name="roles" value="tracker-users"/>
    <property name="authenticate" value="true"/>
</bean>

<bean id="constraintMapping" class="org.mortbay.jetty.security.ConstraintMapping">
    <property name="constraint" ref="constraint"/>
    <property name="pathSpec" value="/*"/>
</bean>

<bean id="securityHandler" class="org.mortbay.jetty.security.SecurityHandler">
    <property name="userRealm" ref="userRealm"/>
    <property name="constraintMappings" ref="constraintMapping"/>
</bean>
```

Camel 2.3 以降では、 Jetty ハンドラーの一覧を以下のように設定できます。

```
<-- Jetty Security handling -->
<bean id="constraint" class="org.eclipse.jetty.http.security.Constraint">
    <property name="name" value="BASIC"/>
    <property name="roles" value="tracker-users"/>
    <property name="authenticate" value="true"/>
</bean>

<bean id="constraintMapping" class="org.eclipse.jetty.security.ConstraintMapping">
    <property name="constraint" ref="constraint"/>
    <property name="pathSpec" value="/*"/>
</bean>

<bean id="securityHandler" class="org.eclipse.jetty.security.ConstraintSecurityHandler">
    <property name="authenticator">
        <bean class="org.eclipse.jetty.security.authentication.BasicAuthenticator"/>
    </property>
    <property name="constraintMappings">
        <list>
            <ref bean="constraintMapping"/>
        </list>
    </property>
</bean>
```

次に、以下のようにエンドポイントを定義できます。

```
from("jetty:http://0.0.0.0:9080/myservice?handlers=securityHandler")
```

追加のハンドラーが必要な場合は、ハンドラー オプションを Bean ID のカンマ区切りリストに設定します。

164.12. カスタム HTTP 500 応答メッセージを返す方法

デフォルトのリプライメッセージ Camel Jetty 応答ではなく、問題が発生した場合はカスタムリプライメッセージを返すことがあります。
カスタムの HttpBinding を使用してメッセージマッピングを制御できますが、多くの場合、Camel の例外句を使用してカスタムリプライメッセージを構築する方が簡単です。たとえば、以下の ように Dude を返すと HTTP エラーコード 500 で問題が発生しまし た。

164.13. マルチパートフォームのサポート

Camel 2.3.0 以降、camel-jetty が追加設定なしでマルチパートフォームをサポート。送信されたフォームデータがメッセージヘッダーにマッピングされます。camel-jetty は、アップロードした各ファイルのアタッチメントを作成します。ファイル名は、添付ファイルの名前にマッピングされます。コンテンツタイプは、添付ファイル名のコンテンツタイプとして設定されます。ここ に例が記載されています。

注記： getName（）はバージョン 2.5 以降のバージョンで示されているように機能します。以前のバージョンでは、代わりに添付の一時ファイル名を受信します。

164.14. JETTY JMX サポート

Camel 2.3.0 以降、camel-jetty は、エンドポイント設定が優先されるコンポーネントおよびエンドポイントレベルで Jetty の JMX 機能の有効化をサポートします。このコンポーネントで JMX サポートを有効にするには、JMX を Camel コンテキスト内で有効にする必要があります。これは、コンポーネントが Camel コンテキストに登録されている MBeanServer への参照を持つ Jetty を提供するためです。camel-jetty コンポーネントは特定のプロトコル/ホスト/ポートのペアに Jetty リソースをキャッシュして再利用するため、この設定オプションは、プロトコル/ホスト/ポートのペアを使用するために最初のエンドポイントの作成時にのみ評価されます。たとえば、以下の XML フラグメントから作成された 2 つのルートがある場合、JMX サポートは "https://0.0.0.0" をリッスンするすべてのエンドポイントに対して有効のままとなります。

```
<from uri="jetty:https://0.0.0.0/myapp/myservice1/?enableJmx=true"/>
```

```
<from uri="jetty:https://0.0.0.0/myapp/myservice2/?enableJmx=false"/>
```

camel-jetty コンポーネントは、Jetty MBeanContainer の直接的な設定も提供します。Jetty は MBean 名を動的に作成します。Camel コンテキストの外部で Jetty の別のインスタンスを実行し、インスタンス間で同じ MBeanServer を共有する場合は、Jetty MBean の登録時に名前の競合を回避するために、両方のインスタンスに同じ MBeanContainer への参照を指定することができます。

164.15. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*
- *HTTP*

第165章 JGROUPS コンポーネント

Camel バージョン 2.13 から利用可能

JGroups は信頼できるマルチキャスト通信のためのツールキットです。jgroups: コンポーネントは Camel インフラストラクチャーと JGroups クラスター間のメッセージの交換を提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache-extras.camel-extra</groupId>
  <artifactId>camel-jgroups</artifactId>
  <!-- use the same version as your Camel core version -->
  <version>x.y.z</version>
</dependency>
```

Camel 2 .13.0 以降、JGroups コンポーネントは Apache Camel の umbrella 下で Camel Extra から移動されました。Camel 2 .13.0 以降を使用している場合は、代わりに以下の POM エントリーを使用してください。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jgroups</artifactId>
  <!-- use the same version as your Camel core version -->
  <version>x.y.z</version>
</dependency>
```

165.1. URI 形式

```
jgroups:clusterName[?options]
```

clusterName は、コンポーネントが接続する JGroups クラスターの名前を表します。

165.2. オプション

JGroups コンポーネントは、以下に挙げる 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
チャンネル（共通）	使用するチャネル		jChannel
channelProperties (common)	エンドポイントによって使用される JChannel の設定プロパティーを指定します。		文字列
enableViewMessages (consumer)	true に設定すると、コンシューマーエンドポイントは（org.jgroups.Message インスタンスだけでなく）org.jgroups.View メッセージも受信します。デフォルトでは、通常のメッセージのみがエンドポイントによって消費されます。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JGroups エンドポイントは URI 構文を使用します。

```
jgroups:clusterName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

165.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
clusterName	***必須***。コンポーネントが接続する JGroups クラスターの名前。		文字列

165.2.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
channelProperties (common)	エンドポイントによって使用される JChannel の設定プロパティーを指定します。		文字列

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
enableViewMessages (consumer)	true に設定すると、コンシューマーエンドポイントは（org.jgroups.Message インスタンスだけでなく）org.jgroups.View メッセージも受信します。デフォルトでは、通常のメッセージのみがエンドポイントによって消費されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

165.3. HEADERS

ヘッダー	Constant	バージョン以降	説明

ヘッダー	Constant	バージョン以降	説明
JGROUPS_ORIGINAL_MESSAGE	**JGroupsEndpoint.HEADER_JGROUPS_ORIGINAL_MESSAGE**	2.13.0	消費されたメッセージのボディーが抽出された元の **org.jgroups.Message** インスタンスが抽出されます。
JGROUPS_SRC	`JGroupsEndpoint.`HEADER_JGROUPS_SRC	2.10.0	**Consumer** : 消費されるメッセージの **org.jgroups.Message**.getSrc（）メソッドによって展開した **org.jgroups.Address** インスタンス。**プロデューサー**: 送信されるメッセージのカスタムソースの **org.jgroups.Address**。
JGROUPS_DEST	`JGroupsEndpoint.`HEADER_JGROUPS_DEST	2.10.0	**Consumer**: 消費されるメッセージの **org.jgroups.Message**.getDest（）メソッドによって抽出される **org.jgroups.Address** インスタンス。**プロデューサー**: 送信されるメッセージのカスタム宛先 **org.jgroups.Address**。
JGROUPS_CHANNEL_ADDRESS	`JGroupsEndpoint.`HEADER_JGROUPS_CHANNEL_ADDRESS	2.13.0	エンドポイントに関連付けられたチャネルのアドレス（**org.jgroups.Address**）。

 # Usage

ルートのコンシューマー側で jgroups コンポーネントを使用すると、エンドポイントに関連付けられた JChannel によって受信されるメッセージを取得し、それらを Camel ルートに転送します。JGroups コンシューマーは受信メッセージを 非同期 で処理します。

```
// Capture messages from cluster named
// 'clusterName' and send them to Camel route.
from("jgroups:clusterName").to("seda:queue");
```

ルートのプロデューサー側で jgroups コンポーネントを使用すると、Camel エクスチェンジのボディーがエンドポイントによって管理される JChannel インスタンスに転送されます。

```
// Send message to the cluster named 'clusterName'
from("direct:start").to("jgroups:clusterName");
```

165.4. 事前定義されたフィルター

Camel バージョン 2 .13.0 以降、JGroups コンポーネントには JGroupsFilters という名前の事前定義されたフィルターファクトリークラスが同梱されています。

クラスターのコーディネーターに送信される変更通知のみを消費する（また「スレーブ」ノードに送信される）は、JGroupsFilters.dropNonCoordinatorViews（） フィルターを使用します。このフィルターは、特定のノードがクラスターのコーディネーターになると、単一の Camel ノードがクラスターでマスターになる必要がある場合に特に便利です。以下のスニペットは、マスターノードが受信したメッセージのみを収集する方法を示しています。

```
import static
org.apache.camel.component.jgroups.JGroupsFilters.dropNonCoordinatorViews;
...
from("jgroups:clusterName?enableViewMessages=true").
  filter(dropNonCoordinatorViews()).
  to("seda:masterNodeEventsQueue");
```

165.5. 事前定義された式

Camel バージョン 2 .13.0 以降、JGroups コンポーネントには JGroups Expressions という名前の事前定義された式ファクトリークラスが同梱されています。

Camel コンテキストがまだ開始されていない場合にのみルートに影響を与える遅延を作成する場合は、JGroupsExpressions.delayIfContextNotStarted(long delay) ファクトリーメソッドを使用します。このファクトリーメソッドによって作成された式は、Camel コンテキストが 開始 したものとは異なる場合にのみ指定の遅延値を返します。この式は、クラスター内でシングルトン（マスター）ルートを維持するために JGroups コンポーネントを使用する場合に便利です。Control Bus start コマンドは、Camel Context が起動していない場合はシングルトンルートを初期化しません。そのため、マスタールートの起動を遅延させ、Camel Context の起動後に初期化されていることを確認する必要があります。このようなシナリオはクラスターの初期化中にのみ発生する可能性があるので、スレーブノードの起動を新しいマスターに遅延させたくないので、条件的な遅延式が必要なからです。

以下のスニペットは、JGroups コンポーネントと条件遅延を使用して、クラスター内のマスターノードの初期起動を遅らせる方法を示しています。

```
import static java.util.concurrent.TimeUnit.SECONDS;
import static
org.apache.camel.component.jgroups.JGroupsExpressions.delayIfContextNotStarted;
import static
org.apache.camel.component.jgroups.JGroupsFilters.dropNonCoordinatorViews;
...
from("jgroups:clusterName?enableViewMessages=true").
  filter(dropNonCoordinatorViews()).
  threads().delay(delayIfContextNotStarted(SECONDS.toMillis(5))). // run in separated and
delayed thread. Delay only if the context hasn't been started already.
  to("controlbus:route?routeId=masterRoute&action=start&async=true");

from("timer://master?
repeatCount=1").routeId("masterRoute").autoStartup(false).to(masterMockUri);
```

165.6. 例

165.6.1. JGroups クラスターへ（受信）メッセージの送信（受信）

JGroups クラスターにメッセージを送信するには、以下のスニペットにあるようにプロデューサーエンドポイントを使用します。

```
from("direct:start").to("jgroups:myCluster");
...
producerTemplate.sendBody("direct:start", "msg")
```

（同じまたは他の物理マシン上）上のスニペットからメッセージを受信するには、以下のコードフラグメントで示すように、指定のクラスターから送信されるメッセージをリッスンします。

```
mockEndpoint.setExpectedMessageCount(1);
mockEndpoint.message(0).body().isEqualTo("msg");
...
from("jgroups:myCluster").to("mock:messagesFromTheCluster");
...
mockEndpoint.assertIsSatisfied();
```

165.6.2. クラスタービューの変更通知の受信

以下のスニペットは、クラスターメンバーシップの変更に関する通知をリッスンするコンシューマーエンドポイントを作成する方法を示しています。デフォルトでは、通常のメッセージのみがエンドポイントによって消費されます。

-

```
mockEndpoint.setExpectedMessageCount(1);
mockEndpoint.message(0).body().isInstanceOf(org.jgroups.View.class);
...
from("jgroups:clusterName?enableViewMessages=true").to(mockEndpoint);
...
mockEndpoint.assertIsSatisfied();
```

165.6.3. クラスター内でのシングルトンルートの維持

以下のスニペットは、Camel コンテキストのクラスターでシングルトンコンシューマールートを保持する方法を示しています。マスターノードが停止したら、スレーブの 1 つが新しいマスターとして選択され、起動します。この例では、アドレスの http://localhost:8080/orders` のリクエストをリッスンするシングルトン jetty インスタンスを維持します。

```
import static java.util.concurrent.TimeUnit.SECONDS;
import static
org.apache.camel.component.jgroups.JGroupsExpressions.delayIfContextNotStarted;
import static
org.apache.camel.component.jgroups.JGroupsFilters.dropNonCoordinatorViews;
...
from("jgroups:clusterName?enableViewMessages=true").
  filter(dropNonCoordinatorViews()).
  threads().delay(delayIfContextNotStarted(SECONDS.toMillis(5))). // run in separated and
delayed thread. Delay only if the context hasn't been started already.
  to("controlbus:route?routeId=masterRoute&action=start&async=true");

from("jetty:http://localhost:8080/orders").routeId("masterRoute").autoStartup(false).to("jms:or
ders");
```

第166章 JIBX DATAFORMAT

Camel バージョン 2.6 で利用可能

JiBX は、JiBX ライブラリー を使用して Java オブジェクトを XML にマーシャリングおよびアンマーシャリングするデータ形式です。

```
// lets turn Object messages into XML then send to MQSeries
from("activemq:My.Queue").
  marshal().jibx().
  to("mqseries:Another.Queue");
```

マーシャリングプロセスは、実行時にメッセージタイプを認識できることに注意してください。ただし、XML からメッセージをアンマーシャリングする場合は、ターゲットクラスを明示的に指定する必要があります。

```
// lets turn XML into PurchaseOrder message
from("mqseries:Another.Queue").
  unmarshal().jibx(PurchaseOrder.class).
  to("activemq:My.Queue");
```

166.1. オプション

JiBX データフォーマットは、以下に示す 3 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
unmarshallClass		文字列	XML から Java へのアンマーシャリング時に使用するクラス名。
bindingName		文字列	カスタムバインディングファクトリーの使用
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

166.2. JIBX SPRING DSL

JiBX データフォーマットは Camel Spring DSL でもサポートされます。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">

  <!-- Define data formats -->
  <dataFormats>
    <jibx id="jibx" unmarshallClass="org.apache.camel.dataformat.jibx.PurchaseOrder"/>
  </dataFormats>

  <!-- Marshal message to XML -->
  <route>
    <from uri="direct:marshal"/>
    <marshal ref="jibx"/>
    <to uri="mock:result"/>
  </route>

  <!-- Unmarshal message from XML -->
  <route>
    <from uri="direct:unmarshal"/>
    <unmarshal ref="jibx"/>
    <to uri="mock:result"/>
  </route>

</camelContext>
```

166.3. 依存関係

camel ルートで JiBX を使用するには、このデータ形式を実装する camel-jibx の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jibx</artifactId>
  <version>2.6.0</version>
</dependency>
```

第167章 JING コンポーネント

Camel バージョン 1.1 で利用可能

Jing コンポーネントは Jing Library を使用して、どちらかを使用してメッセージボディーの XML 検証を実行します。

- RelaxNG XML Syntax
- RelaxNG Compact Syntax

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jing</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

MSV コンポーネントは RelaxNG XML 構文をサポートすることもできます。

167.1. URI FORMAT CAMEL 2.16

```
jing:someLocalOrRemoteResource
```

Camel 2.16 以降、コンポーネントは jing を名前として使用し、オプション compactSyntax オプションを使用して RNG モードまたは RNC モードのいずれかを有効にすることができます。

167.2. オプション

Jing コンポーネントにはオプションがありません。

Jing エンドポイントは URI 構文を使用して設定します。

jing:resourceUri

以下の path パラメーターおよびクエリーパラメーターを使用します。

167.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceUri	クラスパスのローカルリソースまたは検証用のスキーマを含むファイルシステム上のリモートリソースまたはリソースへの完全な URL に **必要な** URL。		文字列

167.2.2. クエリーパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
compactSyntax (producer)	RelaxNG compact 構文を使用して検証するかどうか。デフォルトでは、これは RelaxNG XML Syntax(rng)使用では false です。true は RelaxNG Compact Syntax(rnc)を使用する場合に該当します。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

167.3. 例

以下の 例 は、エンドポイント direct:start からのルートを設定する方法を示しています。その後、XML が指定の RelaxNG Compact Syntax スキーマと一致するかどうかをもとに、mock:valid または mock:invalid のいずれかに移動します（クラスパスで提供されます）。

167.4. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*

- *エンドポイント*

- *はじめに*

第168章 JIRA コンポーネント

Camel バージョン 2.15 から利用可能

JIRA コンポーネントは、JIRA の Atlassian の REST Java クライアントをカプセル化して JIRA API と対話し ます。現在、新しい問題のポーリングと新しいコメントが提供されます。 新しい問題を作成することもできます。

このエンドポイントは Webhook ではなく、単純なポーリングに依存します。理由は次のとおりです。

- *信頼性/安定性の懸念*
- *ポーリングしているペイロードのタイプは通常大きくありません（さらに、ページングは API で利用可能です）*
- *Webhook が一般にアクセスできない場所で実行されているアプリケーションをサポートする必要がある*

JIRA API はかなり拡張されることに注意してください。 そのため、このコンポーネントは、追加の対話を提供するために簡単に拡張できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jira</artifactId>
  <version>${camel-version}</version>
</dependency>
```

168.1. URI 形式

```
jira://endpoint[?options]
```

168.2. JIRA オプション

JIRA コンポーネントにはオプションがありません。

JIRA エンドポイントは、URI 構文を使用して設定します。

```
jira:type
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

168.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
type	新規の問題や新規コメントの作成など、**必要な** 操作		JIRAType

168.2.2. クエリーパラメーター（9 パラメーター）：

Name	説明	デフォルト	Type
パスワード (common)	ログインのパスワード		文字列
serverUrl (common)	JIRA サーバーに **必要な** URL		文字列
username (common)	ログインのユーザー名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
遅延 （コンシューマー）	コンシューマーを使用して JIRA をクエリーするときの遅延（秒単位）。	6000	int

Name	説明	デフォルト	Type
jql（コンシューマー）	JQL は JIRA からのクエリー言語で、必要なデータを取得できます。たとえば、jql=project=MyProject Where MyProject は Jira の製品キーになります。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

168.3. JQL:

JQL URI オプションは、両方のコンシューマーエンドポイントによって使用されます。 理論的には、「プロジェクトキー」などの項目は URI オプション自体である可能性があります。 しかし、JQL を使うことで、コンシューマーはより柔軟で強力になります。

最低でも、コンシューマーには以下のものが必要です。

```
jira://[endpoint]?[required options]&jql=project=[project key]
```

重要なことは、newIssue コンシューマーが自動的に「ORDER BY key desc」を JQL に追加することです。 これは、プロジェクトのすべての問題についてインデックスを作成するのではなく、起動処理を最適化するために行われます。

同様に、newComment コンシューマーは、単一の問題 やプロジェクトのコメントをすべてインデックス化 する必要があります。 そのため、大規模なプロジェクトの場合、可能 な 限り JQL 式を最適化することが重要になります。 たとえば、JIRA Toolkit プラグインには、クエリーに「Number of comments」カスタムフィールド「Number of comments '"Number of comments」 > 0' が含まれています。また、状態(status=Open)に基づいて最小化を試みます。ポーリングの遅延などを増やします。 例:

```
jira://[endpoint]?[required options]&jql=RAW(project=[project key] AND status in (Open,
\"Coding In Progress\") AND \"Number of comments\">0)"
```

第169章 JMS コンポーネント

169.1. JMS コンポーネント

ヒント

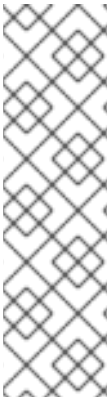
ActiveMQ の使用

Apache ActiveMQ を使用している場合は、ActiveMQ コンポーネントを最適化したため、ActiveMQ コンポーネントを使用することが推奨されます。このページのすべてのオプションとサンプルも ActiveMQ コンポーネントで有効です。

注記

トランザクションおよびキャッシュ

パフォーマンスに影響を与える可能性があるため、JMS でトランザクションを使用している場合は、以下の「トランザクション レベルおよびキャッシュ レベル」を参照してください。



注記

JMS での要求/応答

Camel はパフォーマンスやクラスター環境を設定するためのオプションを多数提供するため、このページの重要なヒントについては、このページの「Request-reply over JMS 」セクションを確認してください。

このコンポーネントを使用すると、JMS Queue または Topic にメッセージを送信（または消費）できます。送信に Spring の JmsTemplate や消費用に MessageListenerContainer を含む、宣言的トランザクションに Spring の JMS サポートを使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jms</artifactId>
```

```
    <version>x.x.x</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

169.2. URI 形式

```
jms:[queue:|topic:]destinationName[?options]
```

destinationName は JMS キューまたはトピック名です。デフォルトでは、destinationName はキュー名として解釈されます。たとえば、キューに接続するには、FOO.BAR は以下を使用します。

```
jms:FOO.BAR
```

必要に応じて、オプションの queue: プレフィックスを含めることができます。

```
jms:queue:FOO.BAR
```

トピックに接続するには、topic: プレフィックスを含める 必要があり ます。たとえば、でトピック Stocks.Prices に接続するには、以下を使用します。

```
jms:topic:Stocks.Prices
```

以下のフォーマットを使用して URI にクエリーオプションを追加します（？option=value&option=value&…

169.3. 備考

169.3.1. ActiveMQ の使用

JMS コンポーネントは、メッセージの送信に Spring 2 の JmsTemplate を再利用します。これは非 J2EE コンテナーでの使用には理想的ではなく、通常、パフォーマンス低下 を防ぐために JMS プロバイダーでのキャッシュの一部が必要になります。

Apache ActiveMQ を Message Broker として使用する場合（ActiveMQ ロックとして適切な選択肢）には、以下のいずれかを行うことが推奨されます。

- ActiveMQ を効率的に使用するようにすでに最適化された ActiveMQ コンポーネントを使用する

- *ActiveMQ で PoolingConnectionFactory を使用します。*

169.3.2. トランザクションおよびキャッシュレベル

メッセージを消費し、トランザクション(transacted=true)を使用する場合、キャッシュレベルのデフォルト設定はパフォーマンスに影響を及ぼす可能性があります。

XA トランザクションを使用している場合は、XA トランザクションが適切に機能しなくなる可能性があるため、キャッシュできません。

XA を使用して いない 場合は、cacheLevelName=CACHE_CONSUMER の設定など、パフォーマンスをスピードアップするため、キャッシュを検討する必要があります。

Camel 2.7.x では、cacheLevelName のデフォルト設定は CACHE_CONSUMER です。cacheLevelName=CACHE_NONE を明示的に設定する必要があります。

Camel 2.8 以降では、cacheLevelName のデフォルト設定は CACHE_AUTO です。このデフォルトの自動検出によりモードが検出され、それに応じてキャッシュレベルが設定されます。

- *CACHE_CONSUMER if transacted=false*
- *CACHE_NONE if transacted=true*

そのため、デフォルト設定が標準の状態であるとします。非 XA トランザクションを使用している場合は、cacheLevelName=CACHE_CONSUMER を使用することを検討してください。

169.3.3. 永続サブスクリプション

永続トピックサブスクリプションを使用する場合は、clientId と durableSubscriptionName の両方を指定する必要があります。clientId の値は一意で、ネットワーク全体で単一の JMS 接続インスタンスでのみ使用できます。この制限を回避するには 、仮想トピック の使用が推奨されます。ここでは、永続メッセージングの背景が上がり ます。

169.3.4. メッセージヘッダーマッピング

メッセージヘッダーを使用する場合、JMS 仕様ではヘッダー名が有効な Java 識別子である必要があることを示しています。そのため、ヘッダーの名前を有効な Java 識別子として命名します。これを実行する利点の 1 つは、JMS セレクター内でヘッダーを使用できることです（ヘッダーの Java 識別子構文が想定される SQL92 構文）。

ヘッダー名をマッピングする単純なストラテジーがデフォルトで使用されます。ストラテジーは、以下に示すようにヘッダー名のドットおよびハイフンを置き換え、ヘッダー名がネットワーク経由で送信される JMS メッセージから復元される際に置換を元に戻します。意味を確認するBean コンポーネントで呼び出すメソッド名の損失がなくなり、File Component などのファイル名のヘッダーが失われることはありません。

Camel でヘッダー名を受け入れるための現在のヘッダー名ストラテジーは次のとおりです。

- ドットは DOT に置き換えられ、Camel がメッセージを消費すると置換は逆になります。
- ハイフンは HYPHEN に置き換えられ、Camel がメッセージを消費すると置換が逆になります。

169.4. オプション

JMSConfiguration POJO のプロパティーにマップする JMS エンドポイントで多くの異なるプロパティーを設定できます。

警告

Spring JMS へのマッピング

これらのプロパティーの多くは、Camel がメッセージの送受信に使用する Spring JMS のプロパティーにマッピングされます。そのため、関連する Spring ドキュメントを参照して、これらのプロパティーの詳細情報を取得できます。

169.4.1. コンポーネントのオプション

JMS コンポーネントは、以下に示す 80 個のオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有 JMS 設定を使用するには、以下を実行します。		JmsConfiguration
acceptMessages while Stopping (consumer)	停止中にコンシューマーがメッセージを受け入れるかどうかを指定します。実行時に JMS ルートを開始および停止する場合に、キューでキューにまだキューに追加されたメッセージがある場合も、このオプションの有効化を検討してください。このオプションが false で、JMS ルートを停止すると、メッセージを拒否し、JMS ブローカーは再配信を試みる必要があります。これは再度拒否される可能性があり、最終的に JMS ブローカーのデッドレターキューに移動される可能性があります。これを回避するには、このオプションを有効にすることが推奨されます。	false	boolean
allowReplyManagerQuick Stop (consumer)	Request-reply メッセージングの応答マネージャーで使用される DefaultMessageListenerContainer により、JmsConfigurationisAcceptMessagesWhileStopping が有効になっている場合に DefaultMessageListenerContainer.runningAllowed フラグがすぐに停止でき、org.apache.camel.CamelContext が現在停止中です。この迅速な停止機能は、通常の JMS コンシューマーでデフォルトで有効になっていますが、応答マネージャーを有効にするには、このフラグを有効にする必要があります。	false	boolean
acknowledgementMode (consumer)	整数として定義される JMS 確認モード。ベンダー固有の拡張機能を確認モードに設定することができます。通常のモードでは、代わりに acknowledgementModeName を使用することが推奨されます。		int
EagerLoadingOf プロパティー （コンシューマー）	JMS プロパティーは必要でない可能性があるため、通常は非効率となるメッセージの読み込み直後に JMS プロパティーの Eager 読み込みを有効にしますが、場合によっては、基礎となる JMS プロバイダーの問題と JMS プロパティーの使用に早期にキャッチできます。	false	boolean
acknowledgementModeName (consumer)	JMS 確認名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	文字列

Name	説明	デフォルト	Type
autoStartup (consumer)	コンシューマーコンテナーが自動起動すべきかどうかを指定します。	true	boolean
cacheLevel (consumer)	ベースにある JMS リソースの ID でキャッシュレベルを設定します。詳細は、cacheLevelName オプションを参照してください。		int
cacheLevelName (consumer)	ベースにある JMS リソースの名前でキャッシュレベルを設定します。使用できる値は、CACHE_AUTO、CACHE_CONNECTION、CACHE_CONSUMER、CACHE_NONE、CACHE_SESSION です。デフォルト設定は CACHE_AUTO です。詳細は、Spring ドキュメント および Transactions Cache Levels を参照してください。	CACHE_AUTO	文字列
replyToCacheLevelName (producer)	JMS で要求/応答を実行するときに、リプライコンシューマーの名前でキャッシュレベルを設定します。このオプションは、（一時的ではなく）固定応答キューを使用している場合に限り該当します。Camel はデフォルトで CACHE_CONSUMER（排他的の場合）または shared w/ replyToSelectorName を使用します。replyToSelectorName なしで共有される CACHE_SESSION。IBM WebSphere などの JMS ブローカーによっては、reToCacheLevelName=CACHE_NONE が機能するように設定する必要がある場合があります。注記： 一時キューを使用する場合は CACHE_NONE は許可されず、CACHE_CONSUMER や CACHE_SESSION などの高い値を使用する必要があります。		文字列
clientId (common)	使用する JMS クライアント ID を設定します。この値は一意でなければならず、単一の JMS 接続インスタンスでのみ使用できることに注意してください。通常、永続トピックサブスクリプションにのみ必要です。Apache ActiveMQ を使用する場合は、代わりに Virtual Topics を使用することが推奨されます。		文字列
concurrentConsumers (consumer)	（JMS 上でリクエストや応答ではなく）JMS から消費する場合の同時コンシューマーのデフォルト数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。JMS で要求/応答を行う場合、オプション replyToConcurrentConsumers を使用して、リプライメッセージリスナーで同時コンシューマーの数を制御します。	1	int

Name	説明	デフォルト	Type
replyToConcurrent Consumers (producer)	JMS で要求/応答を実行する際の同時コンシューマーのデフォルト数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。	1	int
connectionFactory (common)	使用する接続ファクトリー。接続ファクトリーはコンポーネントまたはエンドポイントのいずれかで設定する必要があります。		ConnectionFactory
ユーザー名 （セキュリティー）	ConnectionFactory で使用するユーザー名。また、ユーザー名/パスワードを ConnectionFactory に直接設定することもできます。		文字列
パスワード （セキュリティー）	ConnectionFactory で使用するパスワード。また、ユーザー名/パスワードを ConnectionFactory に直接設定することもできます。		文字列
deliveryPersistent (producer)	永続配信がデフォルトで使用されるかどうかを指定します。	true	boolean
deliveryMode (producer)	使用する配信モードを指定します。使用できる値は、javax.jms.DeliveryMode で定義される値です。NON_PERSISTENT = 1 and PERSISTENT = 2.		整数
durableSubscriptionName (common)	永続トピックサブスクリプションを指定するための永続サブスクライバー名。clientId オプションも設定する必要があります。		文字列
exceptionListener (advanced)	基礎となる JMS 例外が通知される JMS 例外リスナーを指定します。		ExceptionListener
errorHandler (advanced)	メッセージの処理中にキャッチされない例外が発生した場合に呼び出される org.springframework.util.ErrorHandler を指定します。デフォルトでは、errorHandler が設定されていない場合、これらの例外は WARN レベルでログに記録されます。errorHandlerLoggingLevel および errorHandlerLogStackTrace オプションを使用して、ロギングレベルおよびスタックトレースをログに記録するかどうかを設定できます。これにより、カスタム errorHandler をコーディングする必要よりも、設定が非常に簡単になります。		ErrorHandler
errorHandlerLoggingLevel (logging)	キャッチされない例外についてのデフォルトの errorHandler ロギングレベルの設定を可能にします。	WARN	LoggingLevel

Name	説明	デフォルト	Type
errorHandlerLogStack Trace (logging)	デフォルトの errorHandler によってスタックトレースをログに記録するかどうかを制御できます。	true	boolean
explicitQosEnabled (producer)	メッセージの送信時に、サービスの deliveryMode、priority、または timeToLive qualities が使用される場合を設定します。このオプションは Spring の JmsTemplate に基づいています。deliveryMode、priority、および timeToLive オプションは、現在のエンドポイントに適用されます。これは、メッセージの粒度で操作する preserveMessageQos オプションとは対照的で、Camel In メッセージヘッダーからのみ QoS プロパティーを読み取ります。	false	boolean
exposeListenerSession (consumer)	メッセージの消費時にリスナーセッションを公開するかどうかを指定します。	false	boolean
idleTaskExecutionLimit (advanced)	実行内でメッセージを受信せずに、受信タスクのアイドル実行の制限を指定します。この制限に達すると、タスクはシャットダウンし、他の実行中のタスク（動的なスケジューリングの場合は maxConcurrentConsumers の設定）を受信し続けます。Spring には追加のドキュメントがあります。	1	int
idleConsumerLimit (advanced)	いつでもアイドル状態にできるコンシューマーの数の制限を指定します。	1	int
maxConcurrentConsumers (consumer)	（JMS 上でリクエストや応答ではなく）JMS から消費する場合の同時コンシューマーの最大数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。JMS で要求/応答を行う場合、オプション replyToMaxConcurrentConsumers を使用して、リプライメッセージリスナーで同時コンシューマーの数を制御します。		int
replyToMaxConcurrent Consumers (producer)	JMS 上でリクエスト/応答を使用する場合の同時コンシューマーの最大数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。		int

Name	説明	デフォルト	Type
replyOnTimeoutToMaxConcurrentConsumers (producer)	JMS でリクエスト/リプライタイムアウトを使用する場合に、継続中のルーティングの同時コンシューマーの最大数を指定します。	1	int
maxMessagesPerTask (advanced)	タスクごとのメッセージ数。-1 は無制限です。同時コンシューマー（min max など）に範囲を使用する場合、このオプションを使用して値を eg 100 に設定して、作業が少なくなる場合にコンシューマーのスピードを制御することができます。	-1	int
messageConverter (advanced)	javax.jms.Message への/からのマップ方法を制御するため、カスタムの Spring org.springframework.jms.support.converter.MessageConverter を使用します。		MessageConverter
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ（javax.jms.TextMessage など）に自動マッピングするかどうかを指定します。詳細は、以下のマッピングがどのように機能するかについてのセクションを参照してください。	true	boolean
messageIdEnabled (advanced)	送信時に、メッセージ ID を追加するかどうかを指定します。これは JMS Broker のヒントに過ぎません。JMS プロバイダーがこのヒントを受け入れる場合、これらのメッセージにはメッセージ ID が null に設定されている必要があります。プロバイダーがヒントを無視する場合は、メッセージ ID を通常の一意の値に設定する必要があります。	true	boolean
messageTimestampEnabled (advanced)	メッセージの送信時にタイムスタンプをデフォルトで有効にするかどうかを指定します。	true	boolean
alwaysCopyMessage (producer)	true の場合、送信のためにプロデューサーに渡されるときに Camel は常にメッセージの JMS メッセージコピーを作成します。replyToDestinationSelectorName が設定されている場合など、メッセージのコピーは一部の状況で必要になります（replyToDestinationSelectorName が設定されている場合、Camel は alwaysCopyMessage オプションを true に設定します）。	false	boolean
useMessageIDAsCorrelationID (advanced)	InOut メッセージに対して JMSCorrelationID として常に JMSMessageID を使用するかどうかを指定します。	false	boolean

Name	説明	デフォルト	Type
優先順位 （プロデューサー）	1より大きい値は送信時にメッセージの優先度を指定します（0 は優先度が最も低く、9 が最も高い値になります）。このオプションを有効にするには、explicitQosEnabled オプションも有効にする必要があります。	4	int
pubSubNoLocal (advanced)	独自の接続によって公開されるメッセージの配信を抑制するかどうかを指定します。	false	boolean
receiveTimeout (advanced)	メッセージの受信のタイムアウト（ミリ秒単位）。	1000	Long
recoveryInterval (advanced)	リカバリーを試行する間隔を指定します（例： 接続の更新時（ミリ秒単位）。デフォルトは 5000 ミリ秒で、5 秒です。	5000	Long
taskExecutor (consumer)	メッセージ消費にカスタムタスクエグゼキューターを指定できます。		TaskExecutor
timeToLive (producer)	メッセージを送信する場合、メッセージの存続時間（ミリ秒単位）を指定します。	-1	Long
トランザクション （トランザクション）	トランザクションモードを使用するかどうかを指定します。	false	boolean
lazyCreateTransaction Manager (transaction)	true の場合、transacted=true オプション時に transactionManager が挿入されていない場合、Camel は JmsTransactionManager を作成します。	true	boolean
transactionManager (transaction)	使用する Spring トランザクションマネージャー。		PlatformTransaction Manager
transactionName (transaction)	使用するトランザクションの名前。		文字列
transactionTimeout (transaction)	トランザクションモードを使用している場合は、トランザクションのタイムアウト値（秒単位）。	-1	int
testConnectionOn Startup (common)	起動時に接続をテストするかどうかを指定します。これにより、Camel が起動すると、すべての JMS コンシューマーが JMS ブローカーへの有効な接続になります。コネクションを許可できない場合、Camel は起動時に例外をスローします。これにより、Camel が接続の失敗で開始しないようになります。JMS プロデューサーもテストされています。	false	boolean

Name	説明	デフォルト	Type
asyncStartListener (advanced)	ルートの開始時に JmsConsumer メッセージリスナーを非同期に起動するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの起動中に Camel がブロックされます。このオプションを true に設定すると、ルートの起動が可能になりますが、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用すると、接続が確立されないと、WARN レベルで例外がログに記録され、コンシューマーはメッセージを受信できなくなります。次にルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止するときに JmsConsumer メッセージリスナーを非同期的に停止するかどうか。	false	boolean
forceSendOriginalMessage (producer)	mapJmsMessage=false を使用する場合、ルート中にヘッダー（get または set）を操作すると、Camel は新しい JMS 宛先に送信する新しい JMS メッセージを送信します。このオプションを true に設定して、Camel が受信した元の JMS メッセージを送信するように強制します。	false	boolean
requestTimeout (producer)	InOut エクスチェンジパターン（ミリ秒単位）を使用する場合の応答を待機するタイムアウト。デフォルトは 20 秒です。ヘッダー CamelJmsRequestTimeout を含めると、このエンドポイントに設定されたタイムアウト値を上書きすることができます。そのため、メッセージごとに個別のタイムアウト値を設定できます。requestTimeoutCheckerInterval オプションも参照してください。	20000	Long
requestTimeoutCheckerInterval (advanced)	JMS でリクエスト/リプライを行うときに Camel がタイムアウトしたエクスチェンジをチェックする頻度を設定します。デフォルトでは、Camel は 1 秒ごとに 1 回チェックします。ただし、タイムアウトの発生時により迅速に対応する必要がある場合は、この間隔を減らしてより頻繁に確認できます。タイムアウトは、requestTimeout オプションで決定されます。	1000	Long

Name	説明	デフォルト	Type
transferExchange (advanced)	ボディーとヘッダーだけでなく、ネットワーク上でエクスチェンジを転送できます。次のフィールドが転送されます。本文、Out ボディー、フォールト本文、In ヘッダー、Out ヘッダー、フォールトヘッダー、エクスチェンジプロパティー、エクスチェンジ例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。プロデューサーとコンシューマー側の両方でこのオプションを有効にする必要があります。そのため、Camel はペイロードが通常のペイロードではなく Exchange を認識します。	false	boolean
transferException (advanced)	有効にすると、リクエスト応答メッセージング(InOut)を使用し、エクスチェンジがコンシューマー側で失敗した場合、発生した例外は javax.jms.ObjectMessage として応答として返信されます。クライアントが Camel の場合、返された Exception が再スローされます。これにより、Camel JMS をルーティングのブリッジとして使用できます。たとえば、永続キューを使用して堅牢なルーティングを有効にすることができます。transferExchange も有効化されている場合には、このオプションが優先されます。例外がシリアライズ可能である必要があります。コンシューマー側の元の例外は、プロデューサーに戻されたときに org.apache.camel.RuntimeCamelException などの外部例外でラップできます。	false	boolean
transferFault (advanced)	有効にすると、Request Reply Messaging(InOut)を使用し、Exchange がコンシューマー側で SOAP 障害（例外ではない）で失敗した場合、リンク org.apache.camel.MessageisFault（）のフォールトフラグはキーリンク JmsConstantsJMS_TRANSFER_FAULT を持つ JMS ヘッダーとして応答として返されます。クライアントが Camel の場合、返されるフォールトフラグはリンク org.apache.camel.MessagesetFault(boolean)に設定されます。これは、cxf や spring-ws などの SOAP ベースの障害をサポートする Camel コンポーネントを使用する際に有効にすることができます。	false	boolean
jmsOperations (advanced)	org.springframework.jms.core.JmsOperations インターフェースの実装を使用できます。Camel はデフォルトで JmsTemplate を使用します。テストには使用できますが、Spring API ドキュメントに記載されているものだけを使用することはできません。		JmsOperations

Name	説明	デフォルト	Type
destinationResolver (advanced)	独自のリゾルバーを使用できるプラグ可能な org.springframework.jms.support.destination.DestinationResolver（たとえば、JNDI レジストリーから実際の宛先を検索するためなど）。		DestinationResolver
replyToType (producer)	JMS で要求/応答を実行するときに replyTo キューに使用するストラテジーを明示的に指定できます。使用できる値は、Temporary、Shared、または Exclusive です。デフォルトでは、Camel は一時キューを使用します。ただし、responseTo が設定されている場合は、デフォルトで Shared が使用されます。このオプションを使用すると、共有キューの代わりに排他キューを使用できます。詳細は、Camel JMS ドキュメント を参照してください。特に、クラスター環境で実行している場合の影響に関する注意事項と、Shared 応答キューのパフォーマンスは、その代替時間または排他的なものよりも低下します。		ReplyToType
preserveMessageQos (producer)	JMS エンドポイントの QoS 設定の代わりに、メッセージに指定された QoS 設定を使用してメッセージを送信する場合は、true に設定します。以下の 3 つのヘッダーは JMSPriority、JMSDeliveryMode、および JMSExpiration とみなされます。提供できるのは、すべてまたは一部のみです。指定しないと、Camel は代わりにエンドポイントから値を使用するようにフォールバックします。そのため、このオプションを使用すると、ヘッダーはエンドポイントの値を上書きします。一方、explicitQosEnabled オプションはエンドポイントに設定されたオプションのみを使用し、メッセージヘッダーの値を使用しません。	false	boolean
asyncConsumer (consumer)	JmsConsumer が Exchange を非同期的に処理するかどうか。有効にすると、JmsConsumer は JMS キューから次のメッセージを取得し、以前のメッセージは非同期で処理されます（非同期ルーティングエンジンにより）。つまり、メッセージは 100% を厳密に順番に処理できます。無効にすると（デフォルト）、JmsConsumer が JMS キューから次のメッセージを取得する前に Exchange が完全に処理されます。トランザクション処理が有効になっている場合、非同期Consumer=true は非同期的に実行されません。トランザクションは同期的に実行される必要があるため（Camel 3.0 は非同期トランザクションをサポートする可能性があります）。	false	boolean
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false で、メッセージボディーが null の場合、JMSException が発生します。	true	boolean

Name	説明	デフォルト	Type
includeSentJMSMessageID (producer)	InOnly を使用して JMS 宛先に送信する場合にのみ適用されます（例： fire and forget）。このオプションを有効にすると、Camel Exchange は、メッセージが JMS 宛先に送信されたときに JMS クライアントによって使用される実際の JMSMessageID でエンリッチされます。	false	boolean
includeAllJMSXプロパティー (advanced)	JMS から Camel メッセージへのマッピング時に、すべての JMSXxxx プロパティーを含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合は、このオプションは適用されません。	false	boolean
defaultTaskExecutor Type (consumer)	コンシューマーエンドポイントとプロデューサーエンドポイントの ReplyTo コンシューマーの両方に使用するデフォルトの TaskExecutor タイプを DefaultMessageListenerContainer に指定します。使用できる値 - SimpleAsync（Spring の SimpleAsyncTaskExecutor を使用）または ThreadPool（最適な値で Spring の ThreadPoolTaskExecutor を使用）、キャッシュされたスレッドプールのような値。設定しないと、コンシューマーエンドポイントにキャッシュされたスレッドプールを使用し、応答コンシューマーに SimpleAsync を使用する以前の動作がデフォルトになります。ThreadPool を使用すると、同時コンシューマーを動的に増減するエラスティック設定でスレッドごみ箱を減らすことが推奨されます。		DefaultTaskExecutor Type
jmsKeyFormatStrategy (advanced)	JMS 鍵をエンコードおよびデコードするためのプラグ可能なストラテジー。これにより、JMS 仕様に準拠します。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーは、ドットとハイフン（. および -）を安全にマーシャリングします。passthrough ストラテジーは、鍵をそのまま残します。JMS ヘッダーキーに不正な文字が含まれるかどうかは気にしない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy

Name	説明	デフォルト	Type
allowAdditionalHeaders (producer)	このオプションは、JMS 仕様に従って無効な値がある可能性のある追加のヘッダーを許可するために使用されます。たとえば、WMQ などの一部のメッセージシステムは、バイト配列またはその他の無効な型を持つ値が含まれる接頭辞 JMS_IBM_MQMD_ を使用するヘッダー名を持つものです。複数のヘッダー名をコンマで区切って指定し、ワイルドカードのマッチングに使用する接尾辞として使用できます。		文字列
queueBrowseStrategy (advanced)	キューの参照時にカスタム QueueBrowseStrategy を使用する		QueueBrowseStrategy
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信するときに Camel が javax.jms.Message オブジェクトの新規インスタンスを作成する際に呼び出される指定の MessageCreatedStrategy を使用します。		MessageCreatedStrategy
waitForProvisionCorrelationToBeUpdated Counter (advanced)	request/reply over JMS を実行し、オプション useMessageIDAsCorrelationID が有効になっている場合に、プロビジョニング ID が実際の相関 ID に更新されるまで待機する回数。	50	int
waitForProvisionCorrelationToBeUpdated ThreadSleepingTime (advanced)	プロビジョニングの相関 ID の更新を待機する間に毎回スリープ状態になる間隔（ミリ秒単位）。	100	Long
correlationProperty (producer)	この JMS プロパティーを使用して、JMSCorrelationID プロパティーの代わりに InOut 交換パターン（リクエスト-reply）でメッセージを関連付けます。これにより、JMSCorrelationID JMS プロパティーを使用して、メッセージを関連付けないシステムとメッセージを交換できます。JMSCorrelationID を使用した場合、Camel によって使用または設定されません。同じ名前のメッセージのヘッダーに指定されていない場合、ここで名前付きプロパティーの値が生成されます。		文字列
subscriptionDurable (consumer)	サブスクリプションを永続化するかどうかを設定します。使用する永続サブスクリプション名は、subscriptionName プロパティーから指定できます。デフォルトは false です。通常、永続サブスクリプションを登録するには、これを true に設定します（メッセージリスナークラス名がサブスクリプション名として適切である場合）。トピックをリッスンする場合にのみ（pub-sub ドメイン）、このメソッドは pubSubDomain フラグも切り替えます。	false	boolean

Name	説明	デフォルト	Type
subscriptionShared (consumer)	サブスクリプションを共有するかどうかを設定します。使用する共有サブスクリプション名は、subscriptionName プロパティーから指定できます。デフォルトは false です。共有サブスクリプションを登録するには、true に設定します。通常、subscriptionName の値と併用します（メッセージリスナークラス名がサブスクリプション名として適切である場合）。共有サブスクリプションも永続的である可能性があるため、このフラグは subscriptionDurable と組み合わせることもできます。トピックをリッスンする場合にのみ（pub-sub ドメイン）、このメソッドは pubSubDomain フラグも切り替えます。JMS 2.0 と互換性のあるメッセージブローカーが必要です。	false	boolean
subscriptionName (consumer)	作成するサブスクリプションの名前を設定します。共有サブスクリプションまたは永続サブスクリプションがあるトピック（pub-sub ドメイン）の場合に適用できます。サブスクリプション名は、クライアントの JMS クライアント ID 内で一意である必要があります。default は、指定されたメッセージリスナーのクラス名です。注記： 共有サブスクリプション（JMS 2.0 が必要）を除き、サブスクリプションごとに 1 つの同時コンシューマー（このメッセージリスナーコンテナーのデフォルト）のみが許可されます。		文字列
streamMessageType Enabled (producer)	StreamMessage タイプが有効かどうかを指定します。ファイル、InputStream などのストリーミングタイプのメッセージペイロードは、BytesMessage または StreamMessage として送信されます。このオプションは、使用される種類を制御します。デフォルトでは、BytesMessage を使用して、メッセージペイロード全体をメモリーに読み込みます。このオプションを有効にすると、メッセージペイロードはチャンクのメモリーに読み取られ、各チャンクはデータがなくなるまで StreamMessage に書き込まれます。	false	boolean
formatDateHeadersTo Iso8601 (producer)	ISO 8601 標準に従って日付ヘッダーをフォーマットするかどうかを設定します。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

169.4.2. エンドポイントオプション

JMS エンドポイントは、URI 構文を使用して設定します。

```
jms:destinationType:destinationName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

169.4.3. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
destinationType	使用する宛先の種類	queue	文字列
destinationName	宛先として使用するキューまたはトピックの名前。		文字列

169.4.4. クエリーパラメーター（91 パラメーター）：

Name	説明	デフォルト	Type
clientId (common)	使用する JMS クライアント ID を設定します。この値は一意でなければならず、単一の JMS 接続インスタンスでのみ使用できることに注意してください。通常、永続トピックサブスクリプションにのみ必要です。Apache ActiveMQ を使用する場合は、代わりに Virtual Topics を使用することが推奨されます。		文字列

Name	説明	デフォルト	Type
connectionFactory (common)	リンク setTemplateConnectionFactory(ConnectionFactory) またはリンク setListenerConnectionFactory(ConnectionFactory) のいずれかに接続ファクトリーが指定されていない場合に使用するデフォルトの接続ファクトリーを設定します。		ConnectionFactory
disableReplyTo (common)	Camel がメッセージの JMSReplyTo ヘッダーを無視するかどうかを指定します。true の場合、Camel は JMSReplyTo ヘッダーで指定された宛先に返信を返しません。Camel がルートから消費し、コードの別のコンポーネントがリプライメッセージを処理するため、Camel が自動的に応答メッセージを送りないようにするには、このオプションを使用できます。Camel を異なるメッセージブローカー間でプロキシーとして使用し、あるシステムから別のシステムへメッセージをルーティングする場合は、このオプションを使用することもできます。	false	boolean
durableSubscriptionName (common)	永続トピックサブスクリプションを指定するための永続サブスクライバー名。clientId オプションも設定する必要があります。		文字列
jmsMessageType (common)	JMS メッセージの送信に、特定の javax.jms.Message 実装を強制的に使用できるようにします。使用できる値は、Bytes、Map、Object、Stream、Text です。デフォルトでは、Camel は In body タイプから使用する JMS メッセージタイプを決定します。このオプションを指定すると、指定できます。		JmsMessageType
testConnectionOnStartup (common)	起動時に接続をテストするかどうかを指定します。これにより、Camel が起動すると、すべての JMS コンシューマーが JMS ブローカーへの有効な接続になります。コネクションを許可できない場合、Camel は起動時に例外をスローします。これにより、Camel が接続の失敗で開始しないようになります。JMS プロデューサーもテストされています。	false	boolean
acknowledgementModeName (consumer)	JMS 確認名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	文字列

Name	説明	デフォルト	Type
asyncConsumer (consumer)	JmsConsumer が Exchange を非同期的に処理するかどうか。有効にすると、JmsConsumer は JMS キューから次のメッセージを取得し、以前のメッセージは非同期で処理されます（非同期ルーティングエンジンにより）。つまり、メッセージは 100% を厳密に順番に処理できます。無効にすると（デフォルト）、JmsConsumer が JMS キューから次のメッセージを取得する前に Exchange が完全に処理されます。トランザクション処理が有効になっている場合、非同期Consumer=true は非同期的に実行されません。トランザクションは同期的に実行される必要があるため（Camel 3.0 は非同期トランザクションをサポートする可能性があります）。	false	boolean
autoStartup (consumer)	コンシューマーコンテナーが自動起動すべきかどうかを指定します。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
cacheLevel (consumer)	ベースにある JMS リソースの ID でキャッシュレベルを設定します。詳細は、cacheLevelName オプションを参照してください。		int
cacheLevelName (consumer)	ベースにある JMS リソースの名前でキャッシュレベルを設定します。使用できる値は、CACHE_AUTO、CACHE_CONNECTION、CACHE_CONSUMER、CACHE_NONE、CACHE_SESSION です。デフォルト設定は CACHE_AUTO です。詳細は、Spring ドキュメント および Transactions Cache Levels を参照してください。	CACHE_AUTO	文字列
concurrentConsumers (consumer)	（JMS 上でリクエストや応答ではなく）JMS から消費する場合の同時コンシューマーのデフォルト数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。JMS で要求/応答を行う場合、オプション replyToConcurrentConsumers を使用して、リプライメッセージリスナーで同時コンシューマーの数を制御します。	1	int

Name	説明	デフォルト	Type
maxConcurrentConsumers (consumer)	（JMS 上でリクエストや応答ではなく）JMS から消費する場合の同時コンシューマーの最大数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。JMS で要求/応答を行う場合、オプション replyToMaxConcurrentConsumers を使用して、リプライメッセージリスナーで同時コンシューマーの数を制御します。		int
replyTo (consumer)	Message.getJMSReplyTo（）の着信値を上書きする明示的な ReplyTo 宛先を提供します。		文字列
replyToDeliveryPersistent (consumer)	返信に永続配信を使用するかどうかを指定します。	true	boolean
セレクター（コンシューマー）	使用する JMS セレクターを設定します。		文字列
subscriptionDurable (consumer)	サブスクリプションを永続化するかどうかを設定します。使用する永続サブスクリプション名は、subscriptionName プロパティーから指定できます。デフォルトは false です。通常、永続サブスクリプションを登録するには、これを true に設定します（メッセージリスナークラス名がサブスクリプション名として適切である場合）。トピックをリッスンする場合にのみ（pub-sub ドメイン）、このメソッドは pubSubDomain フラグも切り替えます。	false	boolean
subscriptionName (consumer)	作成するサブスクリプションの名前を設定します。共有サブスクリプションまたは永続サブスクリプションがあるトピック（pub-sub ドメイン）の場合に適用できます。サブスクリプション名は、クライアントの JMS クライアント ID 内で一意である必要があります。default は、指定されたメッセージリスナーのクラス名です。注記： 共有サブスクリプション（JMS 2.0 が必要）を除き、サブスクリプションごとに 1 つの同時コンシューマー（このメッセージリスナーコンテナーのデフォルト）のみが許可されます。		文字列

Name	説明	デフォルト	Type
subscriptionShared (consumer)	サブスクリプションを共有するかどうかを設定します。使用する共有サブスクリプション名は、subscriptionName プロパティーから指定できます。デフォルトは false です。共有サブスクリプションを登録するには、true に設定します。通常、subscriptionName の値と併用します（メッセージリスナークラス名がサブスクリプション名として適切である場合）。共有サブスクリプションも永続的である可能性があるため、このフラグは subscriptionDurable と組み合わせることもできます。トピックをリッスンする場合にのみ（pub-sub ドメイン）、このメソッドは pubSubDomain フラグも切り替えます。JMS 2.0 と互換性のあるメッセージブローカーが必要です。	false	boolean
acceptMessagesWhileStopping (consumer)	停止中にコンシューマーがメッセージを受け入れるかどうかを指定します。実行時に JMS ルートを開始および停止する場合に、キューでキューにまだキューに追加されたメッセージがある場合も、このオプションの有効化を検討してください。このオプションが false で、JMS ルートを停止すると、メッセージを拒否し、JMS ブローカーは再配信を試みる必要があります。これは再度拒否される可能性があり、最終的に JMS ブローカーのデッドレターキューに移動される可能性があります。これを回避するには、このオプションを有効にすることが推奨されます。	false	boolean
allowReplyManagerQuickStop (consumer)	request-reply メッセージングの応答マネージャーで使用される DefaultMessageListenerContainer が、リンク JmsConfigurationisAcceptMessagesWhileStopping（）が有効になっている場合に、リンク DefaultMessageListenerContainerrunningAllowed（）フラグがすぐに停止できるようにするかどうか。また、org.apache.camel.CamelContext が現在停止中です。この迅速な停止機能は、通常の JMS コンシューマーでデフォルトで有効になっていますが、応答マネージャーを有効にするには、このフラグを有効にする必要があります。	false	boolean

Name	説明	デフォルト	Type
consumerType (consumer)	使用するコンシューマータイプ。Simple、Default、または Custom のいずれかです。コンシューマータイプは、使用する Spring JMS リスナーを決定します。デフォルトは org.springframework.jms.listener.DefaultMessageListenerContainer を使用します。Simple は org.springframework.jms.listener.SimpleMessageListenerContainer を使用します。Custom を指定すると、messageListenerContainerFactory オプションによって定義された MessageListenerContainerFactory は、使用する org.springframework.jms.listener.AbstractMessageListenerContainer を決定します。	デフォルト	ConsumerType
defaultTaskExecutorType (consumer)	コンシューマーエンドポイントとプロデューサーエンドポイントの ReplyTo コンシューマーの両方に使用するデフォルトの TaskExecutor タイプを DefaultMessageListenerContainer に指定します。使用できる値 - SimpleAsync（Spring の SimpleAsyncTaskExecutor を使用）または ThreadPool（最適な値で Spring の ThreadPoolTaskExecutor を使用）、キャッシュされたスレッドプールのような値。設定しないと、コンシューマーエンドポイントにキャッシュされたスレッドプールを使用し、応答コンシューマーに SimpleAsync を使用する以前の動作がデフォルトになります。ThreadPool を使用すると、同時コンシューマーを動的に増減するエラスティック設定でスレッドごみ箱を減らすことが推奨されます。		DefaultTaskExecutor Type
eagerLoadingOfProperties (consumer)	JMS プロパティーは必須ではありませんが、基礎となる JMS プロバイダーと JMS プロパティーの使用の初期段階で、メッセージが読み込まれるとすぐに JMS プロパティーとペイロードの Eager 読み込みを有効にします。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
exposeListenerSession (consumer)	メッセージの消費時にリスナーセッションを公開するかどうかを指定します。	false	boolean

Name	説明	デフォルト	Type
replyToSameDestination Allowed（コンシューマー）	JMS コンシューマーがコンシューマーが消費に使用する同じ宛先にリプライメッセージを送信することができるかどうか。これにより、同じメッセージを独自に消費してバックエンドレスループを防ぐことができます。	false	boolean
taskExecutor (consumer)	メッセージ消費にカスタムタスクエグゼキューターを指定できます。		TaskExecutor
deliveryMode (producer)	使用する配信モードを指定します。使用できる値は、javax.jms.DeliveryMode で定義される値です。NON_PERSISTENT = 1 and PERSISTENT = 2.		整数
deliveryPersistent (producer)	永続配信がデフォルトで使用されるかどうかを指定します。	true	boolean
explicitQosEnabled (producer)	メッセージの送信時に、サービスの deliveryMode、priority、または timeToLive qualities が使用される場合を設定します。このオプションは Spring の JmsTemplate に基づいています。deliveryMode、priority、および timeToLive オプションは、現在のエンドポイントに適用されます。これは、メッセージの粒度で操作する preserveMessageQos オプションとは対照的で、Camel In メッセージヘッダーからのみ QoS プロパティーを読み取ります。	false	ブール値
formatDateHeadersToIso8601 (producer)	ISO 8601 標準に従って日付ヘッダーをフォーマットするかどうかを設定します。	false	boolean
preserveMessageQos (producer)	JMS エンドポイントの QoS 設定の代わりに、メッセージに指定された QoS 設定を使用してメッセージを送信する場合は、true に設定します。以下の 3 つのヘッダーは JMSPriority、JMSDeliveryMode、および JMSExpiration とみなされます。提供できるのは、すべてまたは一部のみです。指定しないと、Camel は代わりにエンドポイントから値を使用するようにフォールバックします。そのため、このオプションを使用すると、ヘッダーはエンドポイントの値を上書きします。一方、explicitQosEnabled オプションはエンドポイントに設定されたオプションのみを使用し、メッセージヘッダーの値を使用しません。	false	boolean
優先順位（プロデューサー）	1 より大きい値は送信時にメッセージの優先度を指定します（0 は優先度が最も低く、9 が最も高い値になります）。このオプションを有効にするには、explicitQosEnabled オプションも有効にする必要があります。	4	int

Name	説明	デフォルト	Type
replyToConcurrentConsumers (producer)	JMS で要求/応答を実行する際の同時コンシューマーのデフォルト数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。	1	int
replyToMaxConcurrent Consumers (producer)	JMS 上でリクエスト/応答を使用する場合の同時コンシューマーの最大数を指定します。スレッドの動的スケールアップ/ダウンを制御する maxMessagesPerTask オプションも参照してください。		int
replyToOnTimeoutMax ConcurrentConsumers (producer)	JMS でリクエスト/リプライタイムアウトを使用する場合に、継続中のルーティングの同時コンシューマーの最大数を指定します。	1	int
replyToOverride (producer)	JMS メッセージで明示的な ReplyTo 宛先を提供します。これにより、ReplyTo の設定が上書きされます。メッセージをリモートキューへ転送し、ReplyTo 宛先からリプライメッセージを受信する場合に便利です。		文字列
replyToType (producer)	JMS で要求/応答を実行するときに replyTo キューに使用するストラテジーを明示的に指定できます。使用できる値は、Temporary、Shared、または Exclusive です。デフォルトでは、Camel は一時キューを使用します。ただし、responseTo が設定されている場合は、デフォルトで Shared が使用されます。このオプションを使用すると、共有キューの代わりに排他キューを使用できます。詳細は、Camel JMS ドキュメント を参照してください。特に、クラスター環境で実行している場合の影響に関する注意事項と、Shared 応答キューのパフォーマンスは、その代替時間または排他的なものよりも低下します。		ReplyToType
requestTimeout (producer)	InOut エクスチェンジパターン（ミリ秒単位）を使用する場合の応答を待機するタイムアウト。デフォルトは 20 秒です。ヘッダー CamelJmsRequestTimeout を含めると、このエンドポイントに設定されたタイムアウト値を上書きすることができます。そのため、メッセージごとに個別のタイムアウト値を設定できます。requestTimeoutCheckerInterval オプションも参照してください。	20000	Long
timeToLive (producer)	メッセージを送信する場合、メッセージの存続時間（ミリ秒単位）を指定します。	-1	Long

Name	説明	デフォルト	Type
allowAdditionalHeaders (producer)	このオプションは、JMS 仕様に従って無効な値がある可能性のある追加のヘッダーを許可するために使用されます。たとえば、WMQ などの一部のメッセージシステムは、バイト配列またはその他の無効な型を持つ値が含まれる接頭辞 JMS_IBM_MQMD_ を使用するヘッダー名を持つものです。複数のヘッダー名をコンマで区切って指定し、ワイルドカードのマッチングに使用する接尾辞として使用できます。		文字列
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false で、メッセージボディーが null の場合、JMSException が発生します。	true	boolean
alwaysCopyMessage (producer)	true の場合、送信のためにプロデューサーに渡されるときに Camel は常にメッセージの JMS メッセージコピーを作成します。replyToDestinationSelectorName が設定されている場合など、メッセージのコピーは一部の状況で必要になります（replyToDestinationSelectorName が設定されている場合、Camel は alwaysCopyMessage オプションを true に設定します）。	false	boolean
correlationProperty (producer)	この JMS プロパティーを使用して、JMSCorrelationID プロパティーの代わりに InOut 交換パターン（リクエスト-reply）でメッセージを関連付けます。これにより、JMSCorrelationID JMS プロパティーを使用して、メッセージを関連付けないシステムとメッセージを交換できます。JMSCorrelationID を使用した場合、Camel によって使用または設定されません。同じ名前のメッセージのヘッダーに指定されていない場合、ここで名前付きプロパティーの値が生成されます。		文字列
disableTimeToLive (producer)	このオプションを使用して、ライブ時間を強制的に無効にします。たとえば、JMS でリクエスト/応答を行う場合、Camel はデフォルトで送信されるメッセージの有効期間として requestTimeout の値を使用します。この問題は、送信側および受信側システムにクロックを同期して同期する必要があることです。これは常に簡単にアーカイブできる訳ではありません。したがって、disableTimeToLive=true を使用して、送信されたメッセージで時間がライブ値を設定しないようにすることができます。その後、メッセージは受信側システムで期限切れになりません。詳細は、以下のセクションの「ライブ時間について」を参照してください。	false	boolean

Name	説明	デフォルト	Type
forceSendOriginalMessage (producer)	mapJmsMessage=false を使用する場合、ルート中にヘッダー（get または set）を操作すると、Camel は新しい JMS 宛先に送信する新しい JMS メッセージを送信します。このオプションを true に設定して、Camel が受信した元の JMS メッセージを送信するように強制します。	false	boolean
includeSentJMSMessageID (producer)	InOnly を使用して JMS 宛先に送信する場合にのみ適用されます（例： fire and forget）。このオプションを有効にすると、Camel Exchange は、メッセージが JMS 宛先に送信されたときに JMS クライアントによって使用される実際の JMSMessageID でエンリッチされます。	false	boolean
replyToCacheLevelName (producer)	JMS で要求/応答を実行するときに、リプライコンシューマーの名前でキャッシュレベルを設定します。このオプションは、（一時的ではなく）固定応答キューを使用している場合に限り該当します。Camel はデフォルトで CACHE_CONSUMER（排他的の場合）または shared w/ replyToSelectorName を使用します。replyToSelectorName なしで共有される CACHE_SESSION。IBM WebSphere などの JMS ブローカーによっては、reToCacheLevelName=CACHE_NONE が機能するように設定する必要がある場合があります。注記： 一時キューを使用する場合は CACHE_NONE は許可されず、CACHE_CONSUMER や CACHE_SESSION などの高い値を使用する必要があります。		文字列
replyToDestinationSelector Name (producer)	共有キューの使用時に（一時的な応答キューを使用していない場合は）他のユーザーからの独自のリプライをフィルタリングできるように、固定名を使用して JMS セレクターを設定します。		文字列
streamMessageTypeEnabled (producer)	StreamMessage タイプが有効かどうかを指定します。ファイル、InputStream などのストリーミングタイプのメッセージペイロードは、BytesMessage または StreamMessage として送信されます。このオプションは、使用される種類を制御します。デフォルトでは、BytesMessage を使用して、メッセージペイロード全体をメモリーに読み込みます。このオプションを有効にすると、メッセージペイロードはチャンクのメモリーに読み取られ、各チャンクはデータがなくなるまで StreamMessage に書き込まれます。	false	boolean

Name	説明	デフォルト	Type
allowSerializedHeaders (advanced)	シリアライズされたヘッダーを含めるかどうかを制御します。Applies only when link isTransferExchange() is true.これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。	false	boolean
asyncStartListener (advanced)	ルートの開始時に JmsConsumer メッセージリスナーを非同期に起動するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの起動中に Camel がブロックされます。このオプションを true に設定すると、ルートの起動が可能になりますが、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用すると、接続が確立されないと、WARN レベルで例外がログに記録され、コンシューマーはメッセージを受信できなくなります。次にルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止するときに JmsConsumer メッセージリスナーを非同期的に停止するかどうか。	false	boolean
destinationResolver (advanced)	独自のリゾルバーを使用できるプラグ可能な org.springframework.jms.support.destination.DestinationResolver（たとえば、JNDI レジストリーから実際の宛先を検索するためなど）。		DestinationResolver
errorHandler (advanced)	メッセージの処理中にキャッチされない例外が発生した場合に呼び出される org.springframework.util.ErrorHandler を指定します。デフォルトでは、errorHandler が設定されていない場合、これらの例外は WARN レベルでログに記録されます。errorHandlerLoggingLevel および errorHandlerLogStackTrace オプションを使用して、ロギングレベルおよびスタックトレースをログに記録するかどうかを設定できます。これにより、カスタム errorHandler をコーディングする必要よりも、設定が非常に簡単になります。		ErrorHandler
exceptionListener (advanced)	基礎となる JMS 例外が通知される JMS 例外リスナーを指定します。		ExceptionListener
headerFilterStrategy (advanced)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy

Name	説明	デフォルト	Type
idleConsumerLimit (advanced)	いつでもアイドル状態にできるコンシューマーの数の制限を指定します。	1	int
idleTaskExecutionLimit (advanced)	実行内でメッセージを受信せずに、受信タスクのアイドル実行の制限を指定します。この制限に達すると、タスクはシャットダウンし、他の実行中のタスク（動的なスケジューリングの場合は maxConcurrentConsumers の設定）を受信し続けます。Spring には追加のドキュメントがあります。	1	int
includeAllJMSXProperties (advanced)	JMS から Camel メッセージへのマッピング時に、すべての JMSXxxx プロパティーを含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合は、このオプションは適用されません。	false	boolean
jmsKeyFormatStrategy (advanced)	JMS 鍵をエンコードおよびデコードするためのプラグ可能なストラテジー。これにより、JMS 仕様に準拠します。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーは、ドットとハイフン（. および -）を安全にマーシャリングします。passthrough ストラテジーは、鍵をそのまま残します。JMS ヘッダーキーに不正な文字が含まれるかどうかは気にしない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		文字列
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ（javax.jms.TextMessage など）に自動マッピングするかどうかを指定します。	true	boolean
maxMessagesPerTask (advanced)	タスクごとのメッセージ数。-1 は無制限です。同時コンシューマー（min max など）に範囲を使用する場合、このオプションを使用して値を eg 100 に設定して、作業が少なくなる場合にコンシューマーのスピードを制御することができます。	-1	int
messageConverter (advanced)	javax.jms.Message への/からのマップ方法を制御するため、カスタムの Spring org.springframework.jms.support.converter.MessageConverter を使用します。		MessageConverter

Name	説明	デフォルト	Type
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信するときに Camel が javax.jms.Message オブジェクトの新規インスタンスを作成する際に呼び出される指定の MessageCreatedStrategy を使用します。		MessageCreatedStrategy
messageIdEnabled (advanced)	送信時に、メッセージ ID を追加するかどうかを指定します。これは JMS Broker のヒントに過ぎません。JMS プロバイダーがこのヒントを受け入れる場合、これらのメッセージにはメッセージ ID が null に設定されている必要があります。プロバイダーがヒントを無視する場合は、メッセージ ID を通常の一意の値に設定する必要があります。	true	boolean
messageListenerContainerFactory (advanced)	メッセージを消費するために使用する org.springframework.jms.listener.AbstractMessageListenerContainer を決定するために使用される MessageListenerContainerFactory のレジストリー ID。これを設定すると、consumerType が自動的に Custom に設定されます。		MessageListenerContainerFactory
messageTimestampEnabled (advanced)	メッセージの送信時にタイムスタンプをデフォルトで有効にするかどうかを指定します。これは JMS Broker のヒントに過ぎません。JMS プロバイダーがこのヒントを受け入れる場合、これらのメッセージのタイムスタンプをゼロに設定する必要があります。プロバイダーがヒントを無視する場合は、タイムスタンプを通常の値に設定する必要があります。	true	boolean
pubSubNoLocal (advanced)	独自の接続によって公開されるメッセージの配信を抑制するかどうかを指定します。	false	boolean
receiveTimeout (advanced)	メッセージの受信のタイムアウト（ミリ秒単位）。	1000	Long
recoveryInterval (advanced)	リカバリーを試行する間隔を指定します（例： 接続の更新時（ミリ秒単位）。デフォルトは 5000 ミリ秒で、5 秒です。	5000	Long
requestTimeoutCheckerInterval (advanced)	JMS でリクエスト/リプライを行うときに Camel がタイムアウトしたエクスチェンジをチェックする頻度を設定します。デフォルトでは、Camel は 1 秒ごとに 1 回チェックします。ただし、タイムアウトの発生時により迅速に対応する必要がある場合は、この間隔を減らしてより頻繁に確認できます。タイムアウトは、requestTimeout オプションで決定されます。	1000	Long

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
transferException (advanced)	有効にすると、リクエスト応答メッセージング (InOut)を使用し、エクスチェンジがコンシューマー側で失敗した場合、発生した例外は javax.jms.ObjectMessage として応答として返信されます。クライアントが Camel の場合、返された Exception が再スローされます。これにより、Camel JMS をルーティングのブリッジとして使用できます。たとえば、永続キューを使用して堅牢なルーティングを有効にすることができます。transferExchange も有効化されている場合には、このオプションが優先されます。例外がシリアライズ可能である必要があります。コンシューマー側の元の例外は、プロデューサーに戻されたときに org.apache.camel.RuntimeCamelException などの外部例外でラップできます。	false	boolean
transferExchange (advanced)	ボディーとヘッダーだけでなく、ネットワーク上でエクスチェンジを転送できます。次のフィールドが転送されます。本文、Out ボディー、フォールト本文、In ヘッダー、Out ヘッダー、フォールトヘッダー、エクスチェンジプロパティー、エクスチェンジ例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。プロデューサーとコンシューマー側の両方でこのオプションを有効にする必要があります。そのため、Camel はペイロードが通常のペイロードではなく Exchange を認識します。	false	boolean
transferFault (advanced)	有効にすると、Request Reply Messaging(InOut)を使用し、Exchange がコンシューマー側で SOAP 障害（例外ではない）で失敗した場合、リンク org.apache.camel.MessageisFault（）のフォールトフラグはキーリンク JmsConstantsJMS_TRANSFER_FAULT を持つ JMS ヘッダーとして応答として返されます。クライアントが Camel の場合、返されるフォールトフラグはリンク org.apache.camel.MessagesetFault(boolean)に設定されます。これは、cxf や spring-ws などの SOAP ベースの障害をサポートする Camel コンポーネントを使用する際に有効にすることができます。	false	boolean
useMessageIDAs CorrelationID (advanced)	InOut メッセージに対して JMSCorrelationID として常に JMSMessageID を使用するかどうかを指定します。	false	boolean

Name	説明	デフォルト	Type
waitForProvisionCorrelationToBeUpdatedCounter (advanced)	request/reply over JMS を実行し、オプション useMessageIDAsCorrelationID が有効になっている場合に、プロビジョニング ID が実際の相関 ID に更新されるまで待機する回数。	50	int
waitForProvisionCorrelationToBeUpdatedThreadSleeping Time (advanced)	プロビジョニングの相関 ID の更新を待機する間に毎回スリープ状態になる間隔（ミリ秒単位）。	100	Long
errorHandlerLoggingLevel (logging)	キャッチされない例外についてのデフォルトの errorHandler ロギングレベルの設定を可能にします。	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	デフォルトの errorHandler によってスタックトレースをログに記録するかどうかを制御できます。	true	boolean
パスワード （セキュリティー）	ConnectionFactory で使用するパスワード。また、ユーザー名/パスワードを ConnectionFactory に直接設定することもできます。		文字列
ユーザー名 （セキュリティー）	ConnectionFactory で使用するユーザー名。また、ユーザー名/パスワードを ConnectionFactory に直接設定することもできます。		文字列
トランザクション （トランザクション）	トランザクションモードを使用するかどうかを指定します。	false	boolean
lazyCreateTransaction Manager (transaction)	true の場合、transacted=true オプション時に transactionManager が挿入されていない場合、Camel は JmsTransactionManager を作成します。	true	boolean
transactionManager (transaction)	使用する Spring トランザクションマネージャー。		PlatformTransaction Manager
transactionName (transaction)	使用するトランザクションの名前。		文字列
transactionTimeout (transaction)	トランザクションモードを使用している場合は、トランザクションのタイムアウト値（秒単位）。	-1	int

169.5. JMS と CAMEL 間のメッセージマッピング

Camel は javax.jms.Message と org.apache.camel.Message の間でメッセージを自動的にマッピングします。

JMS メッセージを送信する場合、Camel はメッセージボディーを以下の JMS メッセージタイプに変換します。

本文タイプ	JMS メッセージ	Comment
文字列	**javax.jms.TextMessage**	
org.w3c.dom.Node	**javax.jms.TextMessage**	DOM は **String** に変換されます。
マップ	**javax.jms.MapMessage**	
java.io.Serializable	**javax.jms.ObjectMessage**	
byte[]	**javax.jms.BytesMessage**	
java.io.File	**javax.jms.BytesMessage**	
java.io.Reader	**javax.jms.BytesMessage**	
java.io.InputStream	**javax.jms.BytesMessage**	

本文タイプ	JMS メッセージ	Comment
java.nio.ByteBuffer	javax.jms.BytesMessage	

JMS メッセージを受信するとき、Camel は JMS メッセージを以下のボディータイプに変換します。

JMS メッセージ	本文タイプ
javax.jms.TextMessage	文字列
javax.jms.BytesMessage	byte[]
javax.jms.MapMessage	Map<String, Object>
javax.jms.ObjectMessage	オブジェクト

169.5.1. JMS メッセージの自動マッピングの無効化

mapJmsMessage オプションを使用すると、上記の自動マッピングを無効にできます。無効な場合、Camel は受信した JMS メッセージをマッピングせず、代わりにペイロードとして直接使用します。これにより、マッピングのオーバーヘッドを回避し、Camel が JMS メッセージを通過させることのみを行うことができます。たとえば、クラスパスに ない クラスを持つ javax.jms.ObjectMessage JMS メッセージをルーティングすることもできます。

169.5.2. カスタム MessageConverter の使用

messageConverter オプションを使用して、Spring org.springframework.jms.support.converter.MessageConverter クラスでマッピングを独自に実行できます。

たとえば、以下のルートでは、メッセージを JMS 順序キューに送信するときにカスタムメッセージコンバーターを使用します。

```
from("file://inbox/order").to("jms:queue:order?messageConverter=#myMessageConverter");
```

JMS 宛先から消費する場合に、カスタムメッセージコンバーターを使用することもできます。

169.5.3. 選択したマッピングストラテジーの制御

エンドポイント URL で jmsMessageType オプションを使用して、すべてのメッセージに対して特定のメッセージタイプを強制できます。

以下のルートでは、JMS プロデューサーエンドポイントが強制的にテキストメッセージを使用するため、フォルダーからファイルをポーリングして javax.jms.TextMessage として送信します。

```
from("file://inbox/order").to("jms:queue:order?jmsMessageType=Text");
```

ヘッダーを CamelJmsMessageType キーに設定して、各メッセージに使用するメッセージタイプを指定することもできます。以下に例を示します。

```
from("file://inbox/order").setHeader("CamelJmsMessageType",
JmsMessageType.Text).to("jms:queue:order");
```

使用できる値は enum クラス org.apache.camel.jms.JmsMessageType で定義されます。

169.6. 送信時のメッセージ形式

JMS ネットワーク上で送信されるエクスチェンジは、JMS Message 仕様 に準拠する必要があります。

exchange.in.header の場合、以下のルールがヘッダー キー に適用されます。

- JMS または JMSX で始まるキーが予約されています。
- exchange.in.headers キーはリテラルで、すべて有効な Java 識別子である必要があります（キー名のドットは使用しないでください）。
- Camel は、JMS メッセージの使用時にドットとハイフンを置き換え ます。
. は DOT に置き換え、Camel がメッセージを消費すると逆の置換になります。

は HYPHEN に置き換えられます。は、Camel がメッセージを消費する際に逆の置換になります。

- *また、オプションの jmsKeyFormatStrategy も併せて参照してください。この場合、キーのフォーマットに独自のカスタムストラテジーを使用できます。*

exchange.in.header では、ヘッダーの 値 に以下のルールが適用されます。

- *値はプリミティブまたはカウンターオブジェクトでなければなりません（例：Integer、Long、Characterなど）。types、String、CharSequence、Date BigDecimal、および BigInteger はすべて toString（） 表現に変換されます。その他のタイプはすべてドロップされます。*

指定のヘッダー値をドロップすると、Camel はカテゴリー org.apache.camel.component.jms.JmsBinding を DEBUG レベルでログに記録します。以下に例を示します。

```
2008-07-09 06:43:04,046 [main           ] DEBUG JmsBinding
  - Ignoring non primitive header: order of class:
org.apache.camel.component.jms.issues.DummyOrder with value: DummyOrder{orderId=333,
itemId=4444, quantity=2}
```

169.7. 受信時のメッセージ形式

Camel は、メッセージを受信するときに以下のプロパティーを Exchange に追加します。

プロパティー	型	説明
org.apache.camel.jms.replyDestination	javax.jms.Destination	応答先。

Camel は、JMS メッセージの受信時に、以下の JMS プロパティーを In メッセージヘッダーに追加します。

ヘッダー	Type	説明
JMSCorrelationID	**文字列**	JMS 相関 ID。
JMSDeliveryMode	**int**	JMS 配信モード。
JMSDestination	**javax.jms.Destination**	JMS 宛先。
JMSExpiration	**Long**	JMS の有効期限。
JMSMessageID	**文字列**	JMS 固有のメッセージ ID。
JMSPriority	**int**	JMS の優先度（優先度が低い場合は 0、最も高い優先度は 9）。
JMSRedelivered	**boolean**	再配信された JMS メッセージです。
JMSReplyTo	**javax.jms.Destination**	JMS 応答先
JMSTimestamp	**Long**	JMS タイムスタンプ。
JMSType	**文字列**	JMS タイプ。
JMSXGroupID	**文字列**	JMS グループ ID。

上記の情報はすべて標準 JMS であるため、JMS のドキュメント で詳細を確認できます。

169.8. CAMEL を使用したメッセージおよび JMSREPLYTO の送受信

JMS コンポーネントは複雑で、場合によってはその動作に細心の注意を払う必要があります。そこで、検索する一部の領域/パイル(pitfall)の簡単な概要です。

Camel が JMSProducer を使用してメッセージを送信すると、以下の条件を確認します。

- メッセージ交換パターン
- JMSReplyTo がエンドポイントまたはメッセージヘッダーに設定されているかどうか。
- JMS エンドポイントで以下のオプションのいずれかが設定されているかどうか（ ReplyTo、preserveMessageQos、explicitQosEnabled ）。

これらはすべて、ユースケースに対応するように理解し、設定するために複雑になる可能性があります。

169.8.1. JmsProducer

JmsProducer は設定に応じて以下のように動作します。

Exchange パターン	その他のオプション	説明
InOut	-	Camel は応答を期待し、一時的な **JMSReplyTo** を設定します。メッセージの送信後に、一時キューのリプライメッセージをリッスンし始めます。
InOut	**JMSReplyTo** が設定されている	Camel はリプライを期待し、メッセージの送信後に、指定された **JMSReplyTo** キューのリプライメッセージをリッスンします。
InOnly	-	Camel はメッセージを送信し、リプライを期待し **ません**。

Exchange パターン	その他のオプション	説明
InOnly	**JMSReplyTo** が設定されている	デフォルトでは、Camel は **JMSReplyTo** 宛先を破棄し、メッセージを送信する前に **JMSReplyTo** ヘッダーをクリアします。その後 Camel はメッセージを送信し、応答を想定し **ません**。Camel は **WARN** レベルでログにロギングします（**Camel 2.6** 以降の **DEBUG** レベルに変更）。**preserveMessageQuo=true** を使用して、**JMSReplyTo** を保持するよう Camel に指示できます。すべての状況では、**JmsProducer** は応答を期待し **ない** ため、メッセージの送信後も続行されます。

169.8.2. JmsConsumer

JmsConsumer は、設定に応じて以下のように動作します。

Exchange パターン	その他のオプション	説明
InOut	-	Camel は応答を **JMSReplyTo** キューに送信します。
InOnly	-	パターンは InOnly であるため、Camel はリプライを返さません。
-	**disableReplyTo=true**	このオプションでは、応答が抑制されます。

したがって、エクスチェンジに設定されたメッセージ交換パターンに注意してください。

ルートの途中で JMS 宛先にメッセージを送信する場合は、使用する交換パターンを指定できます。詳細は「要求応答」を参照してください。
これは、InOnly メッセージを JMS トピックに送信する場合に便利です。

```
from("activemq:queue:in")
  .to("bean:validateOrder")
  .to(ExchangePattern.InOnly, "activemq:topic:order")
  .to("bean:handleOrder");
```

169.9. エンドポイントを再利用してランタイム時に計算される異なる宛先に送信する

多くの異なる JMS 宛先にメッセージを送信する必要がある場合は、JMS エンドポイントを再利用し、メッセージヘッダーに実際の宛先を指定することが理にかなっています。これにより、Camel は同じエンドポイントを再利用できますが、異なる宛先に送信することができます。これにより、メモリーやスレッドリソースで作成されたエンドポイントの数が大幅に削減されます。

以下のヘッダーで宛先を指定できます。

ヘッダー	Type	説明
CamelJmsDestination	javax.jms.Destination	宛先オブジェクト。
CamelJmsDestinationName	文字列	宛先名。

たとえば、以下のルートはランタイム時に宛先を計算し、これを使用して JMS URL に表示される宛先を上書きする方法を示しています。

```
from("file://inbox")
  .to("bean:computeDestination")
  .to("activemq:queue:dummy");
```

キュー名の ダミー はプレースホルダーです。JMS エンドポイント URL の一部として指定する必要がありますが、この例では無視されます。

computeDestination Bean で、以下のように CamelJmsDestinationName ヘッダーを設定して、実際の宛先を指定します。

```
public void setJmsHeader(Exchange exchange) {
   String id = ....
   exchange.getIn().setHeader("CamelJmsDestinationName", "order:" + id");
}
```

Camel はこのヘッダーを読み取り、エンドポイントに設定されたヘッダーの代わりに宛先として使用します。この例では、Camel はメッセージを activemq:queue:order:2 に送信します。

CamelJmsDestination と CamelJmsDestination Name ヘッダーの両方が設定されている場合、CamelJmsDestination が優先されます。JMS プロデューサーは、エクスチェンジから CamelJmsDestination および CamelJmsDestinationName ヘッダーの両方を削除し、ルートで誤ってループを回避するために作成された JMS メッセージに伝播されないことに注意してください（メッセージが別の JMS エンドポイントに転送されるシナリオ）。

169.10. 異なる JMS プロバイダーの設定

以下のように Spring XML で JMS プロバイダーを設定できます。

基本的に、必要なだけ JMS コンポーネントインスタンスを設定し、id 属性 を 使用して一意の名前を 指定することができます。上記の例は activemq コンポーネントを設定します。MQSeries、TibCo、BEA、Sonic などを設定するためにも同じことを実行できます。

名前付きの JMS コンポーネントを使用すると、URI を使用してそのコンポーネント内のエンドポイントを参照できます。たとえば、コンポーネント名 activemq の場合、URI 形式 activemq:[queue:|topic:]destinationName を使用して宛先を参照できます。他のすべての JMS プロバイダーで同じアプローチを使用できます。

これは、エンドポイント URI に使用するスキーム名に対して SpringCamelContext が Spring コンテキストからコンポーネントを取得し、Component がエンドポイント URI を解決させることで機能します。

169.10.1. JNDI を使用した接続ファクトリーの検索

J2EE コンテナーを使用している場合、Spring で通常の < bean > メカニズムを使用するのではなく、JNDI を検索して JMS ConnectionFactory を検索する必要がある場合があります。これは、Spring のファクトリー Bean または新しい Spring XML namespace を使用して実行できます。以下に例を示します。

```
<bean id="weblogic" class="org.apache.camel.component.jms.JmsComponent">
  <property name="connectionFactory" ref="myConnectionFactory"/>
</bean>

<jee:jndi-lookup id="myConnectionFactory" jndi-name="jms/connectionFactory"/>
```

JNDI ルックアップの詳細は、Spring リファレンスドキュメントの「The jee schema」を参照してください。

169.11. 同時消費

JMS の一般的な要件は、アプリケーションの応答を改善するために、複数のスレッドでメッセージを同時に消費することです。concurrentConsumers オプションを設定して、JMS エンドポイントを提供するスレッド数を指定できます。

```
from("jms:SomeQueue?concurrentConsumers=20").
  bean(MyClass.class);
```

このオプションは以下のいずれかの方法で設定できます。

- JmsComponent で、以下を実行します。
- エンドポイント URI で、以下を実行します。
- JmsEndpoint で setConcurrentConsumers（） を直接呼び出します。

169.11.1. 非同期コンシューマーの使用の同時使用

現在のメッセージが完全に処理されたときに、各同時コンシューマーは JMS ブローカーから次に利用可能なメッセージのみを取得することに注意してください。オプション asyncConsumer=true を設定して、コンシューマーが JMS キューから次のメッセージを選択できるようにしますが、以前のメッセージは非同期で処理されます（Asynchronous Routing Engine で）。asyncConsumer オプションの詳細は、ページ上部の表を参照してください。

```
from("jms:SomeQueue?concurrentConsumers=20&asyncConsumer=true").
  bean(MyClass.class);
```

169.12. JMS での REQUEST-REPLY

Camel は JMS 経由での Request Reply をサポートします。そこでは、JMS キューへメッセージを送信するときに、エクスチェンジの MEP は InOut である必要があります。

Camel は、パフォーマンスやクラスター環境に影響する JMS 上でリクエストや応答を設定する数多くのオプションを提供しています。以下の表は、オプションの概要を示しています。

オプション	パフォーマンス	Cluster	説明
temporary	高速	対応	一時キューはリプライキューとして使用され、Camel によって自動作成されます。これを使用するには、reclaimTo キュー名を指定し **ません**。また、任意で **replyToType=Temporary** を設定して、一時キューが使用されていることを示唆することができます。
共有	slow	対応	共有永続キューはリプライキューとして使用されます。キューは事前に作成する必要がありますが、一部のブローカーは Apache ActiveMQ などの場で作成できます。これを使用するには、replyTo キュー名を指定する必要があります。また、任意で **replyToType=Shared** を設定して、共有キューが使用中の状態にすることができます。共有キューは、この Camel アプリケーションを同時に実行している複数のノードを持つクラスター環境で使用できます。すべてで同じ共有応答キューを使用します。これは、JMS メッセージセレクターが想定されるリプライメッセージを関連付けるために使用されるため可能です。ただし、これはパフォーマンスに影響します。JMS メッセージセレクターは低速であるため、**Temporary** キューまたは **排他的** キューとして高速ではありません。パフォーマンスを向上させるには、以下の手順でこれを調整してください。
排他的	高速	No(*Yes)	排他的永続キューはリプライキューとして使用されます。キューは事前に作成する必要がありますが、一部のブローカーは Apache ActiveMQ などの場で作成できます。これを使用するには、replyTo キュー名を指定する必要があります。さらに、ReplyToType=Exclusive を設定して、**replyToType=Exclusive** が設定されている場合は、**Shared** がデフォルトで排他的キューを使用するように指示 **する 必要** があります。排他的応答キューを使用する場合、JMS メッセージセレクターは使用され **ない** ため、他のアプリケーションもこのキューを使用しないでください。排他的キューは、この Camel アプリケーションを実行している複数のノードが同時に使用することが **できません**。応答キューが要求メッセージを送信した同じノードに戻らなくても制御されません。共有キューが JMS メッセージセレクターを使用してこれを確認するのは、共有キューが JMS メッセージセレクターを使用してこれを確認するためです。ノードごとに一意の名前で各排他的リプライキューを設定する場合 **は**、これをクラスター環境で実行できます。その結果、リプライメッセージは指定のノードのそのキューに返信され、応答メッセージが待機します。
concurrentConsumers	高速	対応	Camel 2.10.3: 使用中の同時メッセージリスナーを使用して、リプライメッセージを同時に処理できます。**concurrentConsumers** オプションおよび **maxConcurrentConsumers** オプションを使用して範囲を指定できます。注記 : **共有 応答キューを使用している** と同時リスナーと機能できないので、このオプションを注意して使用してください。
maxConcurrentConsumers	高速	対応	Camel 2.10.3: 使用中の同時メッセージリスナーを使用して、リプライメッセージを同時に処理できます。**concurrentConsumers** オプションおよび **maxConcurrentConsumers** オプションを使用して範囲を指定できます。注記 : **共有 応答キューを使用している** と同時リスナーと機能できないので、このオプションを注意して使用してください。

JmsProducer は InOut を検出し、使用されるリプライ宛先を持つ JMSReplyTo ヘッダーを提供します。デフォルトでは、Camel は一時キューを使用しますが、エンドポイントで replyTo オプションを使用して、固定応答キューを指定できます（固定応答キューの詳細は以下を参照してください）。

Camel はリプライキューをリッスンするコンシューマーを自動的に設定するため、何もする必要はありません。
このコンシューマーは、返信をリッスンする Spring DefaultMessageListenerContainer です。ただし、同時コンシューマーは 1 に固定されています。
つまり、返信を処理する 1 つのスレッドしかないため、返信は順番に処理されます。応答の処理速度が速い場合は、同時実行を使用する必要があります。ただし、concurrentConsumer オプション は使用しません。以下のルートのように、代わりに Camel DSL からの スレッド を使用する必要があります。

スレッドを使用する代わりに、Camel 2.10.3 以上を使用する場合は concurrentConsumers オプションを使用します。詳細は以下を参照してください。

```
from(xxx)
.inOut().to("activemq:queue:foo")
.threads(5)
.to(yyy)
.to(zzz);
```

このルートでは、5 つのスレッドを持つスレッドプールを使用して返信を非同期でルーティングするよう Camel に指示します。

Camel 2.10.3 以降では、concurrentConsumers オプションおよび maxConcurrentConsumers オプションを使用して、同時スレッドを使用するようにリスナーを設定できるようになりました。これにより、以下のように Camel で簡単に設定することができます。

```
from(xxx)
.inOut().to("activemq:queue:foo?concurrentConsumers=5")
.to(yyy)
.to(zzz);
```

169.12.1. JMS でのリクエスト応答キューと共有された応答キューの使用

以下の例のように Request Reply over JMS の実行時に固定応答キューを使用する場合は、注意が必要です。

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar")
.to(yyy)
```

この例では、「bar」という名前の固定応答キューが使用されます。デフォルトでは、Camel は固定応答キューの使用時にキューが共有されているため、JMSSelector を使用して、予想される応答メッセージのみを選択します（ JMSCorrelationIDに基づいています）。特別な固定応答キューについては、次のセクションを参照してください。つまり、一時キューとして高速ではありません。receiveTimeout オプションを使用して、Camel がリプライメッセージにプルする頻度を高めることができます。デフォルトでは 1000 ミリ秒です。したがって、より速くするには、以下のように 1 秒あたり 4 回プルするように 250 ミリ秒に設定します。

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar&receiveTimeout=250")
.to(yyy)
```

これにより、Camel がメッセージブローカーにプルリクエストを送信するため、より多くのネットワークトラフィックが必要になります。
通常、可能な場合は一時キューを使用することが推奨されます。

169.12.2. JMS でのリクエスト応答キューと排他的な応答キューの使用

Camel 2.9 で利用可能

前述の例では、Camel は「bar」という名前の固定応答キューが共有されているため、JMSSelector を使用して、想定するリプライメッセージのみを消費します。ただし、JMS 選択速度は遅いため、これを行う欠点があります。また、リプライキューのコンシューマーは、新しい JMS セレクター ID で更新するために遅くなります。実際、receiveTimeout オプションがタイムアウトした場合にのみ更新されます。デフォルトは 1 秒です。そのため、応答メッセージの検出に約 1 秒の時間がかかる可能性があります。一方、固定応答キューが Camel 応答コンシューマーに排他的である場合、JMS セレクターの使用を回避できるため、より高性能になります。実際、一時キューを使用するのと同じ高速です。Camel 2.9 以降では、ReplyToType オプションが導入されました。これは、以下の例のように応答キューが排他 的であることを Camel に伝えるように設定できるようになりました。

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar&replyToType=Exclusive")
.to(yyy)
```

キューは、各エンドポイントおよびすべてのエンドポイントに対して排他的である必要があることに注意してください。そのため、2 つのルートがある場合、それぞれのルートには次の例に示すように一意の応答キューが必要です。

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar&replyToType=Exclusive")
.to(yyy)
```

```
from(aaa)
.inOut().to("activemq:queue:order?replyTo=order.reply&replyToType=Exclusive")
.to(bbb)
```

クラスター環境で実行する場合も同様です。次に、クラスターの各ノードは一意の応答キュー名を使用する必要があります。そうしないと、クラスターの各ノードは別のノードの応答として意図されたメッセージを取得する可能性があります。クラスター環境では、代わりに共有応答キューを使用することが推奨されます。

169.13. 送信側とレシーバー間のクロックの同期

システム間でメッセージングを実行する場合は、システムがクロックを同期させることが推奨されます。たとえば、JMS メッセージを送信する場合、メッセージをライブ値に設定できます。その後、受信側はこの値を検査し、メッセージの有効期限が切れているかどうかを判断するため、メッセージを消費して処理するのではなくドロップします。ただし、これには、送信側と受信側の両方にクロックが同期されている必要があります。ActiveMQ を使用している場合は、タイムスタンププラグイン を使用してクロックを同期できます。

169.14. ライブ時間

同期されたクロックに関する最初の読み取り。

Camel を使用して JMS 上でリクエスト/リプライ(InOut)を行う場合、Camel は送信者側でタイムアウトを使用します。デフォルトは requestTimeout オプションから 20 秒です。これを制御するには、より高い/低レイテンシーの値を設定します。ただし、ライブ値の持続時間は、送信される JMS メッセージに設定されます。したがって、システム間でクロックを同期する必要があります。有効でない場合は、ライブ値の設定時間を無効にする必要がある場合があります。これは、Camel 2.8 以降の disableTimeToLive オプションを使用できるようになりました。そのため、このオプションを disableTimeToLive=true に設定すると、Camel は JMS メッセージの送信時に常にライブ値を設定しません。ただし、 リクエストのタイムアウトは引き続きアクティブです。たとえば、JMS 上でリクエスト/応答を実行し、ライブ時間を無効にした場合、Camel は引き続き 20 秒のタイムアウトを使用します（ requestTimeout オプション）。このオプションは、ほとんどの場合設定することもできます。そのため、requestTimeout と disableTimeToLive の 2 つのオプションにより、リクエスト/応答の実行時により詳細な制御が可能になります。

Camel 2.13/2.12.3 以降では、メッセージにヘッダーを指定して、設定されたエンドポイントではなく、リクエストのタイムアウト値として使用できます。以下に例を示します。

```
from("direct:someWhere")
  .to("jms:queue:foo?replyTo=bar&requestTimeout=30s")
  .to("bean:processReply");
```

上記のルートでは、エンドポイント requestTimeout を 30 秒に設定します。そのため、Camel は応

答メッセージがバーキューで戻されるまで 30 秒待機します。応答メッセージを受信しないと、org.apache.camel.ExchangeTimedOutException がエクスチェンジに設定され、Camel はメッセージをルーティングを継続し、例外により失敗し、Camel のエラーハンドラーが反応します。

メッセージごとのタイムアウト値を使用する場合は、定数値 "CamelJmsRequestTimeout" が長い型を持つ org.apache.camel.component.jms.JmsConstants#JMS_REQUEST_TIMEOUT のキー org.apache.camel.component.jms.JmsConstants#JMS_REQUEST_TIMEOUT でヘッダーを設定することができます。

たとえば、以下のように Bean を使用して、サービス Bean の "whatIsTheTimeout" メソッドを呼び出すなど、個別のメッセージごとにタイムアウト値を算出することができます。

```
from("direct:someWhere")
  .setHeader("CamelJmsRequestTimeout", method(ServiceBean.class, "whatIsTheTimeout"))
  .to("jms:queue:foo?replyTo=bar&requestTimeout=30s")
  .to("bean:processReply");
```

Camel を使用して JMS 経由で(InOut)を実行し、忘れると、デフォルトでは Camel はメッセージをライブ値に設定し ません。timeToLive オプションを使用して値を設定できます。たとえば、timeToLive=5000 を設定します。disableTimeToLive オプションを使用すると、InOnly メッセージングでも、ライブ時間の無効化を強制できます。requestTimeout オプションは InOnly メッセージングには使用されません。

169.15. トランストラクトの有効化

一般的な要件は、トランザクションのキューから消費し、Camel ルートを使用してメッセージを処理することです。これを実行するには、コンポーネント/エンドポイントに以下のプロパティーを設定するだけです。

- transacted = true
- TransactionManager = Transsaction Manager （通常は JmsTransactionManager）

詳細は、「Transactional Client EIP パターン」を参照してください。

トランザクションおよび JMS 上の [Request Reply]

JMS で Request Reply を使用すると単一のトランザクションを使用することはできません。JMS はコミットが実行されるまでメッセージを送信しないため、サーバー側はトランザクションのコミットまで何も受信しません。そのため、要求応答 を使用する には、リクエストの送信後にトランザクションをコミットし、応答を受信するために別のトランザクションを使用する必要があります。

この問題に対応するために、JMS コンポーネントは異なるプロパティーを使用して一方向メッセージングおよび要求応答メッセージングにトランザクションの使用を指定します。

transacted プロパティーは、InOnly メッセージ交換パターン(MEP)に のみ 適用されます。

transactedInOut プロパティーは InOut(Request Reply)メッセージ交換パターン(MEP)に適用されます。

リクエスト応答(InOut MEP)にトランザクションを使用する場合は、transactedInOut=true を設定する 必要 があります。

Camel 2.10 で利用可能

コンポーネント/エンドポイントの以下のプロパティーを使用して、DMLC transacted セッション API を使用できます。

- transacted = true
- lazyCreateTransactionManager = false

この利点は、設定された TransactionManager なしでローカルトランザクションを使用する場合、cacheLevel 設定を適用することです。TransactionManager の設定時に、DMLC レベルでキャッシュが発生しず、プールされた接続ファクトリーに依存する必要はありません。このようなセットアップの詳細は、ここと こちら を参照してください。

169.16. 応答を遅らせるための JMSREPLYTO の使用

Camel を JMS リスナーとして使用する場合、Exchange プロパティーを ReplyTo javax.jms.Destination オブジェクトの値で設定し、キー ReplyTo を持ちます。この 宛先は以下のように 取得できます。

```
Destination replyDestination =
exchange.getIn().getHeader(JmsConstants.JMS_REPLY_DESTINATION, Destination.class);
```

その後、これを使用して通常の JMS または Camel を使用して応答を送信します。

```
    // we need to pass in the JMS component, and in this sample we use ActiveMQ
    JmsEndpoint endpoint = JmsEndpoint.newInstance(replyDestination,
activeMQComponent);
    // now we have the endpoint we can use regular Camel API to send a message to it
    template.sendBody(endpoint, "Here is the late reply.");
```

応答を送信する別のソリューションは、送信時に同じ Exchange プロパティーに replyDestination オブジェクトを提供することです。その後、Camel はこのプロパティーを取得し、実際の宛先に使用します。ただし、エンドポイント URI にはダミー宛先が含まれている必要があります。以下に例を示します。

```
    // we pretend to send it to some non existing dummy queue
    template.send("activemq:queue:dummy, new Processor() {
        public void process(Exchange exchange) throws Exception {
            // and here we override the destination with the ReplyTo destination object so the
message is sent to there instead of dummy
            exchange.getIn().setHeader(JmsConstants.JMS_DESTINATION, replyDestination);
            exchange.getIn().setBody("Here is the late reply.");
        }
    }
```

169.17. 要求タイムアウトの使用

以下の例では、Camel でさらに処理するために Request Reply スタイルのメッセージ Exchange(requestBody method = InOut)を低速なキューに送信し、返信応答を待機します。

169.18. サンプル

JMS は、他のコンポーネントについても多くの例で使用されます。ただし、以下の例を提供して開始しました。

169.18.1. JMS からの受信

以下の例では、JMS メッセージを受信し、メッセージを POJO にルーティングするルートを設定します。

```
    from("jms:queue:foo").
      to("bean:myBusinessLogic");
```

当然ながら任意の EIP パターンを使用することで、ルートをコンテキストベースとすることができます。たとえば、以下は、大規模なファクターで注文トピックをフィルタリングする方法を示しています。

```
from("jms:topic:OrdersTopic").
  filter().method("myBean", "isGoldCustomer").
    to("jms:queue:BigSpendersQueue");
```

169.18.2. JMS への送信

以下の例では、ファイルフォルダーをポーリングし、ファイルの内容を JMS トピックに送信します。ファイルの内容を BytesMessage ではなく TextMessage として使用するため、ボディーを String に変換する必要があります。

```
from("file://orders").
  convertBodyTo(String.class).
  to("jms:topic:OrdersTopic");
```

169.18.3. アノテーションの使用

Camel にはアノテーションがあるため、*POJO 消費と* POJO 生成を使用できます。

169.18.4. Spring DSL の例

上記の例では Java DSL を使用しています。Camel は Spring XML DSL もサポートします。以下は、Spring DSL を使用した大きなファクターのサンプルです。

```
<route>
  <from uri="jms:topic:OrdersTopic"/>
  <filter>
    <method bean="myBean" method="isGoldCustomer"/>
    <to uri="jms:queue:BigSpendersQueue"/>
  </filter>
</route>
```

169.18.5. その他のサンプル

JMS は、他のコンポーネントや EIP パターンの例の多くだけでなく、この Camel ドキュメントにも表示されます。ドキュメントをご参照ください。時間がある場合は、JMS を使用するチュートリア

ルを確認してくださいが、Spring Remoting と Camel がチュートリアル-JmsRemoting をどのように連携するかに重点を置いてください。

169.18.6. エクスチェンジを格納するデッドレターキューとしての JMS の使用

通常、JMS をトランスポートとして使用すると、本文とヘッダーのみをペイロードとして転送します。デッドレターチャネルで JMS を使用し、JMS キューを Dead Letter Queue として使用する場合、通常は原因となった例外は JMS メッセージに保存されません。ただし、JMS のデッドレターキューで transferExchange オプションを使用して、org.apache.camel.impl.DefaultExchangeHolder を保持する javax.jms.ObjectMessage としてエクスチェンジ全体をキューに保存するように指示します。これにより、Dead Letter Queue から消費し、キー Exchange.EXCEPTION_CAUGHT で Exchange プロパティーから原因の例外を取得できます。以下のデモは、以下を示しています。

```
// setup error handler to use JMS as queue and store the entire Exchange
errorHandler(deadLetterChannel("jms:queue:dead?transferExchange=true"));
```

その後、JMS キューから消費して問題を分析できます。

```
from("jms:queue:dead").to("bean:myErrorAnalyzer");

// and in our bean
String body = exchange.getIn().getBody();
Exception cause = exchange.getProperty(Exchange.EXCEPTION_CAUGHT, Exception.class);
// the cause message is
String problem = cause.getMessage();
```

169.18.7. JMS を Dead Letter Channel として使用するエラーのみを保存する

JMS を使用して原因のエラーメッセージを保存するか、または自分で初期化できるカスタムボディーを保存できます。以下の例では、Message Translator EIP を使用して、JMS のデッドレターキューに移動する前に、失敗したエクスチェンジで変換を行います。

```
// we sent it to a seda dead queue first
errorHandler(deadLetterChannel("seda:dead"));

// and on the seda dead queue we can do the custom transformation before its sent to the
JMS queue
from("seda:dead").transform(exceptionMessage()).to("jms:queue:dead");
```

ここでは、元の原因エラーメッセージのみを変換に保存します。ただし、任意の式を使用して任意の式を送信できます。たとえば、Bean でメソッドを呼び出すか、カスタムプロセッサーを使用できます。

169.19. INONLY メッセージの送信および JMSREPLYTO ヘッダーの維持

camel-jms を使用して JMS 宛先に送信する場合、プロデューサーは MEP を使用して InOnly または InOut メッセージングを検出します。ただし、InOnly メッセージを送信しつつ JMSReplyTo ヘッダーを維持する場合は、時折あります。これを維持するには、Camel に指示する必要があります。そうでないと、JMSReplyTo ヘッダーはドロップされます。

たとえば、InOnly メッセージを foo キューに送信しますが、バーキューを持つ JMSReplyTo では、以下のように実行できます。

```
template.send("activemq:queue:foo?preserveMessageQos=true", new Processor() {
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setBody("World");
        exchange.getIn().setHeader("JMSReplyTo", "bar");
    }
});
```

preserveMessageQos=true を使用して、JMSReplyTo ヘッダーを保持するよう Camel に指示します。

169.20. 宛先での JMS プロバイダーオプションの設定

IBM の WebSphere MQ などの一部の JMS プロバイダーでは、JMS 宛先にオプションを設定する必要があります。たとえば、targetClient オプションを指定する必要がある場合があります。targetClient は Camel URI オプションではなく WebSphere MQ オプションであるため、JMS 宛先名に設定する必要があります。

```
// ...
.setHeader("CamelJmsDestinationName", constant("queue:///MY_QUEUE?targetClient=1"))
.to("wmq:queue:MY_QUEUE?useMessageIDAsCorrelationID=true");
```

WMQ の一部のバージョンは宛先名でこのオプションを受け入れず、以下のような例外が発生します。

```
com.ibm.msg.client.jms.DetailedJMSException: JMSCC0005: The specified
value 'MY_QUEUE?targetClient=1' is not allowed for
'XMSC_DESTINATION_NAME'
```

回避策として、カスタムの DestinationResolver を使用できます。

```
JmsComponent wmq = new JmsComponent(connectionFactory);

wmq.setDestinationResolver(new DestinationResolver() {
    public Destination resolveDestinationName(Session session, String destinationName,
```

```
boolean pubSubDomain) throws JMSException {
        MQQueueSession wmqSession = (MQQueueSession) session;
        return wmqSession.createQueue("queue:///" + destinationName + "?targetClient=1");
    }
});
```

169.21. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- Transactional Client
- Bean インテグレーション
- Tutorial-JmsRemoting
- JMSTemplate gotchas

第170章 JMX コンポーネント

170.1. CAMEL JMX

Apache Camel は JMX への広範なサポートがあり、Camel 管理オブジェクトを JMX クライアントで監視および制御できます。

Camel は、MBean 通知へのサブスクライブを可能にする JMX コンポーネントも提供します。このページは、JMX を使用して Camel を管理および監視する方法です。

170.2. オプション

JMX コンポーネントにはオプションがありません。

JMX エンドポイントは、URI 構文を使用して設定します。

```
jmx:serverURL
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

170.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
serverURL	サーバー URL は、残りのエンドポイントから取得されます。		文字列

170.2.2. クエリーパラメーター（29 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
形式 （コンシューマー）	URI プロパティー： メッセージボディーのフォーマット。xml または raw のいずれか。xml の場合、通知は xml にシリアライズされます。raw の場合、生の java オブジェクトはボディーに設定されます。	xml	文字列
granularityPeriod (consumer)	URI プロパティー： モニターを確認するための Bean をポーリングする頻度のみを監視します。	10000	Long
monitorType (consumer)	URI プロパティー： 作成するモニターのタイプのみを監視します。文字列、ゲージ、カウンターの 1 つ。		文字列
objectDomain (consumer)	**必須** URI プロパティー： 接続している mbean のドメイン		文字列
objectName (consumer)	URI プロパティー： 接続している mbean の名前キー。この値は、渡されたオブジェクトプロパティーと相互に排他的です。		文字列
observedAttribute (consumer)	URI プロパティー： 監視 Bean を監視する属性のみを監視します。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
handback （詳細）	URI プロパティー： 通知が受信される際にリスナーに対話する値。この値は、キー jmx.handback のメッセージヘッダーに配置されます。		オブジェクト

Name	説明	デフォルト	Type
notificationFilter (advanced)	URI Property: NotificationFilter を実装する Bean への参照。		NotificationFilter
objectProperties (advanced)	URI Property: オブジェクト名のプロパティー。これらの値は、objectName パラメーターが設定されていない場合に使用されます。		マップ
reconnectDelay (advanced)	URI プロパティー： 最初の接続の確立を再試行する、または失われた接続の再接続を試みるまで待機する秒数	10	int
reconnectOnConnection Failure (advanced)	URI プロパティー： true の場合、接続障害が発生すると、コンシューマーは JMX サーバーへの再接続を試みます。コンシューマーは接続が確立されるまで 'x' 秒ごとに JMX 接続の再確立を試みます。'x' は設定済みの再接続Delay です。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
testConnectionOnStartup (advanced)	URI プロパティー： true の場合、起動時に JMX 接続を確立できない場合、コンシューマーは例外をスローします。false の場合、接続が設定される reconnectionDelay であるまで、コンシューマーは 'x' 秒ごとに JMX 接続を確立しようとします。	true	boolean
initThreshold (counter)	URI プロパティー： カウンターは、モニターの Initial threshold のみを監視します。この値は、通知が実行される前にこの値を超える必要があります。		int
modulus (counter)	URI プロパティー： カウンターがゼロにリセットされる値のみをカウンター監視		int
offset (counter)	URI プロパティー： カウンター監視。この値を超えると、しきい値を増加させます。		int
differenceMode (gauge)	URI プロパティー： カウンターゲージモニターは true の場合のみ、通知に報告される値は、値自体とは異なりしきい値とは異なります。	false	boolean
notifyHigh (gauge)	URI プロパティー： ゲージモニターは、高いしきい値を超えたときにゲージが通知を実行します。	false	boolean
notifyLow (gauge)	URI プロパティー： ゲージモニターは、低いしきい値を超えたときにゲージが通知を実行します。	false	boolean

Name	説明	デフォルト	Type
thresholdHigh (gauge)	URI プロパティー： ゲージのしきい値が高いしきい値のみの測定監視		double
thresholdLow (gauge)	URI プロパティー： ゲージモニターの値のみでは、ゲージのしきい値のしきい値のみ		double
パスワード （セキュリティー）	URI プロパティー： リモート接続を作成するためのクレデンシャル		文字列
ユーザー （セキュリティー）	URI プロパティー： リモート接続を作成するためのクレデンシャル		文字列
notifyDiffer (string)	URI Property: 文字列監視は、文字列属性が文字列と比較する文字列と異なる場合に、文字列モニターが通知を実行します。	false	boolean
notifyMatch (string)	URI Property: string monitor は、文字列属性が文字列と一致して比較する文字列と一致する場合に、文字列モニターが通知を実行します。	false	boolean
stringToCompare (string)	URI Property: 比較する文字列モニターの文字列の 値のみ を監視します。		文字列

170.3. CAMEL での JMX のアクティブ化

注記

Camel 2.8 以前に必要な Spring JAR 依存関係

spring-context.jar、spring-aop.jar、spring-beans.jar、および spring-core.jar は、JMX インストルメンテーションを使用できるように Camel によってクラスパスに必要です。これらの .jar がクラスパス上にない場合、Camel は非 JMX モードにフォールバックします。この状況は、ロガー名 org.apache.camel.impl.DefaultCamelContext を使用して WARN レベルでログに記録されます。

Camel 2.9 以降では、Spring JAR は Camel を JMX モードで実行することが 不要 になりました。

170.3.1. JMX を使用した Apache Camel の管理

デフォルトでは、JMX インストルメンテーションエージェントは Camel で有効になっています。つ

まり、Camel ランタイムは仮想マシンの MBeanServer インスタンスで MBean 管理オブジェクトを作成し、登録します。これにより、Camel ユーザーは Camel ルートが個別のプロセッサーレベルにダウンする方法に関する洞察をすぐに取得できます。

サポートされる管理オブジェクトは エンドポイント、ルート、サービス、および プロセッサー です。これらの管理オブジェクトの一部は、パフォーマンスカウンター属性の他にライフサイクル操作も公開します。

DefaultManagementNamingStrategy は、MBean 登録に使用されるオブジェクト名をビルドするデフォルトの命名ストラテジーです。デフォルトでは、org.apache.camel は CamelNamingStrategy によって作成されるすべてのオブジェクト名のドメイン名です。MBean オブジェクトのドメイン名は、Java VM システムプロパティーで設定できます。

```
-Dorg.apache.camel.jmx.mbeanObjectDomainName=your.domain.name
```

または、Spring 設定の camelContext 要素内に jmxAgent 要素を追加します。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" mbeanObjectDomainName="your.domain.name"/>
    ...
</camelContext>
```

Spring 設定は、両方が存在する場合は常にシステムプロパティーよりも優先されます。これは、すべての JMX 関連の設定で該当します。

170.3.2. Camel での JMX インストルメンテーションエージェントの無効化

JMX インストルメンテーションエージェントを無効にするには、以下のように Java VM システムプロパティーを設定します。

```
-Dorg.apache.camel.jmx.disabled=true
```

プロパティーの値は ブール 値として処理されます。

または、Spring 設定の camelContext 要素内に jmxAgent 要素を追加します。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" disabled="true"/>
    ...
```

```
</camelContext>
```

また、Camel 2.1 では、以下のように純粋な Java を使用する場合は、少し簡単に JVM システムプロパティーを使用する必要はありません。

```
CamelContext camel = new DefaultCamelContext();
camel.disableJMX();
```

170.3.3. Java VM での MBeanServer の検索

各 CamelContext には InstrumentationAgent のインスタンスが InstrumentationLifecycleStrategy 内でラップされます。InstrumentationAgent は、Camel MBean を登録するために MBeanServer とインターフェースするオブジェクトです。複数の CamelContext / InstrumentationAgent は MBean サーバーを共有することができます。デフォルトでは、Camel ランタイムは org.apache.camel のデフォルトのドメイン名に一致する MBeanServer Factory.findMBeanServer メソッド によって返される最初の MBeanServer を選択します。

アプリケーションですでに使用している MBeanServer インスタンスに一致するように、デフォルトのドメイン名を変更する必要がある場合があります。特に、MBeanServer が JMX コネクターサーバーに割り当てられている場合は、Camel でコネクターサーバーを作成する必要はありません。

システムプロパティーを使用して、一致するデフォルトドメイン名を設定できます。

```
-Dorg.apache.camel.jmx.mbeanServerDefaultDomain=<your.domain.name>
```

または、Spring 設定の camelContext 要素内に jmxAgent 要素を追加します。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" mbeanServerDefaultDomain="your.domain.name"/>
    ...
</camelContext>
```

一致する MBeanServer が見つからない場合は、新しい MBeanServer が作成され、上記のデフォルト設定に従って新しい「MBeanServer」のデフォルトドメイン名が設定されます。

また、システムプロパティーを設定して JVM MBean を管理する必要がある場合は、PlatformMBeanServer を使用することもできます。MBeanServer のデフォルトドメイン名設定は該当しないためは無視されます。

注意

次回のリリース(1.5)より、usePlatformMBeanServer のデフォルト値は true に変わります。プラットフォーム MBeanServer を使用すると、プロパティーを false に設定すると無効にできます。

```
-Dorg.apache.camel.jmx.usePlatformMBeanServer=True
```

または、Spring 設定の camelContext 要素内に jmxAgent 要素を追加します。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" usePlatformMBeanServer="true"/>
    ...
</camelContext>
```

170.3.4. JMX RMI コネクターサーバーの作成

JMX コネクターサーバーを使用すると、JConsole などの JMX クライアントによって MBean をリモートで管理できます。Camel JMX RMI コネクターサーバーは、システムプロパティーを設定し、Camel によって使用される MBeanServer がそのコネクターサーバーにアタッチすることで、任意で有効にすることができます。

```
-Dorg.apache.camel.jmx.createRmiConnector=True
```

または、Spring 設定の camelContext 要素内に jmxAgent 要素を追加します。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" createConnector="true"/>
    ...
</camelContext>
```

170.3.5. JMX サービス URL

デフォルトの JMX サービス URL の形式は、以下のようになります。

```
service:jmx:rmi:///jndi/rmi://localhost:<registryPort>/<serviceUrlPath>
```

registryPort は RMI レジストリーポートで、デフォルト値は 1099 です。

システムプロパティーで RMI レジストリーポートを設定できます。

```
-Dorg.apache.camel.jmx.rmiConnector.registryPort=<port number>
```

または、Spring 設定の camelContext 要素内に jmxAgent 要素を追加します。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" createConnector="true" registryPort="port number"/>
    ...
</camelContext>
```

serviceUrlPath は URL のパス名で、デフォルト値は /jmxrmi/camel です。

システムプロパティーでサービス URL パスを設定できます。

```
-Dorg.apache.camel.jmx.serviceUrlPath=<path>
```

ヒント

Java での ManagementAgent の設定

Camel 2.4 以降では、ManagementAgent でさまざまなオプションを設定することもできます。

```
context.getManagementStrategy().getManagementAgent().setServiceUrlPath("/foo/bar");
context.getManagementStrategy().getManagementAgent().setRegistryPort(2113);
context.getManagementStrategy().getManagementAgent().setCreateConnector(true);
```

または、Spring 設定の camelContext 要素内に jmxAgent 要素を追加します。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" createConnector="true" serviceUrlPath="path"/>
    ...
</camelContext>
```

デフォルトでは、RMI サーバーオブジェクトは動的に生成されたポートをリッスンします。これは、ファイアウォールを介して確立される接続に問題となる可能性があります。このような状況では、システムプロパティーによって RMI 接続ポートを明示的に設定できます。

```
-Dorg.apache.camel.jmx.rmiConnector.connectorPort=<port number>
```

または、Spring 設定の camelContext 要素内に jmxAgent 要素を追加します。

```
<camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring">
  <jmxAgent id="agent" createConnector="true" connectorPort="port number"/>
    ...
</camelContext>
```

コネクターポートオプションが設定されると、JMX サービス URL は以下のようになります。

```
service:jmx:rmi://localhost:<connectorPort>/jndi/rmi://localhost:<registryPort>/<serviceUrlPath>
```

170.3.6. Camel JMX サポート用のシステムプロパティー

プロパティー名	value	説明
org.apache.camel.jmx	**true** または **false**	**true** の場合、Camel で jmx 機能を有効にします。

*本セクションのその他のシステムプロパティーは、**jmxAgent Properties Reference** を参照してください。*

170.3.7. JMX で認証を使用する方法

JDK の JMX には、認証用の機能があり、SSL 経由でセキュアな接続を使用することもできます。この使用方法は、SUN ドキュメントを参照してください。

- *http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html*
- *http://java.sun.com/javase/6/docs/technotes/guides/management/agent.html*

170.3.8. アプリケーションサーバー内の JMX

170.3.8.1. Tomcat 6

Tomcat で JMX を有効にする方法は、このページ を参照してください。

つまり、catalina.sh（Windows では catalina.bat）ファイルを変更して、以下のオプションを設定します。

```
set CATALINA_OPTS=-Dcom.sun.management.jmxremote \
  -Dcom.sun.management.jmxremote.port=1099 \
  -Dcom.sun.management.jmxremote.ssl=false \
  -Dcom.sun.management.jmxremote.authenticate=false
```

170.3.8.2. JBoss AS 4

デフォルトでは、JBoss は独自の MBeanServer を作成します。Camel が同じサーバーに公開できるようにするには、以下の手順に従います。

1. プラットフォーム MBeanServer を使用するように Camel に指示します（デフォルトは Camel 1.5 で true です）。

```
<camel:camelContext id="camelContext">
  <camel:jmxAgent id="jmxAgent" mbeanObjectDomainName="org.yourname"
usePlatformMBeanServer="true"  />
</camel:camelContext>
```

1. プラットフォーム MBeanServer を使用するように JBoss インスタンスを変更します。run.sh または run.conf -Djboss.platform.mbeanserver を編集して、以下のプロパティーを JAVA_OPTS に追加します。See http://wiki.jboss.org/wiki/JBossMBeansInJConsole

170.3.8.3. WebSphere

mbeanServerDefaultDomain を WebSphere に変更します。

```
<camel:jmxAgent id="agent" createConnector="true" mbeanObjectDomainName="org.yourname"
usePlatformMBeanServer="false" mbeanServerDefaultDomain="WebSphere"/>
```

170.3.8.4. Oracle OC4j

Oracle OC4J J2EE アプリケーションサーバーは、Camel がプラットフォーム MBeanServer にアクセスすることはできません。これは、Camel によって WARNING をログに記録するため、ログに特定することができます。

```
xxx xx, xxxx xx:xx:xx xx org.apache.camel.management.InstrumentationLifecycleStrategy
onContextStart
WARNING: Could not register CamelContext MBean
```

```
java.lang.SecurityException: Unauthorized access from application: xx to MBean:
java.lang:type=ClassLoading
        at
oracle.oc4j.admin.jmx.shared.UserMBeanServer.checkRegisterAccess(UserMBeanServer.java:873)
```

これを解決するには、Camel で JMX エージェントを無効にする必要があります。「Disabling JMX instrumentation agent in Camel」を参照してください。

170.3.9. 高度な JMX 設定

Spring 設定ファイルでは、Camel が管理のために JMX に公開される方法を設定できます。コネクターのポートやパス名など、ここに詳細情報を指定できる場合があります。

170.3.10. 例:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <jmxAgent id="agent" createConnector="true" registryPort="2000"
mbeanServerDefaultDomain="org.apache.camel.test"/>
    <route>
      <from uri="seda:start"/>
      <to uri="mock:result"/>
    </route>
</camelContext>
```

Java 5 JMX 設定を変更する場合は、さまざまな JMX システムプロパティーを使用できます。

たとえば、以下の環境変数を設定することで、Sun JMX コネクターへのリモート JMX 接続を有効にできます（プラットフォームに応じて 設定 または エクスポート を使用）。これらの設定は、Camel がデフォルトで作成する JMX コネクターではなく、Java 1.5+ 内の Sun JMX コネクターのみを設定します。

```
SUNJMX=-Dcom.sun.management.jmxremote=true -Dcom.sun.management.jmxremote.port=1616 \
-Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false
```

（SUNJMX 環境変数は、JVM の追加起動パラメーターとして Camel の起動スクリプトが簡単に使用できます。Camel を直接開始する場合は、これらのパラメーターを自分で渡す必要があります。）

170.3.11. jmxagent プロパティーのリファレンス

Spring プロパティー	システムプロパティー	デフォルト値	説明

Spring プロパティー	システムプロパティー	デフォルト値	説明
id			JMX エージェント名。オプションではありません
usePlatformMBeanServer	**org.apache.camel.jmx.usePlatformMBeanServer**	**false**、**true** - リリース 1.5 以降	**true** の場合、JVM から **MBeanServer** を使用します。
mbeanServerDefaultDomain	**org.apache.camel.jmx.mbeanServerDefaultDomain**	**org.apache.camel**	**MBeanServer**のデフォルトの JMX ドメイン
mbeanObjectDomainName	**org.apache.camel.jmx.mbeanObjectDomainName**	**org.apache.camel**	すべてのオブジェクト名が使用する JMX ドメイン
createConnector	**org.apache.camel.jmx.createRmiConnect**	**false**	**MBeanServer**に JMX コネクター（リモート管理を許可する）を作成する場合は、以下を行います。
registryPort	**org.apache.camel.jmx.rmiConnector.registryPort**	**1099**	JMX RMI レジストリーが使用するポート
connectorPort	**org.apache.camel.jmx.rmiConnector.connectorPort**	-1 (dynamic)	JMX RMI サーバーが使用するポート
serviceUrlPath	**org.apache.camel.jmx.serviceUrlPath**	**/jmxrmi/camel**	JMX コネクターが登録されるパス
onlyRegisterProcessorWithCustomId	**org.apache.camel.jmx.onlyRegisterProcessorWithCustomId**	**false**	Camel 2.0: このオプションが有効な場合には、カスタム ID セットを持つプロセッサーのみが登録されます。これにより、JMX コンソールで不要なプロセッサーをファイルアウトできます。

Spring プロパティー	システムプロパティー	デフォルト値	説明
statisticsLevel		**すべての / デフォルト**	**Camel 2.1:** MBean に対してパフォーマンス統計が有効になっているかどうかのレベルを設定します。詳細は、**パフォーマンス統計の粒度のレベルの設定** について参照してください。**Camel 2.16** 以降では、All オプションの名前が Default に変更され、追加のランタイム JMX メトリクスを収集できる新しい Extended オプションが導入されました。
includeHostName	**org.apache.camel.jmx.includeHostName**		**Camel 2.13:** MBean 命名にホスト名を含めるかどうか。Camel 2.13 以降では、これはデフォルトの **false** です。この場合、古いリリースのデフォルトが **true** となります。本当に必要に応じて、このオプションを使用して以前の動作を復元できます。
useHostIPAddress	**org.apache.camel.jmx.useHostIPAddress**	**false**	**Camel 2.16:** リモートコネクターの作成時にサービス URL でホスト名または IP アドレスを使用するかどうか。デフォルトでは、ホスト名が使用されます。
loadStatisticsEnabled	**org.apache.camel.jmx.loadStatisticsEnabled**	**false**	**Camel 2.16:**負荷統計を有効にするかどうか（CamelContext ごとのバックグラウンドスレッドを使用した負荷統計の収集）。
endpointRuntimeStatisticsEnabled	**org.apache.camel.jmx.endpointRuntimeStatisticsEnabled**	**true**	**Camel 2.16:** エンドポイントランタイム統計が有効になっているかどうか（受信および送信エンドポイントのランタイム使用状況を収集）。

170.3.12. 新しいルート、またはデフォルトで MBean を常に登録するかどうかの設定

Camel 2.7 で利用可能

Camel は、mbeans を登録するかどうかを制御する 2 つの設定を提供

オプション	デフォルト	説明
registerAlways	**false**	有効にすると、MBean は常に登録されます。
registerNewRoutes	**true**	有効にすると、CamelContext の起動後に新しいルートを追加すると、指定のルートから MBean も登録されます。

デフォルトでは、Camel は起動時に設定されたすべてのルートに MBean を登録します。registerNewRoutes オプションは、後で新しいルートを追加する場合に MBean も登録すべきかどうかを制御します。たとえば、管理が必要でない一時的なルートを追加および削除する場合に、これを無効にすることができます。

Recipient List に固有のエンドポイントを持つ Recipient List などの動的 EIP パターンを使用する場合は、registerAlways オプションを使用してください。その場合、一意のエンドポイントとその関連サービス/プロデューサーも登録されます。これにより、レジストリーに mbeans が多数あるため、システムの低下が生じる可能性がありました。MBean は軽量オブジェクトではないため、メモリーを消費します。

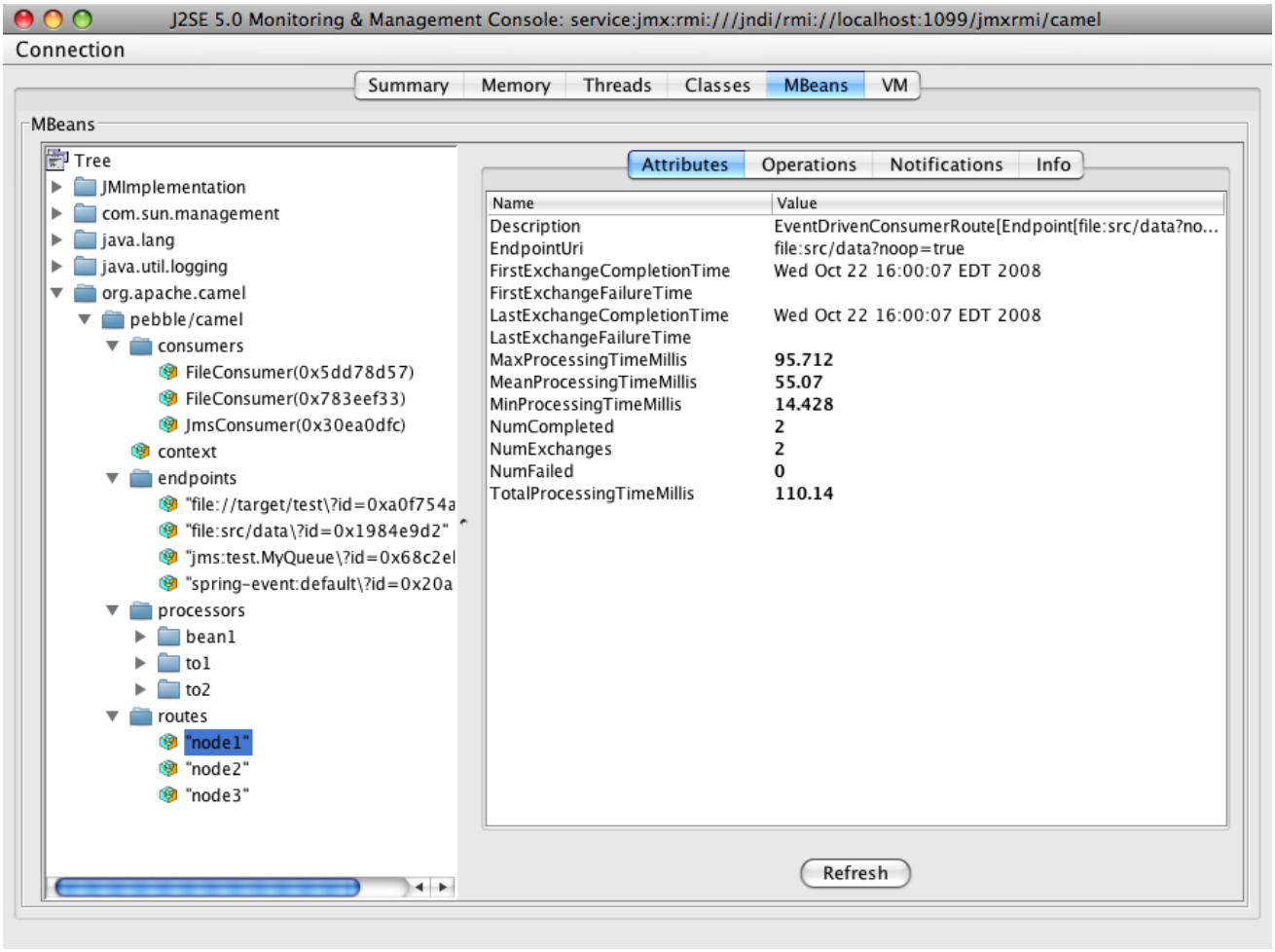
170.4. JMX を使用した CAMEL の監視

170.4.1. JConsole を使用した Camel の監視

Camel と同じホストで JConsole を実行している場合は、ローカル接続のリストに CamelContext が表示されるはずです。

リモートの Camel インスタンスへ接続する場合、またはローカルプロセスが表示されない場合は、Remote Process オプションを使用して URL を入力します。以下は、localhost URL:service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi/camel の例です。

JConsole での Apache Camel の使用：



170.4.2. 登録されているエンドポイント

Camel 2.1 以降では、シングルトン エンドポイント のみ が登録されます。シングルトン以外のオーバーヘッドは、数千または数百万のエンドポイントが使用される場合に大きな登録されるためです。これは、Recipient List EIP を使用する場合や、多くのメッセージを送信する ProducerTemplate から発生する可能性があります。

170.4.3. どのプロセッサーが登録されているか

この FAQ を参照してください。

170.4.4. JMX NotificationListener を使用して Camel イベントをリッスンする方法

Camel 通知イベントは、発生する内容の粒度の細かい概要を提供します。コンテキストおよびエンドポイントからライフサイクルイベントが表示され、エンドポイントへ受信および送信されるエクスチェンジを確認することができます。

Camel 2.4 以降 では、カスタム JMX NotificationListener を使用して Camel イベントをリッスンすることができます。

最初に CamelContext を起動する前に JmxNotificationEventNotifier を設定する必要があります。

```
// Set up the JmxNotificationEventNotifier
notifier = new JmxNotificationEventNotifier();
notifier.setSource("MyCamel");
notifier.setIgnoreCamelContextEvents(true);
notifier.setIgnoreRouteEvents(true);
notifier.setIgnoreServiceEvents(true);

CamelContext context = new DefaultCamelContext(createRegistry());
context.getManagementStrategy().addEventNotifier(notifier);
```

2 番目に、イベントをリッスンするためにリスナーを登録することができます。

```
// register the NotificationListener
ObjectName on = ObjectName.getInstance("org.apache.camel:context=camel-
1,type=eventnotifiers,name=JmxEventNotifier");
MyNotificationListener listener = new MyNotificationListener();
context.getManagementStrategy().getManagementAgent().getMBeanServer().addNotificationLi
stener(on,
    listener,
    new NotificationFilter() {
        private static final long serialVersionUID = 1L;

        public boolean isNotificationEnabled(Notification notification) {
            return notification.getSource().equals("MyCamel");
        }
    }, null);
```

170.4.5. トレーサー MBean を使用した詳細なトレースの取得

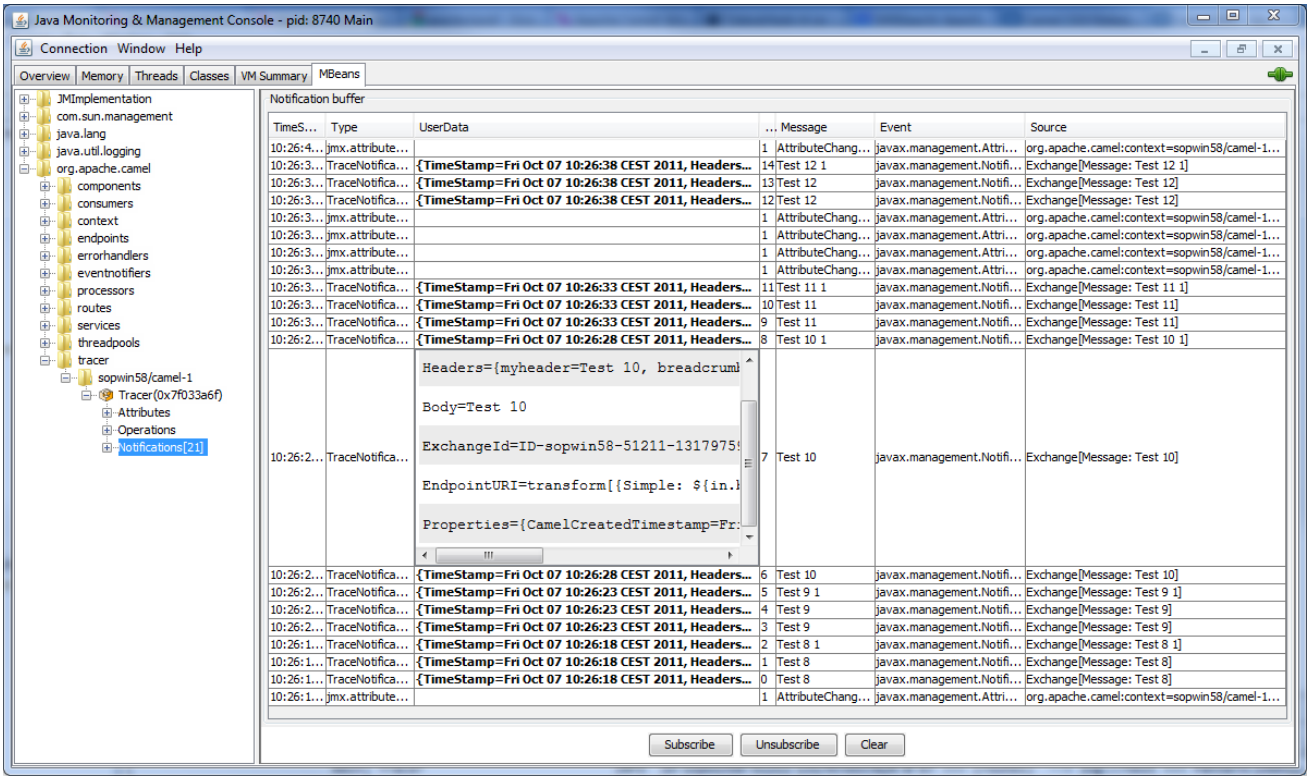
詳細なトレースイベントのために、Camel 2.9.0 より粒度の細かい通知に加えて JMX 通知がサポートされます。

これらは Tracer MBean にあります。詳細なトレースを有効にするには、まずコンテキストまたはルートでトレースをアクティベートする必要があります。

これは、コンテキストの設定時またはコンテキスト / ルート MBean で行うことが可能です。

2 番目のステップでは、トレーサーで jmxTraceNotifications 属性を true に設定する必要があります。これは、コンテキストの設定時またはトレーサー MBean でランタイム時に実行できます。

JConsole を使用して、Tracer MBean でトレースイベント通知に登録できるようになりました。すべてのエクスチェンジおよびメッセージの詳細と共に、ルートのすべてのステップで通知が 1 つあります。



170.5. 独自の CAMEL コードでの JMX の使用

170.5.1. 独自の管理エンドポイントの登録

Camel 2.0
で利用できます。独自のエンドポイントを Spring で管理されるアノテーション @ManagedResource で分離し、Camel MBeanServer に登録でき、JMX を使用してカスタム MBean にアクセスできます。

注記

Camel 2.1 では、エンドポイント以外のものを適用するようにこれを変更しましたが、インターフェース org.apache.camel.spi.ManagementAware も実装する必要があります。詳細は、こちらを参照してください。

たとえば、管理するオプションを定義する以下のカスタムエンドポイントがあります。

```
@ManagedResource(description = "Our custom managed endpoint")
public class CustomEndpoint extends MockEndpoint implements
ManagementAware<CustomEndpoint> {

    public CustomEndpoint(final String endpointUri, final Component component) {
        super(endpointUri, component);
    }

    public Object getManagedObject(CustomEndpoint object) {
        return this;
    }

    public boolean isSingleton() {
        return true;
    }

    protected String createEndpointUri() {
        return "custom";
    }

    @ManagedAttribute
    public String getFoo() {
        return "bar";
    }

    @ManagedAttribute
    public String getEndpointUri() {
        return super.getEndpointUri();
    }
}
```

Camel 2.9 以降では、org.apache.camel.api.management パッケージの @ManagedResource、@ManagedAttribute、および @ManagedOperation を使用することが推奨されています。これにより、カスタムコードは Spring JAR に依存しなくなります。

170.5.2. 独自のマネージドサービスのプログラミング

Camel 2.1 から利用可能

Camel は、管理するサービスを登録する際に独自の MBean を使用できるようになりました。たとえば、カスタム Camel コンポーネントを開発し、エンドポイント、コンシューマー、プロデューサーの MBean を公開することができます。org.apache.camel.spi.ManagementAware インターフェースを実装し、Camel が使用する必要のある管理オブジェクトを返すことだけです。

では、JMX API は非常に優れて悪いと思われる前に、お気に入りが適切です。しかし、Spring にも関わらず、既存の Bean で管理をエクスポートするために使用できるアノテーションの範囲を作成しました。つまり、これ は頻繁に使用し、ManagementAware インターフェースから getManagedObject で返します。たとえば、CustomEndpoint を使用した上記の例を参照してください。

Camel 2.1 では 、Camel が管理のために登録するすべてのオブジェクトに対して、非常に bunch な管理に対してこれを行うことができますが、すべてではありません。

この ManagementAware インターフェースを実装するサービスの場合、Camel は、以下の表で定義されているデフォルトのラッパーを使用するようにフォールバックします。

Type	MBean ラッパー
CamelContext	ManagedCamelContext
コンポーネント	ManagedComponent
エンドポイント	ManagedEndpoint
コンシューマー	ManagedConsumer
プロデューサー	ManagedProducer
ルート	ManagedRoute
プロセッサー	ManagedProcessor
トレーサー	ManagedTracer
サービス	ManagedService

さらに、以下のような特殊なタイプのラッパーもいくつかあります。

Type	MBean ラッパー
ScheduledPollConsumer	ManagedScheduledPollConsumer
BrowsableEndpoint	ManagedBrowseableEndpoint
Throttler	ManagedThrottler

Type	MBean ラッパー
Delayer	**ManagedDelayer**
SendProcessor	**ManagedSendProcessor**

また、今後は、より多くの EIP パターン向けにラッパーを追加します。

170.5.3. ManagementNamingStrategy

Camel 2.1 から利用可能

Camel は org.apache.camel.spi.ManagementNamingStrategy による命名ストラテジーのプラグ可能な API を提供します。デフォルトの実装は、すべての MBean が登録される MBean 名を計算するために使用されます。

170.5.4. 管理命名パターン

Camel 2.10 で利用可能

Camel 2.10 以降では、MBean の命名パターンの設定を容易にします。このパターンは、ドメイン名の後にキーとして ObjectName の一部として使用されます。

デフォルトでは、Camel は以下のように ManagedCamelContextMBean に MBean 名を使用します。

```
org.apache.camel:context=localhost/camel-1,type=context,name=camel-1
```

Camel 2.13 以降では、ホスト名は MBean 名に含まれないため、上記の例は以下のようになります。

```
org.apache.camel:context=camel-1,type=context,name=camel-1
```

CamelContext で名前を設定すると、その名前は ObjectName に含まれます。たとえば、以下があるとします。

```
<camelContext id="myCamel" ...>
```

この場合、MBean 名は以下のようになります。

```
org.apache.camel:context=localhost/myCamel,type=context,name=myCamel
```

JVM に命名の競合がある場合（上記の名前が指定された MBean がすでに存在しているなど）、Camel はデフォルトでカウンターを使用して JMXMBeanServer で新しい空き名を見つけようとします。カウンターが追加されたように、ObjectName の一部として myCamel-1 があります。

```
org.apache.camel:context=localhost/myCamel-1,type=context,name=myCamel
```

これは、Camel はデフォルトで以下のトークンをサポートする命名パターンを使用するためです。

- camelId = the CamelContext id (eg the name)
- name - same as camelId
- counter - インクリメントカウンター * bundleId - OSGi バンドル ID（OSGi 環境のみ）
- symbolicName: OSGi シンボリック名（OSGi 環境のみ）
- バージョン: OSGi バンドルバージョン（OSGi 環境のみ）

デフォルトの命名パターンは、OSGi と非 OSGi を区別します。

- OSGI 以外の： 名前
- OSGi: bundleId-name
- OSGi Camel 2.13

しかし、JMXMBeanServer に命名の競合がある場合、Camel は自動的にフォールバックを行い、パターンの カウンター を使用してこれを変更します。そのため、以下のパターンが使用されます。

- OSGI 以外の - name-counter
- OSGi: bundleId-name-counter
- OSGi Camel 2.13: symbolicName-counter

明示的な命名パターンを設定した場合、そのパターンは常に使用され、上記のデフォルトのパターンは使用され ません。

これにより、レジストリー内の CamelContext ID と JMXMBeanRegistry の JMX MBean の両方に対する命名を完全に制御し、非常に簡単に制御できます。

Camel 2.15 以降では、JVM システムプロパティーを使用してデフォルトの管理名パターンを設定し、これを JVM に対してグローバルに設定できます。以下の例のように、このパターンを明示的に設定することで、このパターンを上書きできます。

JVM システムプロパティーを設定して、その名前の前に名前を追加するデフォルトの管理名パターンを使用します。

```
System.setProperty(JmxSystemPropertyKeys.MANAGEMENT_NAME_PATTERN, "cool-#name#");
```

したがって、CamelContext を明示的に名前を付け、固定 MBean 名を使用する場合は、変更しない（カウンターはありません）、新しい managementNamePattern 属性を使用できます。

```
<camelContext id="myCamel" managementNamePattern="#name#">
```

この場合、MBean 名は常に以下のようになります。

```
org.apache.camel:context=localhost/myCamel,type=context,name=myCamel
```

Java では、以下のように managementNamePattern を設定できます。

```
context.getManagementNameStrategy().setNamePattern("#name#");
```

また、id 以外の名前を managementNamePattern で使用することができるため、たとえば以下を行うことができます。

```
<camelContext id="myCamel" managementNamePattern="coolCamel">
```

OSGi バンドル ID を MBean 名の一部として望ましくない場合に、OSGi 環境ではこの設定を行うことができます。サーバーを再起動するか、同じアプリケーションのアンインストールおよびインストールを行うと、OSGi バンドル ID が変更される可能性があります。次に、以下のように、OSGi バンドル ID を名前の一部として使用しないようにすることができます。

```
<camelContext id="myCamel" managementNamePattern="#name#">
```

これには、myCamel は JVM 全体で一意である必要があることに注意してください。同じ CamelContext id と managementNamePattern を持つ 2 番目の Camel アプリケーションをインストールすると、Camel は起動時に失敗し、MBean はすでに存在する例外を報告します。

170.5.5. ManagementStrategy

Camel 2.1 から利用可能

Camel は、管理の制御における完全にプラグ可能な管理ストラテジーを提供するようになりました。管理には多くのメソッドが含まれるリッチインターフェースです。MBeanServer への管理オブジェクトの追加および削除だけでなく、イベント通知も org.apache.camel.spi.EventNotifier API を使用して提供されます。たとえば、他の管理製品へのアダプターの提供が容易になります。さらに、Apache で追加設定なしで提供される詳細情報や機能も提供できます。

170.5.6. パフォーマンス統計の粒度のレベルの設定

Camel 2.1 から利用可能

Camel の起動時にパフォーマンス統計が有効になっているかどうかを事前に設定できるようになりました。レベルは以下のようになります。

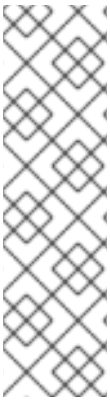
-

拡張: デフォルトとして、エンドポイントの使用状況を細かく指定するなど、ランタイム時に収集される追加の統計が提供されます。このオプションには Camel 2.16 が必要です。

- *All / Default: Camel はルートおよびプロセッサーの両方の統計を有効にします（詳細にわたります）。Camel 2.16 以降では、All オプションの名前が Default に変更されました。*
- *RoutesOnly: Camel はルートの統計のみを有効にします（詳細）*
- *off - Camel は統計を有効にしません。*

Camel 2.9 以降では、パフォーマンスの統計には、CamelContext および Route MBean ごとの平均負荷統計が含まれます。統計は、1 分、5 分、15 分間のインフライトエクスチェンジの数に基づく平均負荷です。これは、Unix システムの負荷統計と似ています。Camel 2.11 以降では、< jmxAgent> に loadStatisticsEnabled=false を設定することで、負荷パフォーマンスの統計を明示的に無効にできます。静的レベルも off に設定されている場合、オフになります。Camel 2.13 以降では、負荷パフォーマンスの統計はデフォルトで無効になっています。これを有効にするには、< jmxAgent> に loadStatisticsEnabled=true を設定します。

ランタイムでは、常に管理コンソール（JConsole など）を使用して、統計が有効かどうかに関わらず、指定のルートやプロセッサーで変更できます。



注記

統計が有効な意味は何ですか？

統計を有効にすると、Camel はその特定の MBean のパフォーマンス統計を細かく設定することを意味します。完了/失敗、last/total/mina/max/mean 処理時間、first/last failed time など、表示される統計は多数あります。

Java DSL を使用すると、以下でこのレベルを設定します。

```
// only enable routes when Camel starts
context.getManagementStrategy().setStatisticsLevel(ManagementStatisticsLevel.RoutesOnly);
```

Spring DSL から以下を行います。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <jmxAgent id="agent" statisticsLevel="RoutesOnly"/>
 ...
</camelContext>
```

170.6. 機密情報の非表示

Camel 2.12 から利用可能

デフォルトでは、URI を使用して設定されたエンドポイントなど、Camel は JMX で MBean を登録します。この設定では、パスワードなどの機密情報が存在する可能性があります。

この情報は、以下のように マスク オプションを有効にすることで非表示にすることができます。

Java DSL を使用すると、以下が可能になります。

```
// only enable routes when Camel starts
context.getManagementStrategy().getManagementAgent().setMask(true);
```

Spring DSL から以下を行います。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <jmxAgent id="agent" mask="true"/>
 ...
</camelContext>
```

これにより、パスワードやパスフレーズなどのオプションを持つ URI をマスクし、xxxxxx を置き換えの値として使用します。

170.6.1. マスクする JMX 属性および操作を宣言する

org.apache.camel.api.management.ManagedAttribute と org.apache.camel.api.management.ManagedOperation では、属性 マスク を true に設定し、この JMX 属性/操作の結果をマスクする必要があることを示します（JMX エージェントで有効にされている場合）。

たとえば、camel-core org.apache.camel.api.management.mbean.ManagedEndpointMBean からのデフォルトの管理エンドポイントでは、EndpointUri JMX 属性がマスクされていることを宣言し

ています。

```
@ManagedAttribute(description = "Endpoint URI", mask = true)
String getEndpointUri();
```

170.7. 関連項目

- 管理の例
- JConsole でプロセッサーが表示されない理由

第171章 JOLT コンポーネント

Camel バージョン 2.16 から利用可能

jolt: コンポーネントを使用すると、JOLT 仕様を使用して JSON メッセージを処理することができます。これは、JSON を JSON 変換する場合に適しています。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jolt</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

171.1. URI 形式

```
jolt:specName[?options]
```

specName は、呼び出す仕様のクラスパスローカル URI、またはリモート仕様の完全な URL です（例： file://folder/myfile.json）。

URI にクエリーオプションを追加するには、 ?option=value&option=value&...

171.2. オプション

JOLT コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
変換（詳細）	使用するトランスフォーマーを明示的に設定します。設定されていない場合は、transformDsl によって指定された Transform が作成されます。		変換

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JOLT エンドポイントは、URI 構文を使用して設定します。

```
jolt:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

171.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceUri	リソースへの **必須** パス。プレフィックス：classpath、file、http、ref、または bean. classpath、file、http は、これらのプロトコルを使用してリソースをロードします（classpath はデフォルト）。ref はレジストリーのリソースを検索します。Bean はリソースとして使用される Bean でメソッドを呼び出します。Bean には、ドットの後にメソッド名を指定できます（例：bean:myBean.myMethod）。		文字列

171.2.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
inputType (producer)	入力が引用された JSON または JSON 文字列であるかを指定します。	Hydrated	JoltInputOutputType
outputType (producer)	出力が JSON または JSON String を識別すべきかどうかを指定します。	Hydrated	JoltInputOutputType

Name	説明	デフォルト	Type
transformDsl (producer)	エンドポイントリソースの Transform DSL を指定します。指定がない場合は、Chainr が使用されます。	Chainr	JoltTransformType
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

171.3. サンプル

たとえば、以下のようなものを使用できます。

```
from("activemq:My.Queue").
  to("jolt:com/acme/MyResponse.json");
```

ファイルベースのリソースの場合：

```
from("activemq:My.Queue").
  to("jolt:file://myfolder/MyResponse.json?contentCache=true").
  to("activemq:Another.Queue");
```

また、以下のように、コンポーネントをヘッダーで動的に使用する仕様を指定することもできます。

```
from("direct:in").
  setHeader("CamelJoltResourceUri").constant("path/to/my/spec.json").
  to("jolt:dummy");
```

171.4. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*

- *エンドポイント*
- *はじめに*

第172章 JPA COMPONENT（JPA コンポーネント）

Camel バージョン 1.0 で利用可能

jpa コンポーネントを使用すると、EJB 3 の Java Persistence Architecture(JPA)を使用して永続ストレージから Java オブジェクトを保存および取得できます。これは、OpenJPA、Hibernate、TopLink などのオブジェクト/Relational Mapping(ORM)製品をラップする標準インターフェースレイヤーです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jpa</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

172.1. エンドポイントへの送信

Java エンティティー Bean をデータベースに保存するには、これを JPA プロデューサーエンドポイントに送信します。In メッセージのボディーはエンティティー bean（つまり @Entity アノテーションを持つ POJO）またはエンティティー Bean のコレクションまたは配列であると見なされます。

ボディーがエンティティーの一覧である場合は、プロデューサーエンドポイントに渡される設定として entityType=java.util.ArrayList を使用するようにしてください。

ボディーに前述のタイプの 1 つが含まれていない場合は、エンドポイントの前に Message Translator を置き、最初に必要な変換を実行します。

Camel 2.19 以降では、プロデューサーにQuery または nativeQuery の クエリー を使用することもできます。また、パラメーター の値でも Simple 式を使用できます。これにより、メッセージボディー、ヘッダーなどからパラメーター値を取得できます。これらのクエリーは、SELECT JPQL/SQL ステートメントを使用した一連のデータの取得に使用できます。また、UPDATE/DELETE JPQL/SQL ステートメントを使用して一括更新/削除を実行するために使用できます。namedQuery で UPDATE/DELETE を実行すると、query および nativeQuery とは異なり名前付きクエリーを確認しない場合、useExecuteUpdate を true に指定する必要があります。

172.2. エンドポイントからの消費

JPA コンシューマーエンドポイントからメッセージを消費すると、データベース内のエンティティー Bean を削除（または更新）します。これにより、データベーステーブルを論理キューとして使用できます。コンシューマーはキューからメッセージを取得し、それらを削除/更新してキューから論理的に削除できます。

エンティティー Bean が処理されたときに削除したくない場合は（ルーティングが完了したら）、URI で consumeDelete=false を指定できます。これにより、エンティティーがポーリングごとに処理されます。

エンティティーで更新を実行して（今後のクエリーから除外するなど）、@Consumed でメソッドにアノテーションを付けることができます。これは、処理時にエンティティー Bean で呼び出される（およびルーティングが完了したとき）エンティティー Bean で呼び出されます。

Camel 2.13 以降では、@PreConsumed を使用できます。これは、処理前にエンティティー Bean で呼び出されます（ルーティング前）。

多数の行(100K+)を使用し、OutOfMemory の問題が発生している場合は、maximumResults を適切な値に設定する必要があります。

172.3. URI 形式

```
jpa:entityClassName[?options]
```

エンドポイントに送信する場合、entityClassName は任意です。指定されている場合は、Type Converter でボディーが正しいタイプであることを確認します。

使用するには、entityClassName は必須です。

URI にクエリーオプションを追加するには、?option=value&option=value&...

172.4. オプション

JPA コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
entityManagerFactory (common)	EntityManagerFactory を使用します。この設定が強く推奨されます。		EntityManagerFactory
transactionManager (common)	トランザクションの管理に PlatformTransactionManager を使用します。		PlatformTransaction Manager
joinTransaction (common)	camel-jpa コンポーネントはデフォルトでトランザクションに参加します。このオプションを使用するとこれをオフにすることができます。たとえば、LOCAL_RESOURCE を使用し、結合トランザクションが JPA プロバイダーと動作しません。このオプションは、すべてのエンドポイントに設定する代わりに、JpaComponent でグローバルに設定することもできます。	true	boolean
sharedEntityManager (common)	コンシューマー/プロデューサーに Spring の SharedEntityManager を使用するかどうか。ほとんどの場合、joinTransaction は EXTENDED EntityManager ではないため、false に設定する必要があります。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JPA エンドポイントは URI 構文を使用します。

```
jpa:entityType
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

172.4.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
entityType	**必須**。エンティティーとして使用する JPA アノテーションが付けられたクラス。		class<?>

172.4.2. クエリーパラメーター（42 パラメーター）：

Name	説明	デフォルト	Type
joinTransaction (common)	camel-jpa コンポーネントはデフォルトでトランザクションに参加します。このオプションを使用するとこれをオフにすることができます。たとえば、LOCAL_RESOURCE を使用し、結合トランザクションが JPA プロバイダーと動作しません。このオプションは、すべてのエンドポイントに設定する代わりに、JpaComponent でグローバルに設定することもできます。	true	boolean
maximumResults (common)	Query で取得する結果の最大数を設定します。	-1	int
namedQuery (common)	名前付きクエリーを使用するには、以下を実行します。		文字列
nativeQuery (common)	カスタムのネイティブクエリーを使用します。ネイティブクエリーを使用する場合も resultClass オプションを使用することが望ましい場合があります。		文字列
パラメーター （共通）	このキーと値のマッピングは、クエリーパラメーターの構築に使用されます。キーは指定された JPA クエリーの名前付きパラメーターで、選択する対応する有効な値である java.util.Map 汎用型 java.util.Map が使用されることが予想されます。プロデューサーに使用される場合、Simple 式をパラメーター値として使用できます。これにより、メッセージボディー、ヘッダーなどからパラメーター値を取得できます。		マップ
persistenceUnit (common)	デフォルトで使用される JPA 永続ユニットが **必要です**。	Camel	文字列
クエリー （共通）	カスタムクエリーを使用するには、以下を行います。		文字列
resultClass (common)	返されるペイロードのタイプを定義します（ここでは entityManager.createNativeQuery(nativeQuery)ではなく entityManager.createNativeQuery(nativeQuery)を呼び出します）。このオプションを指定しないと、オブジェクトアレイを返します。データの消費時にネイティブクエリーと併用する場合にのみ影響があります。		class<?>
sharedEntityManager (common)	コンシューマー/プロデューサーに Spring の SharedEntityManager を使用するかどうか。ほとんどの場合、joinTransaction は EXTENDED EntityManager ではないため、false に設定する必要があります。	false	boolean

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumeDelete (consumer)	true の場合、エンティティーは消費後に削除されます。false の場合は、エンティティーは削除されません。	true	boolean
consumeLockEntity (consumer)	ポーリングからの結果を処理する間に、各エンティティー bean に排他ロックを設定するかどうかを指定します。	true	boolean
deleteHandler (consumer)	コンシューマーがエクスチェンジの処理が完了した後にカスタム DeleteHandler を使用して行を削除するには、以下を実行します。		Object>
lockModeType (consumer)	コンシューマーでロックモードを設定します。	PESSIMISTIC_WRITE	LockModeType
maxMessagesPerPoll (consumer)	ポーリングごとに収集するメッセージの最大数を定義する整数値。デフォルトでは、最大値は設定されません。サーバーを起動すると、数千ものメッセージをポーリングしないようにできます。無効にするには、0 または negative の値を設定します。		int
preDeleteHandler (consumer)	コンシューマーがエンティティーを読み取った後にカスタム Pre-DeleteHandler を使用して行を削除するには、以下を行います。		Object>
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
skipLockedEntity (consumer)	ロックで NOWAIT を使用し、警告なしでエンティティーをスキップするかどうかを設定します。	false	boolean

Name	説明	デフォルト	Type
トランザクション（コンシューマー）	バッチ全体が処理されたときにすべてのメッセージがコミットまたはロールバックされるトランザクションモードでコンシューマーを実行するかどうか。デフォルトの動作(false)は、以前に処理されたすべてのメッセージをコミットし、最後に失敗したメッセージのみをロールバックします。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
flushOnSend (producer)	エンティティー Bean が永続化された後に EntityManager をフラッシュします。	true	boolean
削除（プロデューサー）	entityManager.remove(entity)を使用することを示します。	false	boolean
useExecuteUpdate (producer)	プロデューサーがクエリーを実行する際に executeUpdate（）を使用するかどうかを設定します。INSERT、UPDATE、または DELETE ステートメントを名前付きクエリーとして使用する場合は、このオプションを 'true' に指定する必要があります。		ブール値
usePassedInEntityManager (producer)	true に設定すると、Camel はコンポーネント/エンドポイントの設定済みのエンティティーマネージャーの代わりに、ヘッダー JpaConstants.ENTITYMANAGER から EntityManager を使用します。これにより、エンドユーザーは使用中のエンティティーマネージャーを制御できます。	false	boolean

Name	説明	デフォルト	Type
usePersist (producer)	entityManager.merge（entity の代わりに entityManager.persist(entity)を使用することを示します。注記： entityManager.persist(entity)は、分離されたエンティティーでは動作しません（EntityManager は INSERT クエリーの代わりに UPDATE を実行する必要がある場合）。	false	boolean
entityManagerProperties (advanced)	使用するエンティティーマネージャーの追加プロパティー。		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel

Name	説明	デフォルト	Type
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

172.5. メッセージヘッダー

Camel は以下のメッセージヘッダーをエクスチェンジに追加します。

ヘッダー	Type	説明
CamelJpaTemplate	**JpaTemplate**	Camel 2.12 **以降はサポート対象外になり** ました。エンティティー Bean へのアクセスに使用される **JpaTemplate** オブジェクト。このオブジェクトは、型コンバーターやカスタム処理を行う場合などに必要になります。このヘッダーのサポートが削除された理由は、CAMEL-5932 を参照してください。
CamelEntityManager	**EntityManager**	Camel 2.12: JPA consumer / Camel 2.12.2: JPA producer: **JpaConsumer** または **JpaProducer** によって使用される JPA **EntityManager** オブジェクト。

172.6. CONFIGURING ENTITYMANAGERFACTORY

特定の EntityManagerFactory インスタンスを使用するよう JPA コンポーネントを設定することを

強く推奨します。これを実行しない場合、各 JpaEndpoint は必要でない EntityManagerFactory の独自のインスタンスを自動作成します。

たとえば、以下のように myEMFactory エンティティーマネージャーファクトリーを参照する JPA コンポーネントをインスタンス化することができます。

```
<bean id="jpa" class="org.apache.camel.component.jpa.JpaComponent">
  <property name="entityManagerFactory" ref="myEMFactory"/>
</bean>
```

Camel 2.3 では、JpaComponent は Registry から EntityManagerFactory を自動検出するため、上記のように JpaComponent でこれを設定する必要はありません。あいまいな場合にのみこれを実行する必要があります。その場合、Camel は WARN をログに記録します。

172.7. TRANSACTIONMANAGER の設定

Camel 2.3 以降、JpaComponent は Registry から TransactionManager を自動的に検索します。Camel が登録された TransactionManager インスタンスが見つからない場合は、TransactionTemplate も検索し、そこから TransactionManager を抽出してみてください。

レジストリーで使用された TransactionTemplate がない場合、JpaEndpoint は必要でない トランザクションManager の独自のインスタンスを自動作成します。

TransactionManager の複数のインスタンスが見つかった場合は、Camel が WARN をログに記録します。このような場合は、以下のように myTransactionManager トランザクションマネージャーを参照する JPA コンポーネントをインスタンス化し、明示的に設定したい場合があります。

```
<bean id="jpa" class="org.apache.camel.component.jpa.JpaComponent">
  <property name="entityManagerFactory" ref="myEMFactory"/>
  <property name="transactionManager" ref="myTransactionManager"/>
</bean>
```

172.8. 名前付きクエリーでコンシューマーの使用

選択したエンティティーのみを消費する場合は、consumer.namedQuery URI クエリーオプションを使用できます。まず、JPA Entity クラスに名前付きクエリーを定義する必要があります。

```
@Entity
@NamedQuery(name = "step1", query = "select x from MultiSteps x where x.step = 1")
public class MultiSteps {
```

```
    ...
}
```

その後、以下のようにコンシューマー URI を定義できます。

```
from("jpa://org.apache.camel.examples.MultiSteps?consumer.namedQuery=step1")
.to("bean:myBusinessLogic");
```

172.9. クエリーでのコンシューマーの使用

選択したエンティティーのみを消費する場合は、consumer.query URI クエリーオプションを使用できます。クエリーオプションは定義する必要があります。

```
from("jpa://org.apache.camel.examples.MultiSteps?consumer.query=select o from
org.apache.camel.examples.MultiSteps o where o.step = 1")
.to("bean:myBusinessLogic");
```

172.10. ネイティブクエリーでのコンシューマーの使用

選択したエンティティーのみを消費するには、consumer.nativeQuery URI クエリーオプションを使用できます。ネイティブクエリーオプションを定義する必要があります。

```
from("jpa://org.apache.camel.examples.MultiSteps?consumer.nativeQuery=select * from
MultiSteps where step = 1")
.to("bean:myBusinessLogic");
```

ネイティブクエリーオプションを使用する場合、メッセージボディーにオブジェクトアレイを受信します。

172.11. 名前付きクエリーでのプロデューサーの使用

選択したエンティティーを取得したり、一括更新/削除を実行したりする場合は、namedQuery URI クエリーオプションを使用できます。まず、JPA Entity クラスに名前付きクエリーを定義する必要があります。

```
@Entity
@NamedQuery(name = "step1", query = "select x from MultiSteps x where x.step = 1")
public class MultiSteps {
    ...
}
```

その後、以下のようにプロデューサー URI を定義できます。

```
from("direct:namedQuery")
.to("jpa://org.apache.camel.examples.MultiSteps?namedQuery=step1");
```

UPDATE/DELETE ステートメントを名前付きクエリーとして実行するには、useExecuteUpdate オプションを true に指定する必要があります。

172.12. クエリーでのプロデューサーの使用

選択したエンティティーを取得したり、一括更新/削除を実行したりする場合は、クエリー URI クエリー オプションを使用できます。クエリーオプションは定義する必要があります。

```
from("direct:query")
.to("jpa://org.apache.camel.examples.MultiSteps?query=select o from
org.apache.camel.examples.MultiSteps o where o.step = 1");
```

172.13. ネイティブクエリーでのプロデューサーの使用

選択したエンティティーを取得したり、一括更新/削除を実行したりする場合は、nativeQuery URI クエリーオプションを使用できます。ネイティブクエリーオプションを定義する必要があります。

```
from("direct:nativeQuery")
.to("jpa://org.apache.camel.examples.MultiSteps?
resultClass=org.apache.camel.examples.MultiSteps&nativeQuery=select * from MultiSteps
where step = 1");
```

resultClass を指定せずにネイティブクエリーオプションを使用する場合、メッセージボディーにオブジェクト配列を受け取ります。

172.14. 例

JPA を使用してトレースされたメッセージをデータベースに保存する例は、「トレーサー の例」を参照してください。

172.15. JPA ベースのべき等リポジトリーの使用

EIP パターン のべき等コンシューマーは、重複メッセージをフィルターするために使用されます。JPA ベースのべき等リポジトリーが提供されます。

JPA ベースのべき等リポジトリーを使用します。

手順

1. *persistence.xml ファイルで persistence-unit を設定します。*
2. *org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository によって使用される org.springframework.orm.jpa.JpaTemplate を設定します。*
3. *error format マクロの設定： snippet: java.lang.IndexOutOfBoundsException: Index: 20, Size: 20*
4. *べき等リポジトリーを設定します(org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository)。*
5. *Spring XML ファイルに JPA べき等リポジトリーを作成します。*

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route id="JpaMessageIdRepositoryTest">
    <from uri="direct:start" />
    <idempotentConsumer messageIdRepositoryRef="jpaStore">
      <header>messageId</header>
      <to uri="mock:result" />
    </idempotentConsumer>
  </route>
</camelContext>
```

IDE 内で Camel コンポーネントテストを実行する場合

Maven ではなく、IDE 内で このコンポーネントのテスト を直接実行する場合は、以下のような例外が表示されます。

```
org.springframework.transaction.CannotCreateTransactionException: Could not open JPA
EntityManager for transaction; nested exception is
<openjpa-2.2.1-r422266:1396819 nonfatal user error>
org.apache.openjpa.persistence.ArgumentException: This configuration disallows runtime
optimization,
but the following listed types were not enhanced at build time or at class load time with a
```

```
javaagent: "org.apache.camel.examples.SendEmail".
    at
org.springframework.orm.jpa.JpaTransactionManager.doBegin(JpaTransactionManager.java:427)
    at
org.springframework.transaction.support.AbstractPlatformTransactionManager.getTransaction(AbstractPlatformTransactionManager.java:371)
    at
org.springframework.transaction.support.TransactionTemplate.execute(TransactionTemplate.java:127)
    at org.apache.camel.processor.jpa.JpaRouteTest.cleanupRepository(JpaRouteTest.java:96)
    at
org.apache.camel.processor.jpa.JpaRouteTest.createCamelContext(JpaRouteTest.java:67)
    at org.apache.camel.test.junit4.CamelTestSupport.doSetUp(CamelTestSupport.java:238)
    at org.apache.camel.test.junit4.CamelTestSupport.setUp(CamelTestSupport.java:208)
```

この問題は、ソースが IDE 経由でコンパイルまたは再コンパイルされ、Maven ではなく IDE でコンパイルされているため、ビルド時にバイトコードを強化 します。これに対応するには、OpenJPA の動的バイトコード拡張 を有効にする必要があります。たとえば、Camel で現在使用されている OpenJPA のバージョンが 2.2.1 で、IDE 内でテストを実行するには、以下の引数を JVM に渡す必要があります。

```
-javaagent:
<path_to_your_local_m2_cache>/org/apache/openjpa/openjpa/2.2.1/openjpa-2.2.1.jar
```

172.16. 関連項目

- Configuring Camel (Camel の設定)
- Component (コンポーネント)
- Endpoint (エンドポイント)
- はじめに
- トレーサーの例

第173章 JSON FASTJSON DATAFORMAT

Camel バージョン 2.20 で利用可能

Fastjson は Fastjson ライブラリーを使用するデータ形式です。

```
from("activemq:My.Queue").
  marshal().json(JsonLibrary.Fastjson).
  to("mqseries:Another.Queue");
```

173.1. FASTJSON オプション

JSon Fastjson データフォーマットは 19 個のオプションをサポートします。これらのオプションは以下のとおりです。

Name	デフォルト	Java タイプ	説明
objectMapper		文字列	Jackson を使用する際に、指定の ID で既存の ObjectMapper を検索し、使用します。
useDefaultObjectMapper	true	ブール値	レジストリーからデフォルトの Jackson ObjectMapper を検索し、使用するかどうか。
prettyPrint	false	ブール値	プリションでフォーマットされた出力を有効にします。デフォルトは false です。
library	XStream	JsonLibrary	使用する json ライブラリー。
unmarshalTypeName		文字列	無効化解除時に使用する java タイプのクラス名
jsonView		class<?>	POJO を JSON にマーシャリングする場合は、JSON 出力から特定のフィールドを除外したい場合があります。Jackson を使用すると、JSON ビューを使用してこれを実行できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		文字列	pojo を JSON にマーシャリングし、pojo には null 値を持つフィールドがあります。また、これらの null 値を省略する場合は、このオプションを NOT_NULL に設定します。

Name	デフォルト	Java タイプ	説明
allowJmsType	**false**	**ブール値**	JMS 仕様からの JMSType ヘッダーがアンマーシャリングに使用する FQN クラス名を指定するために JMS ユーザーに使用されます。
collectionTypeName		**文字列**	使用するレジストリーで検索するカスタムコレクションタイプを参照します。このオプションはほとんど使用されることはありませんが、デフォルトとして java.util.Collection とは異なるコレクションタイプを使用できます。
useList	**false**	**ブール値**	マップのリストまたは Pojo のリストへ無視する場合は、以下を行います。
enableJaxbAnnotationModule	**false**	**ブール値**	Jackson を使用する際に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson により JAXB アノテーションを使用できます。
moduleClassNames		**文字列**	FQN クラス名で String として指定されたカスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用します。複数のクラスをコンマで区切ることができます。
moduleRefs		**文字列**	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		**文字列**	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。
disableFeatures		**文字列**	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能セット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。

Name	デフォルト	Java タイプ	説明
permissions		文字列	xml/json から Java Bean へのアンマーシャリング中に使用可能な Java パッケージとクラス XStream を制御するパーミッションを追加します。パーミッションは、ここに設定するか、JVM システムプロパティーを使用してグローバルに設定する必要があります。パーミッションは、プラス記号が allow の構文で指定でき、マイナス記号は deny になります。ワイルドカードは、. をプレフィックスとして使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、specfy com.foo を使用します。com.foo.,-com.foo.bar.MySecretBean など、複数のパーミッションをコンマで区切って設定できます。以下のデフォルトパーミッションは常に、JVM システムプロパティーと org.apache.camel.xstream.permissions キーを指定して上書きされない限り、-,java.lang.,java.util. です。
allowUnmarshallType	false	ブール値	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する場合にのみ有効にしてください。
timezone		文字列	設定された場合、Jackson はマーシャリング/アンマーシャリング時にタイムゾーンを使用します。このオプションは、gson、fastjson、xstream などの他の Json DataFormat には影響を与えません。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

173.2. 依存関係

camel ルートで Fastjson を使用するには、このデータ形式を実装する camel-fastjson の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-fastjson</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第174章 JSON GSON DATAFORMAT

Camel バージョン 2.10 で利用可能

Gson は、Gson ライブラリーを使用するデータ形式です。

```
from("activemq:My.Queue").
  marshal().json(JsonLibrary.Gson).
  to("mqseries:Another.Queue");
```

174.1. GSON オプション

JSon GSon データフォーマットは、以下に示す 19 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
objectMapper		文字列	Jackson を使用する際に、指定の ID で既存の ObjectMapper を検索し、使用します。
useDefaultObjectMapper	true	ブール値	レジストリーからデフォルトの Jackson ObjectMapper を検索し、使用するかどうか。
prettyPrint	false	ブール値	プリションでフォーマットされた出力を有効にします。デフォルトは false です。
library	XStream	JsonLibrary	使用する json ライブラリー。
unmarshalTypeName		文字列	無効化解除時に使用する java タイプのクラス名
jsonView		class<?>	POJO を JSON にマーシャリングする場合は、JSON 出力から特定のフィールドを除外したい場合があります。Jackson を使用すると、JSON ビューを使用してこれを実行できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		文字列	pojo を JSON にマーシャリングし、pojo には null 値を持つフィールドがあります。また、これらの null 値を省略する場合は、このオプションを NOT_NULL に設定します。

Name	デフォルト	Java タイプ	説明
allowJmsType	**false**	**ブール値**	JMS 仕様からの JMSType ヘッダーがアンマーシャリングに使用する FQN クラス名を指定するために JMS ユーザーに使用されます。
collectionTypeName		**文字列**	使用するレジストリーで検索するカスタムコレクションタイプを参照します。このオプションはほとんど使用されることはありませんが、デフォルトとして java.util.Collection とは異なるコレクションタイプを使用できます。
useList	**false**	**ブール値**	マップのリストまたは Pojo のリストへ無視する場合は、以下を行います。
enableJaxbAnnotationModule	**false**	**ブール値**	Jackson を使用する際に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson により JAXB アノテーションを使用できます。
moduleClassNames		**文字列**	FQN クラス名で String として指定されたカスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用します。複数のクラスをコンマで区切ることができます。
moduleRefs		**文字列**	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		**文字列**	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。
disableFeatures		**文字列**	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能セット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。

Name	デフォルト	Java タイプ	説明
permissions		文字列	xml/json から Java Bean へのアンマーシャリング中に使用可能な Java パッケージとクラス XStream を制御するパーミッションを追加します。パーミッションは、ここに設定するか、JVM システムプロパティーを使用してグローバルに設定する必要があります。パーミッションは、プラス記号が allow の構文で指定でき、マイナス記号は deny になります。ワイルドカードは、. をプレフィックスとして使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、specfy com.foo を使用します。com.foo.,-com.foo.bar.MySecretBean など、複数のパーミッションをコンマで区切って設定できます。以下のデフォルトパーミッションは常に、JVM システムプロパティーと org.apache.camel.xstream.permissions キーを指定して上書きされない限り、-,java.lang.,java.util. です。
allowUnmarshallType	false	ブール値	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する場合にのみ有効にしてください。
timezone		文字列	設定された場合、Jackson はマーシャリング/アンマーシャリング時にタイムゾーンを使用します。このオプションは、gson、fastjson、xstream などの他の Json DataFormat には影響を与えません。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

174.2. 依存関係

camel ルートで Gson を使用するには、このデータ形式を実装する camel-gson の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-gson</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第175章 JSON JACKSON DATAFORMAT

Camel バージョン 2.0 で利用可能

Jackson は、Jackson ライブラリーを使用するデータ形式です。

```
from("activemq:My.Queue").
  marshal().json(JsonLibrary.Jackson).
  to("mqseries:Another.Queue");
```

175.1. JACKSON オプション

JSon Jackson データフォーマットは、以下に示す 19 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
objectMapper		文字列	Jackson を使用する際に、指定の ID で既存の ObjectMapper を検索し、使用します。
useDefaultObjectMapper	true	ブール値	レジストリーからデフォルトの Jackson ObjectMapper を検索し、使用するかどうか。
prettyPrint	false	ブール値	プリションでフォーマットされた出力を有効にします。デフォルトは false です。
library	XStream	JsonLibrary	使用する json ライブラリー。
unmarshalTypeName		文字列	無効化解除時に使用する java タイプのクラス名
jsonView		class<?>	POJO を JSON にマーシャリングする場合は、JSON 出力から特定のフィールドを除外したい場合があります。Jackson を使用すると、JSON ビューを使用してこれを実行できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		文字列	pojo を JSON にマーシャリングし、pojo には null 値を持つフィールドがあります。また、これらの null 値を省略する場合は、このオプションを NOT_NULL に設定します。

Name	デフォルト	Java タイプ	説明
allowJmsType	**false**	**ブール値**	JMS 仕様からの JMSType ヘッダーがアンマーシャリングに使用する FQN クラス名を指定するために JMS ユーザーに使用されます。
collectionTypeName		**文字列**	使用するレジストリーで検索するカスタムコレクションタイプを参照します。このオプションはほとんど使用されることはありませんが、デフォルトとして java.util.Collection とは異なるコレクションタイプを使用できます。
useList	**false**	**ブール値**	マップのリストまたは Pojo のリストへ無視する場合は、以下を行います。
enableJaxbAnnotationModule	**false**	**ブール値**	Jackson を使用する際に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson により JAXB アノテーションを使用できます。
moduleClassNames		**文字列**	FQN クラス名で String として指定されたカスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用します。複数のクラスをコンマで区切ることができます。
moduleRefs		**文字列**	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		**文字列**	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。
disableFeatures		**文字列**	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能セット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。

Name	デフォルト	Java タイプ	説明
permissions		文字列	xml/json から Java Bean へのアンマーシャリング中に使用可能な Java パッケージとクラス XStream を制御するパーミッションを追加します。パーミッションは、ここに設定するか、JVM システムプロパティーを使用してグローバルに設定する必要があります。パーミッションは、プラス記号が allow の構文で指定でき、マイナス記号は deny になります。ワイルドカードは、. をプレフィックスとして使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、specfy com.foo を使用します。com.foo.,-com.foo.bar.MySecretBean など、複数のパーミッションをコンマで区切って設定できます。以下のデフォルトパーミッションは常に、JVM システムプロパティーと org.apache.camel.xstream.permissions キーを指定して上書きされない限り、-,java.lang.,java.util. です。
allowUnmarshallType	false	ブール値	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する場合にのみ有効にしてください。
timezone		文字列	設定された場合、Jackson はマーシャリング/アンマーシャリング時にタイムゾーンを使用します。このオプションは、gson、fastjson、xstream などの他の Json DataFormat には影響を与えません。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

175.2. カスタム OBJECTMAPPER の使用

マッピング設定をより制御する必要がある場合に、JacksonDataFormat がカスタム ObjectMapper を使用するように設定できます。

レジストリーで単一の ObjectMapper を設定すると、Camel は自動的に検索され、この ObjectMapper を使用します。たとえば、Spring Boot を使用する場合、Spring Boot は Spring MVC が有効になっている場合にデフォルトの ObjectMapper を指定できます。これにより、Camel は Spring Boot Bean レジストリーに ObjectMapper クラスタイプを 1 つ持つことを検知し、それを使用することができます。このような場合は、Camel から INFO ロギングを設定する必要があります。

175.3. 依存関係

camel ルートで Jackson を使用するには、このデータ形式を実装する camel-jackson の依存関係を

追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jackson</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第176章 JSON JOHNZON DATAFORMAT

Camel バージョン 2.18 から利用可能

Johnzon は、Johnzon ライブラリーを使用するデータ形式です。

```
from("activemq:My.Queue").
  marshal().json(JsonLibrary.Johnzon).
  to("mqseries:Another.Queue");
```

176.1. JOHNZON オプション

JSon Johnzon データフォーマットは、以下に示す 19 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
objectMapper		文字列	Jackson を使用する際に、指定の ID で既存の ObjectMapper を検索し、使用します。
useDefaultObjectMapper	true	ブール値	レジストリーからデフォルトの Jackson ObjectMapper を検索し、使用するかどうか。
prettyPrint	false	ブール値	プリションでフォーマットされた出力を有効にします。デフォルトは false です。
library	XStream	JsonLibrary	使用する json ライブラリー。
unmarshalTypeName		文字列	無効化解除時に使用する java タイプのクラス名
jsonView		class<?>	POJO を JSON にマーシャリングする場合は、JSON 出力から特定のフィールドを除外したい場合があります。Jackson を使用すると、JSON ビューを使用してこれを実行できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		文字列	pojo を JSON にマーシャリングし、pojo には null 値を持つフィールドがあります。また、これらの null 値を省略する場合は、このオプションを NOT_NULL に設定します。

Name	デフォルト	Java タイプ	説明
allowJmsType	**false**	**ブール値**	JMS 仕様からの JMSType ヘッダーがアンマーシャリングに使用する FQN クラス名を指定するために JMS ユーザーに使用されます。
collectionTypeName		**文字列**	使用するレジストリーで検索するカスタムコレクションタイプを参照します。このオプションはほとんど使用されることはありませんが、デフォルトとして java.util.Collection とは異なるコレクションタイプを使用できます。
useList	**false**	**ブール値**	マップのリストまたは Pojo のリストへ無視する場合は、以下を行います。
enableJaxbAnnotationModule	**false**	**ブール値**	Jackson を使用する際に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson により JAXB アノテーションを使用できます。
moduleClassNames		**文字列**	FQN クラス名で String として指定されたカスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用します。複数のクラスをコンマで区切ることができます。
moduleRefs		**文字列**	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		**文字列**	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。
disableFeatures		**文字列**	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能セット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。

Name	デフォルト	Java タイプ	説明
permissions		文字列	xml/json から Java Bean へのアンマーシャリング中に使用可能な Java パッケージとクラス XStream を制御するパーミッションを追加します。パーミッションは、ここに設定するか、JVM システムプロパティーを使用してグローバルに設定する必要があります。パーミッションは、プラス記号が allow の構文で指定でき、マイナス記号は deny になります。ワイルドカードは、. をプレフィックスとして使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、specfy com.foo を使用します。com.foo.,-com.foo.bar.MySecretBean など、複数のパーミッションをコンマで区切って設定できます。以下のデフォルトパーミッションは常に、JVM システムプロパティーと org.apache.camel.xstream.permissions キーを指定して上書きされない限り、-,java.lang.,java.util. です。
allowUnmarshallType	false	ブール値	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する場合にのみ有効にしてください。
timezone		文字列	設定された場合、Jackson はマーシャリング/アンマーシャリング時にタイムゾーンを使用します。このオプションは、gson、fastjson、xstream などの他の Json DataFormat には影響を与えません。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

176.2. 依存関係

camel ルートで Johnzon を使用するには、このデータ形式を実装する camel-johnzon の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-johnzon</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第177章 JSON スキーマバリデーターコンポーネント

Camel バージョン 2.20 で利用可能

JSON Schema Validator コンポーネントは、NetworkNT JSON Schema library(https://github.com/networknt/json-schema-validator)を使用して JSON Schemas v4 ドラフトに対してメッセージボディーの Bean 検証を実行します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-json-validator</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

177.1. URI 形式

```
json-validator:resourceUri[?options]
```

resourceUri は、クラスパス上のローカルリソースへの URL、または検証する JSON スキーマを含むファイルシステム上のリモートリソースまたはリソースの完全な URL です。

177.2. URI オプション

JSON Schema Validator コンポーネントにはオプションがありません。

JSON スキーマバリデーターエンドポイントは URI 構文を使用して設定されます。

```
json-validator:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

177.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceUri	リソースへの **必須** パス。プレフィックス：classpath、file、http、ref、または bean. classpath、file、http は、これらのプロトコルを使用してリソースをロードします（classpath はデフォルト）。ref はレジストリーのリソースを検索します。Bean はリソースとして使用される Bean でメソッドを呼び出します。Bean には、ドットの後にメソッド名を指定できます（例：bean:myBean.myMethod）。		文字列

177.2.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
failOnNullBody (producer)	ボディーが存在しない場合に失敗するかどうか。	true	boolean
failOnNullHeader (producer)	ヘッダーに対して検証時にヘッダーが存在しない場合に失敗するかどうか。	true	boolean
headerName (producer)	メッセージボディーではなくヘッダーに対して検証するには、以下を行います。		文字列
errorHandler (advanced)	カスタム ValidatorErrorHandler を使用します。デフォルトのエラーハンドラーはエラーをキャプチャーし、例外をスローします。		JsonValidatorError Handler
schemaLoader (advanced)	カスタム形式検証を追加できるようにするカスタムスキーマローダーを使用します。デフォルトの実装では、ドラフト v4 サポートのあるスキーマローダーが作成されます。		JsonSchemaLoader
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

177.3. 例

以下の JSON スキーマがあるとします。

myschema.json

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "definitions": {},
  "id": "my-schema",
  "properties": {
    "id": {
      "default": 1,
      "description": "An explanation about the purpose of this instance.",
      "id": "/properties/id",
      "title": "The id schema",
      "type": "integer"
    },
    "name": {
      "default": "A green door",
      "description": "An explanation about the purpose of this instance.",
      "id": "/properties/name",
      "title": "The name schema",
      "type": "string"
    },
    "price": {
      "default": 12.5,
      "description": "An explanation about the purpose of this instance.",
      "id": "/properties/price",
      "title": "The price schema",
      "type": "number"
    }
  },
  "required": [
    "name",
    "id",
    "price"
  ],
  "type": "object"
}
```

以下の Camel ルートを使用して受信 JSON を検証できます。myschema.json はクラスパスから読み込まれます。

```
from("direct:start")
  .to("json-validator:myschema.json")
  .to("mock:end")
```

第178章 JSON XSTREAM DATAFORMAT

Camel バージョン 2.0 で利用可能

XStream は、XStream ライブラリー を使用して Java オブジェクトを XML にマーシャリングおよびアンマーシャリングするデータ形式です。

camel ルートで XStream を使用するには、このデータ形式を実装する camel-xstream の依存関係を追加する必要があります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xstream</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

178.1. オプション

JSon XStream データフォーマットは、以下に示す 19 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
objectMapper		文字列	Jackson を使用する際に、指定の ID で既存の ObjectMapper を検索し、使用します。
useDefaultObjectMapper	true	ブール値	レジストリーからデフォルトの Jackson ObjectMapper を検索し、使用するかどうか。
prettyPrint	false	ブール値	プリションでフォーマットされた出力を有効にします。デフォルトは false です。
library	XStream	JsonLibrary	使用する json ライブラリー。
unmarshalTypeName		文字列	無効化解除時に使用する java タイプのクラス名

Name	デフォルト	Java タイプ	説明
jsonView		class<?>	POJO を JSON にマーシャリングする場合は、JSON 出力から特定のフィールドを除外したい場合があります。Jackson を使用すると、JSON ビューを使用してこれを実行できます。このオプションは、JsonView アノテーションを持つクラスを参照します。
include		文字列	pojo を JSON にマーシャリングし、pojo には null 値を持つフィールドがあります。また、これらの null 値を省略する場合は、このオプションを NOT_NULL に設定します。
allowJmsType	false	ブール値	JMS 仕様からの JMSType ヘッダーがアンマーシャリングに使用する FQN クラス名を指定するために JMS ユーザーに使用されます。
collectionTypeName		文字列	使用するレジストリーで検索するカスタムコレクションタイプを参照します。このオプションはほとんど使用されることはありませんが、デフォルトとして java.util.Collection とは異なるコレクションタイプを使用できます。
useList	false	ブール値	マップのリストまたは Pojo のリストへ無視する場合は、以下を行います。
enableJaxbAnnotationModule	false	ブール値	Jackson を使用する際に JAXB アノテーションモジュールを有効にするかどうか。有効にすると、Jackson により JAXB アノテーションを使用できます。
moduleClassNames		文字列	FQN クラス名で String として指定されたカスタム Jackson モジュール com.fasterxml.jackson.databind.Module を使用します。複数のクラスをコンマで区切ることができます。
moduleRefs		文字列	Camel レジストリーから参照されるカスタム Jackson モジュールを使用します。複数のモジュールはコンマで区切ることができます。
enableFeatures		文字列	Jackson com.fasterxml.jackson.databind.ObjectMapper で有効にする機能のセット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。
disableFeatures		文字列	Jackson com.fasterxml.jackson.databind.ObjectMapper で無効にする機能セット。この機能は、com.fasterxml.jackson.databind.SerializationFeature、com.fasterxml.jackson.databind.DeserializationFeature、com.fasterxml.jackson.databind.MapperFeature 複数機能の列挙に一致する名前はコンマで区切ることができます。

Name	デフォルト	Java タイプ	説明
permissions		文字列	xml/json から Java Bean へのアンマーシャリング中に使用可能な Java パッケージとクラス XStream を制御するパーミッションを追加します。パーミッションは、ここに設定するか、JVM システムプロパティーを使用してグローバルに設定する必要があります。パーミッションは、プラス記号が allow の構文で指定でき、マイナス記号は deny になります。ワイルドカードは、. をプレフィックスとして使用することでサポートされます。たとえば、com.foo およびすべてのサブパッケージを許可するには、specfy com.foo を使用します。com.foo.,-com.foo.bar.MySecretBean など、複数のパーミッションをコンマで区切って設定できます。以下のデフォルトパーミッションは常に、JVM システムプロパティーと org.apache.camel.xstream.permissions キーを指定して上書きされない限り、-,java.lang.,java.util. です。
allowUnmarshallType	false	ブール値	有効にすると、Jackson はアンマーシャリング中に CamelJacksonUnmarshalType ヘッダーの使用を試みることができます。これは、使用する場合にのみ有効にしてください。
timezone		文字列	設定された場合、Jackson はマーシャリング/アンマーシャリング時にタイムゾーンを使用します。このオプションは、gson、fastjson、xstream などの他の Json DataFormat には影響を与えません。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

178.2. JAVA DSL の使用

```
// lets turn Object messages into XML then send to MQSeries
from("activemq:My.Queue").
  marshal().xstream().
  to("mqseries:Another.Queue");
```

メッセージ変換に Camel によって使用される XStream インスタンスを設定する場合は、DSL レベルでそのインスタンスへの参照を渡すことができます。

```
XStream xStream = new XStream();
xStream.aliasField("money", PurchaseOrder.class, "cash");
// new Added setModel option since Camel 2.14
xStream.setModel("NO_REFERENCES");
...

from("direct:marshal").
  marshal(new XStreamDataFormat(xStream)).
  to("mock:marshaled");
```

178.3. XMLINPUTFACTORY および XMLOUTPUTFACTORY

XStream ライブラリーは javax.xml.stream.XMLInputFactory および javax.xml.stream.XMLOutputFactory を使用します。このファクトリーの実装はどの実装を使用するかを制御できます。

Factory は、1 アルゴリズムを使用して検出されます。javax.xml.stream.XMLInputFactory、javax.xml.stream.XMLOutputFactory システムプロパティーを使用します。2.JRE_HOME ディレクトリーの lib/xml.stream.properties ファイルを使用します。3.利用できる場合は Services API を使用して、JRE で利用可能な jar の META-INF/services/javax.xml.stream.XMLInputFactory ファイル、META-INF/services/javax.xml.stream.XMLOutputFactory ファイルを検索してクラス名を決定します。4.プラットフォームのデフォルト XMLInputFactory,XMLOutputFactory インスタンスを使用します。

178.4. XSTREAM DATAFORMAT で XML エンコーディングを設定する方法

Camel 2.0 から、Exchange のプロパティーを Exchange.CHARSET_NAME キーで設定したり、DSL または Spring 設定から Xstream の encoding プロパティーを設定したりすることで、Xstream DataFormat の XML のエンコーディングを設定できます。

```
from("activemq:My.Queue").
  marshal().xstream("UTF-8").
  to("mqseries:Another.Queue");
```

178.5. XSTREAM DATAFORMAT のタイプパーミッションの設定

Camel では、ルートで独自の処理ステップを常に使用して、特定の XML ドキュメントをフィルタリングしてブロックし、XStream のアンマーシャリング手順にルーティングすることができます。Camel 2.16.1 から 2.15.5 より、 XStream のタイプパーミッション を設定して、特定タイプのインスタンス化を自動的に許可または拒否できます。

Camel によって使用されるデフォルトのタイプパーミッション設定は、java.lang パッケージおよび java.util パッケージ以外のすべてのタイプを拒否します。この設定は、システムプロパティー org.apache.camel.xstream.permissions を設定することで変更できます。この値は、それぞれが " の

プレフィックスが "（省略可能）か「-」の接頭辞があるかどうかに応じて、許可または拒否される型を表すカンマ区切りのパーミッション用語の文字列です。

各用語にはワイルドカード文字 '' を含めることができます。たとえば、値 "-,java.lang.,java.util." は、java.lang. クラスおよび java.util.* クラス以外のすべてのタイプを拒否します。この値を空の文字列 "" に設定すると、特定のブラックリスト化されたクラスを拒否し、その他のクラスを許可するデフォルトの XStream のタイプパーミッション処理に戻ります。*

タイプパーミッション設定は、タイプパーミッションプロパティーを設定して、個別の XStream DataFormat インスタンスで拡張できます。

```
<dataFormats>
  <xstream id="xstream-default"
           permissions="org.apache.camel.samples.xstream.*"/>
  ...
```

第179章 JSONPATH LANGUAGE

Camel バージョン 2.13 から利用可能

Camel は JSonPath をサポートし、json メッセージで Expression または Predicate を使用できます。

```
from("queue:books.new")
  .choice()
    .when().jsonpath("$.store.book[?(@.price < 10)]")
      .to("jms:queue:book.cheap")
    .when().jsonpath("$.store.book[?(@.price < 30)]")
      .to("jms:queue:book.average")
    .otherwise()
      .to("jms:queue:book.expensive")
```

179.1. JSONPATH オプション

JsonPath 言語は、以下に示す 7 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
resultType		**文字列**	結果タイプのクラス名を設定します（出力のタイプ）。
suppressExceptions	**false**	**ブール値**	PathNotFoundException などの例外を非表示にするかどうか。
allowSimple	**true**	**ブール値**	JsonPath 式でインライン化された簡単な例外を許可するかどうか。
allowEasyPredicate	**true**	**ブール値**	単純な述語パーサーを使用して事前に解析可能な述語を使用することを許可するかどうか。
writeAsString	**false**	**ブール値**	各行/要素の出力を、Map/ POJO の値ではなく JSON String の値として記述するかどうか。
headerName		**文字列**	メッセージボディーの代わりに入力として使用するヘッダーの名前。
trim	**true**	**ブール値**	値をトリミングして先頭および末尾の空白と改行を削除するかどうか。

179.2. XML 設定の使用

Spring XML ファイルでルートを設定する場合は、以下のように JSonPath 式を使用できます。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <choice>
      <when>
        <jsonpath>$.store.book[?(@.price &lt; 10)]</jsonpath>
        <to uri="mock:cheap"/>
      </when>
      <when>
        <jsonpath>$.store.book[?(@.price &lt; 30)]</jsonpath>
        <to uri="mock:average"/>
      </when>
      <otherwise>
        <to uri="mock:expensive"/>
      </otherwise>
    </choice>
  </route>
</camelContext>
```

179.3. 構文

その他の例については、JSonPath プロジェクトページを参照してください。

179.4. 簡略化構文

Camel 2.19 から利用可能

jsonpath 構文を使用して基本的な述語を定義するだけであれば、構文を覚えておくのが少し難しくなります。したがって、たとえば、必要なチュアットブックをすべて見つけるには、

```
$.store.book[?(@.price < 20)]
```

しかし、以下のように書くことができるもの

```
store.book.price < 20
```

また、価格キーを持つノードを確認する場合は、パスを省略できます。

```
price < 20
```

これをサポートするには、basic スタイルで述語を定義している場合は起動する EasyPredicateParser があります。つまり、述語は $ 記号で開始して、演算子を 1 つだけ含めることはできません。

構文は以下のとおりです。

```
left OP right
```

right Operator で Camel Simple 言語を使用できます（例：）。

```
store.book.price < ${header.limit}
```

179.5. サポートされるメッセージボディーのタイプ

Camel JSonPath は、以下のタイプのメッセージボディーをサポートしています。

Type	Comment
ファイル	ファイルからの読み込み
String	プレーンテキスト
マップ	メッセージ本文（**java.util.Map** タイプ）
リスト	**java.util.List** 型としてのメッセージ本文
POJO	**任意**。Jackson がクラスパス上にある場合、camel-jsonpath は Jackson を使用してメッセージボディーを POJO として読み取り、JSonPath でサポートされる **java.util.Map** に変換できます。たとえば、依存関係として **camel-jackson** を追加して、Jackson を含めることができます。
InputStream	上記のタイプがどれも一致しない場合、Camel はメッセージボディーを **java.io.InputStream** 型で読み込みます。

メッセージボディーがサポートされないタイプの場合は、デフォルトで例外が発生しますが、JSonPath を設定して例外を抑制することができます（以下を参照してください）。

179.6. 例外の抑制

Camel 2.16 から利用可能

json ペイロードに設定済みの jsonpath 式に応じて有効なパスがない場合は、デフォルトでは jsonpath が例外をスローします。一部のユースケースでは、json ペイロードにオプションのデータが含まれる場合にこれを無視することができます。そのため、以下のように suppressExceptions オプションを true に設定すると、これを無視できます。

```
from("direct:start")
   .choice()
      // use true to suppress exceptions
      .when().jsonpath("person.middlename", true)
         .to("mock:middle")
      .otherwise()
         .to("mock:other");
```

XML DSL の場合：

```
<route>
  <from uri="direct:start"/>
  <choice>
    <when>
      <jsonpath suppressExceptions="true">person.middlename</jsonpath>
      <to uri="mock:middle"/>
    </when>
    <otherwise>
      <to uri="mock:other"/>
    </otherwise>
  </choice>
</route>
```

このオプションは @JsonPath アノテーションでも利用できます。

179.7. インライン SIMPLE 例外

Camel 2.18 から利用可能

Simple 構文 ${xxx} を使用して、JSonPath 式でインライン化された Simple 言語式を使用できるようになりました。以下に例を示します。

```
from("direct:start")
  .choice()
    .when().jsonpath("$.store.book[?(@.price < ${header.cheap})]")
      .to("mock:cheap")
```

```
    .when().jsonpath("$.store.book[?(@.price < ${header.average})]")
      .to("mock:average")
    .otherwise()
      .to("mock:expensive");
```

XML DSL の場合：

```
<route>
  <from uri="direct:start"/>
  <choice>
    <when>
      <jsonpath>$.store.book[?(@.price < ${header.cheap})]</jsonpath>
      <to uri="mock:cheap"/>
    </when>
    <when>
      <jsonpath>$.store.book[?(@.price < ${header.average})]</jsonpath>
      <to uri="mock:average"/>
    </when>
    <otherwise>
      <to uri="mock:expensive"/>
    </otherwise>
  </choice>
</route>
```

以下のように allowSimple オプションを false に設定すると、インラインの Simple 式のサポートをオフにできます。

```
.when().jsonpath("$.store.book[?(@.price < 10)]", false, false)
```

XML DSL の場合：

```
<jsonpath allowSimple="false">$.store.book[?(@.price < 10)]</jsonpath>
```

179.8. JSONPATH の注入

Bean インテグレーションを使用して Bean でメソッドを呼び出し、JSonPath などのさまざまな言語を使用してメッセージから値を抽出し、メソッドパラメーターにバインドすることができます。

たとえば、以下のようになります。

```
public class Foo {

    @Consume(uri = "activemq:queue:books.new")
    public void doSomething(@JsonPath("$.store.book[*].author") String author, @Body String
```

```
json) {
      // process the inbound message here
    }
}
```

179.9. エンコーディングの検出

Camel バージョン 2.16 以降、ドキュメントが RFC-4627 で指定されているように unicode（UTF-8、UTF-16LE、UTF-16BE、UTF-32LE、UTF-32BE）でエンコードされている場合、JSON ドキュメントのエンコーディングは自動的に検出されます。エンコーディングがunicode 以外のエンコーディングである場合は、ドキュメントを String 形式で JSONPath コンポーネントに入力するか、ヘッダー「CamelJsonPathJsonEncoding"(JsonpathConstants.HEADER_JSON_ENCODING)にエンコーディングを指定することができます。

179.10. JSON データを JSON としてサブ行に分割

jsonpath を使用して、以下のような JSon ドキュメントを分割できます。

```
from("direct:start")
    .split().jsonpath("$.store.book[*]")
    .to("log:book");
```

各ブックはログに記録されますが、メッセージボディーは Map インスタンスです。これを代わりに単純な String JSon 値として出力したい場合があります。これは、以下 の ように Camel 2.20 から実行できます。

```
from("direct:start")
    .split().jsonpathWriteAsString("$.store.book[*]")
    .to("log:book");
```

次に、各ブックは String JSon の値としてログに記録されます。以前のバージョンの Camel では、camel-jackson データフォーマットを使用して、メッセージボディーを Map から String 型に変換する必要があります。

179.11. ヘッダーの入力としての使用

Camel 2.20 から利用可能

デフォルトでは、jsonpath はメッセージボディーを入力ソースとして使用します。ただし、headerName オプションを指定することで、ヘッダーを入力として使用することもできます。

たとえば、books という名前のヘッダーに保存されている json ドキュメントから本書の数をカウントするには、以下を行うことができます。

```
from("direct:start")
    .setHeader("numberOfBooks")
        .jsonpath("$..store.book.length()", false, int.class, "books")
    .to("mock:result");
```

上記の jsonpath 式ではヘッダー名を 書籍 として指定し、結果を int.class で整数に変換するように指示します。

XML DSL を使用した場合の同じ例は次のとおりです。

```
<route>
  <from uri="direct:start"/>
  <setHeader headerName="numberOfBooks">
    <jsonpath headerName="books" resultType="int">$..store.book.length()</jsonpath>
  </transform>
  <to uri="mock:result"/>
</route>
```

179.12. 依存関係

camel ルートで JSonPath を使用するには、JSonPath 言語を実装する camel-jsonpath の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jsonpath</artifactId>
  <version>x.x.x</version>
</dependency>
```

第180章 JT400 COMPONENT

Camel バージョン 1.5 で利用可能

jt400 コンポーネントを使用すると、データキューを使用して AS/400 システムのメッセージを交換できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jt400</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

180.1. URI 形式

```
jt400://user:password@system/QSYS.LIB/LIBRARY.LIB/QUEUE.DTAQ[?options]
```

リモートプログラム(Camel 2.7)を呼び出す

```
jt400://user:password@system/QSYS.LIB/LIBRARY.LIB/program.PGM[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

180.2. JT400 オプション

JT400 コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
connectionPool (advanced)	このコンポーネントによって使用されるデフォルトの接続プールを返します。		AS400ConnectionPool

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

JT400 エンドポイントは、URI 構文を使用して設定します。

```
jt400:userID:password/systemName/objectPath.type
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

180.2.1. パスパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
userID	**必須** は AS/400 ユーザーの ID を返します。		文字列
password	**必須** は AS/400 ユーザーのパスワードを返します。		文字列
systemName	**必須** は AS/400 システムの名前を返します。		文字列
objectPath	**必須** は、このエンドポイントのターゲットオブジェクトの完全修飾統合ファイルシステムのパス名を返します。		文字列
type	データキューまたはリモートプログラム呼び出しを使用するかどうかが **必要**		Jt400Type

180.2.2. クエリーパラメーター（30 パラメーター）：

Name	説明	デフォルト	Type
ccsid (common)	AS/400 システムとの接続に使用する CCSID を設定します。		int
形式 (common)	メッセージを送信するためのデータ形式を設定します。	text	Format

Name	説明	デフォルト	Type
guiAvailable (common)	Camel を実行している環境で AS/400 要求が有効であるかどうかを設定します。	false	boolean
keyed (common)	キーが付けられたデータキューを使用するかどうか。	false	boolean
outputFieldsIdxArray (common)	出力パラメーターであるフィールド（program パラメーター）を指定します。		integer[]
outputFieldsLengthArray (common)	AS/400 プログラム定義でフィールド（program パラメーター）の長さを指定します。		integer[]
searchKey (common)	キーで主要なデータキューを検索します。		文字列
searchType (common)	EQ などの検索タイプ（等しい場合など）。	EQ	SearchType
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
readTimeout (consumer)	タイムアウト（ミリ秒単位）は、データキューの新しいメッセージを読み取ろうとしている間にコンシューマーが待機します。	30000	int
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service

Name	説明	デフォルト	Type
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
procedureName (procedureName)	呼び出すサービスプログラムからの手順名		文字列
secured （セキュリティー）	AS/400 への接続が SSL でセキュア化されるかどうか。	false	boolean

180.3. 用途

コンシューマーエンドポイントとして設定すると、エンドポイントはリモートシステムのデータキューをポーリングします。データキューのすべてのエントリーに対して、新しいエクスチェンジが **In** *メッセージのボディー内のエントリーのデータで送信され、フォーマットに応じて* **String** *または* **byte[]** *としてフォーマットされます。プロバイダーエンドポイントの場合、* **In** *メッセージボディーの内容は生のバイトまたはテキストとしてデータキューに配置されます。*

180.4. 接続プール

***Camel 2.10** で利用可能*

接続プールは **Camel 2.10** *以降から使用されています。* **Jt400Component** *で接続プールを明示的に設定したり、エンドポイントの* **uri** *オプションとして設定したりできます。*

180.4.1. リモートプログラムコール(Camel 2.7)

このエンドポイントは、入力が String 配列または byte[] アレイのいずれかであることを想定し、ネイティブの jt400 ライブラリーメカニズムを介してすべての CCSID 処理を処理します。位置の値として null を渡すとパラメーターを 省略 できます（リモートプログラムはサポートする必要があります）。プログラムを実行すると、エンドポイントは String 配列または byte[] 配列のいずれかをプログラムによって返された値を返します（入力のみのパラメーターには呼び出しの開始と同じデータが含まれます）。このエンドポイントはプロバイダーエンドポイントを実装しません。

180.5. 例

以下のスニペットでは、direct:george エンドポイントに送信されたエクスチェンジのデータは、LIVERPOOL という名前のシステムのライブラリー BEATLES のデータキュー PENNYLANE に配置されます。
別のユーザーは同じデータキューに接続してデータキューから情報を受け取り、mock:ringo エンドポイントに転送します。

```
public class Jt400RouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {

from("direct:george").to("jt400://GEORGE:EGROEG@LIVERPOOL/QSYS.LIB/BEATLES.LIB/PENNYLANE.DTAQ");

from("jt400://RINGO:OGNIR@LIVERPOOL/QSYS.LIB/BEATLES.LIB/PENNYLANE.DTAQ").to("mock:ringo");
    }
}
```

180.5.1. リモートプログラムコールの例(Camel 2.7)

以下のスニペットでは direct:work エンドポイントに送信されたデータ Exchange には、ライブラリー「assets」のプログラム「compute」の引数として使用される 3 つの文字列が含まれます。このプログラムは、2 番目と 3 番目のパラメーターに出力値を書き込みます。すべてのパラメーターは direct:play エンドポイントに送信されます。

```
public class Jt400RouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {

from("direct:work").to("jt400://GRUPO:ATWORK@server/QSYS.LIB/assets.LIB/compute.PGM?fieldsLength=10,10,512&ouputFieldsIdx=2,3").to(“direct:play”);
    }
}
```

180.5.2. キーデータキューへの書き込み

```
from("jms:queue:input")
.to("jt400://username:password@system/lib.lib/MSGINDQ.DTAQ?keyed=true");
```

-

180.5.3. キーされたデータキューからの読み取り

```
from("jt400://username:password@system/lib.lib/MSGOUTDQ.DTAQ?
keyed=true&searchKey=MYKEY&searchType=GE")
.to("jms:queue:output");
```

180.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第181章 KAFKA コンポーネント

Camel バージョン 2.13 から利用可能

kafka: コンポーネントは Apache Kafka メッセージブローカーとの通信に使用されます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-kafka</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

181.1. URI 形式

```
kafka:topic[?options]
```

181.2. オプション

Kafka コンポーネントは、以下に示す 8 個のオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	エンドポイントが再利用する共通のオプションで Kafka コンポーネントを事前設定できます。		KafkaConfiguration
ブローカー （共通）	使用する Kafka ブローカーの URL。形式は host1:port1,host2:port2 であり、一覧はブローカーのサブセットを参照するブローカーまたは VIP のサブセットになります。このオプションは、Kafka ドキュメントの bootstrap.servers と呼ばれます。		文字列
workerPool (advanced)	共有カスタムワーカープールを使用して、Kafka サーバーが非同期非ブロッキング処理を使用して KafkaProducer から送信されたメッセージを確認した後もルーティングエクスチェンジを続行します。このオプションを使用する場合、必要なくなった場合にプールをシャットダウンするためにスレッドプールのライフサイクルを処理する必要があります。		ExecutorService

Name	説明	デフォルト	Type
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
breakOnFirstError (consumer)	このオプションは、コンシューマーがエクスチェンジを処理し、失敗したときに発生する内容を制御します。オプションが false の場合、コンシューマーは次のメッセージを継続し、処理します。オプションが true の場合、コンシューマーは破損し、失敗の原因となったメッセージのオフセットにシークし、再度このメッセージを処理します。ただし、これにより、同じメッセージが毎回失敗すると、同じメッセージが無限に処理される可能性があります（例：poison メッセージ）。そのため、Camel のエラーハンドラーなどを使用してこれに対応することが推奨されます。	false	boolean
allowManualCommit (consumer)	KafkaManualCommit を使用して手動のコミットを許可するかどうか。このオプションを有効にすると、KafkaManualCommit のインスタンスは Exchange メッセージヘッダーに保存されます。これにより、エンドユーザーはこの API にアクセスし、Kafka コンシューマー経由で手動のオフセットコミットを実行できます。	false	boolean
kafkaManualCommit Factory (consumer)	KafkaManualCommit インスタンスの作成に使用するファクトリー。これにより、カスタムファクトリーをプラグインしてカスタム KafkaManualCommit インスタンスを作成できます。		KafkaManualCommit Factory
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Kafka エンドポイントは、URI 構文を使用して設定します。

```
kafka:topic
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

181.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
topic	使用するトピックの名前。**コンシューマーでは、コンマを使用して複数のトピックを分けることができます。プロデューサーはメッセージを単一のトピックにのみ送信できます。**		文字列

181.2.2. クエリーパラメーター（93 パラメーター）：

Name	説明	デフォルト	Type
ブローカー（共通）	使用する Kafka ブローカーの URL。形式は host1:port1,host2:port2 であり、一覧はブローカーのサブセットを参照するブローカーまたは VIP のサブセットになります。このオプションは、Kafka ドキュメントの bootstrap.servers と呼ばれます。		文字列
clientId (common)	クライアント ID は、呼び出しを追跡できるように各リクエストで送信されるユーザー指定の文字列です。要求を行うアプリケーションを論理的に識別する必要があります。		文字列
headerFilterStrategy (common)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
reconnectBackoffMaxMs (common)	接続に繰り返し失敗したブローカーへの再接続時に待機する最大時間（ミリ秒単位）。これが指定されている場合、ホストごとのバックオフは、連続して接続に失敗するたびに、この最大値まで指数関数的に増加します。バックオフの増加を計算した後、コネクションストームを回避するために 20% のランダムなジッターが追加されます。	1000	整数
allowManualCommit (consumer)	KafkaManualCommit を使用して手動のコミットを許可するかどうか。このオプションを有効にすると、KafkaManualCommit のインスタンスは Exchange メッセージヘッダーに保存されます。これにより、エンドユーザーはこの API にアクセスし、Kafka コンシューマー経由で手動のオフセットコミットを実行できます。	false	boolean
autoCommitEnable (consumer)	true の場合、コンシューマーによってすでにフェッチされたメッセージのオフセットを ZooKeeper に定期的にコミットします。このコミットされたオフセットは、新規コンシューマーの開始位置としてプロセスが失敗した場合に使用されます。	true	ブール値

Name	説明	デフォルト	Type
autoCommitIntervalMs (consumer)	コンシューマーオフセットが zookeeper にコミットされる頻度（ミリ秒単位）。	5000	整数
autoCommitOnStop (consumer)	ブローカーが最後に消費されたメッセージからのコミットを確実にするために、コンシューマーが停止したときに明示的な自動コミットを実行するかどうか。これには、autoCommitEnable オプションが有効になっている必要があります。使用できる値は sync、async、または none です。および sync はデフォルト値です。	sync	文字列
autoOffsetReset (consumer)	ZooKeeper に初期オフセットがない場合や、オフセットが範囲外の場合： 最小 : オフセットを最小のオフセットに自動リセットします。 : 自動的にオフセットを最大のオフセットにリセットします： コンシューマーに例外をスローします。	latest	文字列
breakOnFirstError (consumer)	このオプションは、コンシューマーがエクスチェンジを処理し、失敗したときに発生する内容を制御します。オプションが false の場合、コンシューマーは次のメッセージを継続し、処理します。オプションが true の場合、コンシューマーは破損し、失敗の原因となったメッセージのオフセットにシークし、再度このメッセージを処理します。ただし、これにより、同じメッセージが毎回失敗すると、同じメッセージが無限に処理される可能性があります（例：poison メッセージ）。そのため、Camel のエラーハンドラーなどを使用してこれに対応することが推奨されます。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
checkCrcs (consumer)	消費されたレコードの CRC32 を自動的に確認します。これにより、メッセージのネットワーク上またはディスク上の破損が発生しなくなります。このチェックはオーバーヘッドを追加するため、極端なパフォーマンスを求める場合は無効になる可能性があります。	true	ブール値

Name	説明	デフォルト	Type
consumerRequestTimeoutMs (consumer)	この設定は、クライアントの要求の応答を待つ最大時間を制御します。タイムアウトが経過する前に応答が受信されない場合、クライアントは必要に応じてリクエストを再送信します。または、再試行が使い切られるとリクエストが失敗します。	40000	整数
consumersCount (consumer)	kafka サーバーに接続するコンシューマーの数	1	int
consumerStreams (consumer)	コンシューマーの同時コンシューマーの数	10	int
fetchMaxBytes (consumer)	サーバーがフェッチリクエストに対して返す必要のあるデータの最大量。これは、フェッチの最初の空でないパーティションの最初のメッセージがこの値よりも大きい場合に、メッセージを返してコンシューマーが進行できることを確認します。ブローカーによって許可される最大メッセージサイズは、message.max.bytes（ブローカー設定）または max.message.bytes（トピック設定）で定義されます。コンシューマーは複数のフェッチを並行して実行することに注意してください。	52428800	整数
fetchMinBytes (consumer)	サーバーがフェッチ要求に対して返す必要のあるデータの最小量。利用可能なデータが不十分な場合、リクエストは、リクエストに応答する前に、十分なデータが蓄積されるのを待ちます。	1	整数
fetchWaitMaxMs (consumer)	すぐに fetch.min.bytes を満たすのに十分なデータがない場合にサーバーがフェッチリクエストに応答するまでにブロックする最大時間。	500	整数
groupId (consumer)	このコンシューマーが属するコンシューマープロセスのグループを一意に識別する文字列。同じグループ ID を複数設定するプロセスは、すべて同じコンシューマーグループの一部であることを意味します。このオプションはコンシューマーに必要です。		文字列
heartbeatIntervalMs (consumer)	Kafka のグループ管理機能を使用する場合の、ハートビートからコンシューマーコーディネーター間の想定される時間。ハートビートは、コンシューマーのセッションがアクティブな状態を維持し、新しいコンシューマーがグループに参加したり離脱したりする際のリバランスを促進するために使用されます。値は session.timeout.ms よりも低く設定する必要がありますが、通常はその値の 1/3 以下に設定する必要があります。さらに低く調整することで、通常のリバランスの予想時間を制御することもできます。	3000	整数

Name	説明	デフォルト	Type
kafkaHeaderDeserializer (consumer)	kafka ヘッダーの値を camel ヘッダー値にデシリアライズするカスタム KafkaHeaderDeserializer を設定します。		KafkaHeaderDeserializer
keyDeserializer (consumer)	Deserializer インターフェースを実装するキーのデシリアライザークラス。	org.apache.kafka.common.serialization.StringDeserializer	文字列
maxPartitionFetchBytes (consumer)	サーバーが返すパーティションごとのデータの最大量。リクエストに使用される最大メモリーの合計は、パーティション max.partition.fetch.bytes になります。このサイズは、サーバーが許可するメッセージの最大サイズが大きいか、コンシューマーよりも大きなメッセージを送信することができます。この場合、コンシューマーはあるパーティションで大きなメッセージの取得を試行しなくなります。	1048576	整数
maxPollIntervalMs (consumer)	コンシューマーグループ管理を使用する場合の poll() の呼び出し間の最大遅延。これにより、コンシューマーがさらにレコードをフェッチする前にアイドル状態になることができる時間に上限が設定されます。このタイムアウトの期限が切れる前に poll() が呼び出されない場合、コンシューマーは失敗とみなされ、グループはパーティションを別のメンバーに再割り当てするためにリバランスします。		Long
maxPollRecords (consumer)	poll（）への単一呼び出しで返される最大レコード数	500	整数
offsetRepository (consumer)	トピックの各パーティションのオフセットをローカルに保存するために使用するオフセットリポジトリー。定義すると、オートコミットが無効になります。		String>
partitionAssignor (consumer)	グループ管理が使用される場合に、クライアントがコンシューマーインスタンス間でパーティションの所有権を分散するために使用するパーティション割り当てストラテジーのクラス名。	org.apache.kafka.clients.consumer.RangeAssignor	文字列

Name	説明	デフォルト	Type
pollTimeoutMs (consumer)	KafkaConsumer のポーリング時に使用されるタイムアウト。	5000	Long
seekTo (consumer)	起動時に KafkaConsumer が最初から読み取るか、または最後に読み込んだ場合を設定します。end: read from end は、前述のプロパティー seekToBeginning を置き換えます。		文字列
sessionTimeoutMs (consumer)	Kafka のグループ管理機能の使用時に障害の検出に使用されるタイムアウト。	10000	整数
topicIsPattern (consumer)	トピックがパターン（正規表現）であるかどうか。これは、パターンに一致するトピックの動的数をサブスクライブするために使用できます。	false	boolean
valueDeserializer (consumer)	Deserializer インターフェースを実装する値のデシリアライザークラス。	org.apache.kafka.common.serialization.StringDeserializer	文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
bridgeEndpoint (producer)	オプションが true の場合、KafkaProducer はインバウンドメッセージの KafkaConstants.TOPIC ヘッダー設定を無視します。	false	boolean

Name	説明	デフォルト	Type
bufferMemorySize (producer)	プロデューサーが、サーバーへの送信を待機しているレコードをバッファリングするために使用できるメモリーの合計バイト数。レコードの送信速度がサーバーへの配信よりも高速である場合、プロデューサーは block.on.buffer.full で指定された設定に基づいて例外をブロックまたはスローします。この設定は、プロデューサーが使用するメモリーの合計にほぼ対応しますが、プロデューサーが使用するメモリーがすべてバッファーに使用されるわけではないので、ハードバインドではありません。一部の追加メモリーは、圧縮（圧縮が有効な場合）やインフライトリクエストの維持に使用されます。	33554432	整数
circularTopicDetection (producer)	オプションが true の場合、メッセージが kafka コンシューマーから元のメッセージである場合、KafkaProducer はメッセージが同じトピックに返信しようとするかどうかを検出します。KafkaConstants.TOPIC ヘッダーが元の kafka コンシューマートピックと同じ場合、ヘッダー設定は無視され、プロデューサーエンドポイントのトピックが使用されます。言い換えると、同じメッセージが発信元の場所に返信されることを防ぐことができます。このオプションは、bridgeEndpoint オプションが true に設定されている場合は使用されません。	true	boolean
compressionCodec (producer)	このパラメーターを使用すると、このプロデューサーによって生成されたすべてのデータに圧縮コーデックを指定できます。有効な値は none、gzip、および snappy です。	none	文字列
connectionMaxIdleMs (producer)	この設定で指定された期間（ミリ秒単位）の後にアイドル状態の接続を閉じます。	540000	整数
enableIdempotence (producer)	'true' に設定された場合、プロデューサーは各メッセージのコピーが 1 つだけストリームに書き込まれるようにします。「false」の場合、プロデューサーの再試行により、ストリームに再試行されたメッセージの重複が書き込まれる可能性があります。true に設定すると、max.in.flight.requests.per.connection を 1 に設定する必要があり、再試行はゼロにすることはできず、追加の認証は all に設定する必要があります。	false	boolean
kafkaHeaderSerializer (producer)	camel ヘッダー値を kafka ヘッダー値にシリアライズするカスタム KafkaHeaderDeserializer を設定します。		KafkaHeaderSerializer

Name	説明	デフォルト	Type
キー （プロデューサー）	レコードキー（またはキーが指定されていない場合は null）このオプションを設定すると、ヘッダーリンク KafkaConstantsKEY よりも優先されます。		文字列
keySerializerClass (producer)	キーのシリアライザークラス（指定がない場合はメッセージと同じものになります）。	org.apache.kafka.common.serialization.StringSerializer	文字列
lingerMs (producer)	プロデューサーは、リクエストの送信の間に到着したレコードを 1 つのバッチリクエストにグループ化します。通常、これは、レコードが送信できるよりも早く到着した場合に、負荷がかかった状態でのみ発生します。ただし、状況によっては、中程度の負荷がかかっている場合でも、クライアントがリクエストの数を減らしたい場合があります。この設定は、プロデューサーを即座に送信するのではなく、人為的な遅延を追加することで実現します。つまり、プロデューサーはレコードをすぐに送信するのではなく、他のレコードが送信されるまで指定の遅延まで待機し、送信をバッチでバッチ処理できるようにします。これは、TCP の Nagle アルゴリズムに類似するものと考えることができます。この設定は、バッチ処理の遅延の上限を与えます。パーティションのレコードの batch.size は、重要なレコードを取得すると、この設定に関係なくすぐに送信されますが、このパーティションに蓄積されたバイト数がこれより少ない場合は、指定された時間の間、より多くのレコードが表示されるのを「linger」とします。デフォルトは 0（つまり遅延なし）に設定されます。たとえば、linger.ms=5 を設定すると、送信されるリクエストの数が減りますが、負荷の異常で送信されるレコードに最大 5 ミリ秒のレイテンシーが追加されます。	0	整数
maxBlockMs (producer)	この設定は、kafka に送信する期間を制御します。これらのメソッドは、複数の理由でブロックできます。（例： buffer full、メタデータは利用できません）。この設定により、メタデータの取得に費やされた合計時間、キーと値のシリアライズ、send（）の実行時にバッファーメモリーのパーティションおよび割り当てに最大制限が適用されます。partitionsFor（）の場合、この設定によりメタデータの待機中に最大時間のしきい値が適用されます。	60000	整数

Name	説明	デフォルト	Type
maxInFlightRequest (producer)	ブロックする前にクライアントが 1 つの接続で送信する確認されていないリクエストの最大数。この設定が 1 を超える値に設定され、送信に失敗した場合は、再試行によりメッセージの順序が及ぶ可能性があります（再試行が有効な場合など）。	5	整数
maxRequestSize (producer)	リクエストの最大サイズ。また、これは最大レコードサイズの上限でもあります。サーバーにはレコードサイズに固有の上限があり、これとは異なる可能性があることに注意してください。この設定により、プロデューサーが 1 回のリクエストで送信するレコードバッチの数が制限され、大量のリクエストが送信されないようになります。	1048576	整数
metadataMaxAgeMs (producer)	新しいブローカーまたはパーティションをプロアクティブに検出するためのパーティションリーダーシップの変更がない場合でも、メタデータの更新を強制するまでの期間 (ミリ秒単位)。	300000	整数
metricReporters (producer)	メトリクスレポーターとして使用するクラスの一覧。MetricReporter インターフェースを実装すると、新規メトリクスの作成が通知されるクラスのプラグが可能になります。JmxReporter は、JMX 統計を登録するために常に含まれます。		文字列
metricsSampleWindowMs (producer)	メトリクスを計算するために保持されるサンプルの数。	30000	整数
noOfMetricsSample (producer)	メトリクスを計算するために保持されるサンプルの数。	2	整数
partitioner （プロデューサー）	サブトピック間でメッセージを分割するためのパーティションクラス。デフォルトのパーティショニャーは、キーのハッシュに基づいています。	org.apache.kafka.clients.producer.internals.DefaultPartitioner	文字列
partitionKey (producer)	レコードの送信先となるパーティション（パーティションの指定がない場合は null）。このオプションを設定すると、ヘッダーリンク KafkaConstantsPARTITION_KEY よりも優先されます。		整数

Name	説明	デフォルト	Type
producerBatchSize (producer)	複数のレコードが同じパーティションに送信されるときは常に、プロデューサーはレコードをまとめてより少ない要求にバッチ処理しようとします。これにより、クライアントとサーバーの両方でパフォーマンスが向上します。この設定では、デフォルトのバッチサイズをバイト単位で制御します。ブローカーに送信されるリクエストには複数のバッチが含まれ、送信可能なデータがあるパーティションごとにバッチ処理は試行されません。バッチサイズが小さいと、バッチ処理が少なくなることがあります（バッチサイズはゼロの場合、バッチサイズは完全に無効化されます）。バッチサイズが非常に大きい場合は、追加のレコードを想定して、常に指定のバッチサイズのバッファーを割り当てるため、メモリーを多少無駄に使用する可能性があります。	16384	整数
queueBufferingMaxMessages (producer)	プロデューサーがブロックされるか、またはデータを取り除く前に、非同期モードの使用時にプロデューサーをキューに入れることができる未送信メッセージの最大数。	10000	整数
receiveBufferBytes (producer)	データの読み取り時に使用する TCP 受信バッファー(SO_RCVBUF)のサイズ。	65536	整数
reconnectBackoffMs (producer)	指定されたホストへの再接続を試みるまで待機する時間。これにより、タイトなループでホストに繰り返し接続することを回避します。このバックオフは、コンシューマーによってブローカーに送信されるすべてのリクエストに適用されます。	50	整数
recordMetadata (producer)	プロデューサーが RecordMetadata 結果を Kafka に送信から保存すべきかどうか。結果は、recordMetadata メタデータを含む List に保存されます。このリストは、KafkaConstantsKAFKA_RECORDMETA キーのあるヘッダーに保存されます。	true	boolean

Name	説明	デフォルト	Type
requestRequiredAcks (producer)	リクエストが完了したと見なす前に、プロデューサーがリーダーに受け取ったことを要求する確認の数。これは、送信されるレコードの耐久性を制御します。次の設定は共通です： acks=0 ゼロに設定すると、プロデューサーはサーバーからの確認応答を一切待ちません。レコードは直ちにソケットバッファーに追加され、送信済みと見なされます。この場合、サーバーがレコードを受信したことを保証できず、再試行設定は有効になりません（クライアントは通常失敗を認識しないため）。各レコードに返されるオフセットは、常に -1 acks=1 に設定されます。これは、リーダーがレコードをローカルログに書き込みますが、すべてのフォロワーからの完全な確認応答を待たずに応答します。この場合、レコードの承認後すぐにリーダーが失敗するはずですが、フォロワーが複製される前にレコードが失われます。acks=all これは、リーダーが同期レプリカの完全なセットがレコードを確認するのを待ちます。これにより、少なくとも 1 つの In-Sync レプリカが動作している限り、レコードが失われないことが保証されます。これは利用可能な最強の保証になります。	1	文字列
requestTimeoutMs (producer)	エラーをクライアントに送信する前にブローカーが request.required.acks 要件を満たすまで待機する時間。	305000	整数
再試行 （プロデューサー）	ゼロより大きい値を設定すると、クライアントは、一時的なエラーの可能性により送信に失敗したレコードを再送信します。この再試行は、クライアントがエラーを受信したときにレコードを再送した場合と同じであることに注意してください。再試行を許可すると、2 つのレコードが 1 つのパーティションに送信され、最初のレコードが失敗して再試行されますが、2 番目のレコードが最初に表示される可能性があるため、レコードの順序が変わる可能性があります。	0	整数
retryBackoffMs (producer)	各再試行前に、プロデューサーは該当するトピックのメタデータを更新し、新しいリーダーが選出されたかどうかを確認します。リーダーの選択には少し時間がかかるため、このプロパティーはメタデータを更新する前にプロデューサーが待機する時間を指定します。	100	整数
sendBufferBytes (producer)	ソケット書き込みバッファーサイズ	131072	整数

Name	説明	デフォルト	Type
serializerClass (producer)	メッセージのシリアライザークラス。	org.apache.kafka.common.serialization.StringSerializer	文字列
workerPool (producer)	Kafka サーバーが非同期非ブロッキング処理を使用して KafkaProducer から送信されたメッセージを確認した後、カスタムワーカープールを使用してルーティングエクスチェンジを続行します。		ExecutorService
workerPoolCoreSize (producer)	Kafka サーバーが非同期非ブロッキング処理を使用して KafkaProducer から送信されたメッセージを確認した後、ルーティングエクスチェンジを継続するワーカープールのコアスレッドの数。	10	整数
workerPoolMaxSize (producer)	Kafka サーバーが非同期非ブロッキング処理を使用して KafkaProducer から送信されたメッセージを確認した後、ルーティングエクスチェンジを続行するためのワーカープールの最大スレッド数。	20	整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
interceptorClasses (monitoring)	プロデューサーまたはコンシューマーのインターセプターを設定します。プロデューサーインターセプターは、org.apache.kafka.clients.producer.ProducerInterceptor Consumer インターセプターを実装するクラスである必要があります。コンシューマーに Producer インターセプターを使用している場合は、実行時にクラスキャスト例外をスローすることに注意してください。		文字列
kerberosBeforeReloginMin Time (security)	更新試行間のログインスレッドのスリープ時間。	60000	整数
kerberosInitCmd (security)	Kerberos kinit コマンドパス。デフォルトは /usr/bin/kinit です。	/usr/bin/kinit	文字列

Name	説明	デフォルト	Type
kerberosPrincipalToLocal Rules (security)	プリンシパル名から短縮名（通常はオペレーティングシステムのユーザー名）にマッピングするためのルールの一覧です。ルールは順番に評価され、プリンシパル名と一致する最初のルールは、これを短縮名にマップするために使用されます。一覧の後続のルールは無視されます。デフォルトでは、username/hostnameREALM 形式のプリンシパル名は username にマッピングされます。形式の詳細は、「セキュリティー承認および acl」を参照してください。複数の値はコンマで区切ることができます。	DEFAULT	文字列
kerberosRenewJitter (security)	更新時間に追加されたランダムなジッターの割合。	0.05	double
kerberosRenewWindowFactor (security)	ログインスレッドは、最後の更新からチケットの有効期限までの指定された時間のウィンドウファクターに達するまでスリープし、その時点でチケットの更新を試みます。	0.8	double
saslJaasConfig (security)	expose the kafka sasl.jaas.config parameter Example: org.apache.kafka.common.security.plain.PlainLoginModule required username=USERNAME password=PASSWORD;		文字列
saslKerberosServiceName (security)	Kafka が実行される Kerberos プリンシパル名。これは、Kafka の JAAS 設定または Kafka の設定で定義できます。		文字列
saslMechanism (security)	使用される Simple Authentication and Security Layer(SASL)メカニズム。有効な値は http://www.iana.org/assignments/sasl-mechanisms/sasl-mechanisms.xhtmlを参照してください。	GSSAPI	文字列
securityProtocol (security)	ブローカーとの通信に使用されるプロトコル。現在、PLAINTEXT および SSL のみがサポートされます。	PLAINTEXT	文字列
sslCipherSuites (security)	暗号化スイートの一覧。これは、TLS または SSL ネットワークプロトコルを使用してネットワーク接続のセキュリティー設定をネゴシエートするために使用される認証、暗号化、MAC、および鍵交換アルゴリズムの名前付きの組み合わせです。デフォルトで利用可能なすべての暗号スイートがサポートされます。		文字列
sslContextParameters (security)	Camel SSLContextParameters オブジェクトを使用した SSL 設定。設定されている場合は、他の SSL エンドポイントパラメーターの前に適用されます。		SSLContextParameters

Name	説明	デフォルト	Type
sslEnabledProtocols (security)	SSL 接続で有効なプロトコルの一覧。TLSv1.2、TLSv1.1、および TLSv1 はデフォルトで有効になります。	TLSv1.2,TLSv1.1,TLSv1	文字列
sslEndpointAlgorithm (security)	サーバー証明書を使用してサーバーのホスト名を検証するエンドポイント識別アルゴリズム。		文字列
sslKeymanagerAlgorithm (security)	SSL 接続のキーマネージャーファクトリーによって使用されるアルゴリズム。デフォルト値は、Java 仮想マシンに設定されたキーマネージャーファクトリーアルゴリズムです。	SunX509	文字列
sslKeyPassword (security)	キーストアファイルの秘密鍵のパスワード。これはクライアントにとってオプションになります。		文字列
sslKeystoreLocation (security)	キーストアファイルの場所。これはクライアントではオプションで、クライアントの双方向認証に使用できます。		文字列
sslKeystorePassword (security)	キーストアファイルのストアパスワード。これはクライアントに対してオプションであり、ssl.keystore.location が設定されている場合のみ必要です。		文字列
sslKeystoreType (security)	キーストアファイルのファイル形式。これはクライアントにとってオプションになります。デフォルト値は JKS です。	JKS	文字列
sslProtocol (security)	SSLContext の生成に使用される SSL プロトコル。デフォルトの設定は TLS で、ほとんどの場合で問題ありません。最新の JVM で許可される値は TLS、TLSv1.1、および TLSv1.2 です。SSL、SSLv2、および SSLv3 は古い JVM でサポートされる可能性がありますが、既知のセキュリティー脆弱性により使用は推奨されません。	TLS	文字列
sslProvider (security)	SSL 接続に使用されるセキュリティープロバイダーの名前。デフォルト値は JVM のデフォルトのセキュリティープロバイダーです。		文字列
sslTrustmanagerAlgorithm (security)	SSL 接続のトラストマネージャーファクトリーによって使用されるアルゴリズム。デフォルト値は、Java 仮想マシンに設定されたトラストマネージャーファクトリーアルゴリズムです。	PKIX	文字列
sslTruststoreLocation (security)	トラストストアファイルの場所。		文字列

Name	説明	デフォルト	Type
sslTruststorePassword (security)	トラストストアファイルのパスワード。		文字列
sslTruststoreType (security)	トラストストアファイルのファイル形式。デフォルト値は JKS です。	JKS	文字列

プロデューサー/コンシューマー設定の詳細は、以下を参照してください。

http://kafka.apache.org/documentation.html#newconsumerconfigs
http://kafka.apache.org/documentation.html#producerconfigs

181.3. メッセージヘッダー

181.3.1. コンシューマーヘッダー

以下のヘッダーは、Kafka からメッセージを消費する場合に使用できます。

ヘッダー定数	ヘッダーの値	Type	説明
KafkaConstants.TOPIC	**"kafka.TOPIC"**	**文字列**	メッセージの発信元のトピック
KafkaConstants.PARTITION	**"kafka.PARTITION"**	**整数**	メッセージが保存されるパーティション
KafkaConstants.OFFSET	**"kafka.OFFSET"**	**Long**	メッセージのオフセット
KafkaConstants.KEY	**"kafka.KEY"**	**オブジェクト**	設定されている場合メッセージのキー
KafkaConstants.HEADERS	**"kafka.HEADERS"**	**org.apache.kafka.common.header.Headers**	レコードヘッダー

ヘッダー定数	ヘッダーの値	Type	説明
KafkaConstants.LAST_RECORD_BEFORE_COMMIT	**"kafka.LAST_RECORD_BEFORE_COMMIT"**	**ブール値**	コミット前に最後のレコードとなるかどうか（**autoCommitEnable** エンドポイントパラメーターが **false**の場合のみ利用可能）。
KafkaConstants.MANUAL_COMMIT	**"CamelKafkaManualCommit"**	**KafkaManualCommit**	Kafka コンシューマーを使用する場合に、オフセットの手動コミットを強制するために使用できます。

181.3.2. プロデューサーヘッダー

メッセージを Kafka に送信する前に、以下のヘッダーを設定できます。

ヘッダー定数	ヘッダーの値	Type	説明
KafkaConstants.KEY	**"kafka.KEY"**	**オブジェクト**	**必須**。関連するすべてのメッセージが同じパーティションに配置されるようにするためのメッセージのキー。
KafkaConstants.TOPIC	**"kafka.TOPIC"**	**文字列**	メッセージを送信するトピック（**bridgeEndpoint** エンドポイントパラメーターが **true**の場合のみ読み取り）。
KafkaConstants.PARTITION_KEY	**"kafka.PARTITION_KEY"**	**整数**	パーティションを明示的に指定します（**KafkaConstants.KEY** ヘッダーが定義されている場合のみ使用されます）。

メッセージが Kafka に送信された後に、以下のヘッダーを使用できます。

ヘッダー定数	ヘッダーの値	Type	説明
KafkaConstants.KAFKA_RECORDMETA	**"org.apache.kafka.clients.producer.RecordMetadata"**	**List<RecordMetadata>**	メタデータ（**recordMetadata** エンドポイントパラメーターが **true**の場合のみ設定されます）

181.4. サンプル

181.4.1. Kafka からのメッセージの消費

以下は、Kafka からメッセージを読み取るために必要な最小限のルートです。

```
from("kafka:test?brokers=localhost:9092")
   .log("Message received from Kafka : ${body}")
   .log("    on the topic ${headers[kafka.TOPIC]}")
   .log("    on the partition ${headers[kafka.PARTITION]}")
   .log("    with the offset ${headers[kafka.OFFSET]}")
   .log("    with the key ${headers[kafka.KEY]}")
```

Kafka からメッセージを消費する場合は、独自のオフセット管理を使用し、この管理を Kafka に委譲しないでください。オフセットを保持するには、コンポーネントに File StateRepository などの StateRepository 実装が必要です。この Bean はレジストリーで使用できる必要があります。ここでは、この使用方法を紹介します。

```
// Create the repository in which the Kafka offsets will be persisted
FileStateRepository repository = FileStateRepository.fileStateRepository(new
File("/path/to/repo.dat"));

// Bind this repository into the Camel registry
JndiRegistry registry = new JndiRegistry();
registry.bind("offsetRepo", repository);

// Configure the camel context
DefaultCamelContext camelContext = new DefaultCamelContext(registry);
camelContext.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("kafka:" + TOPIC + "?brokers=localhost:{{kafkaPort}}" +
               "&groupId=A" +                    //
               "&autoOffsetReset=earliest" +          // Ask to start from the beginning if we have
unknown offset
               "&offsetRepository=#offsetRepo")       // Keep the offsets in the previously
configured repository
            .to("mock:result");
   }
});
```

181.4.2. Kafka へのメッセージの生成

以下は、メッセージを Kafka に書き込むために必要な最小限のルートです。

```
from("direct:start")
   .setBody(constant("Message from Camel"))         // Message to send
   .setHeader(KafkaConstants.KEY, constant("Camel")) // Key of the message
```

```
    .to("kafka:test?brokers=localhost:9092");
```

181.5. SSL 設定

Kafka コンポーネントで SSL 通信を設定する方法は 2 つあります。

最初の方法は、多くの SSL エンドポイントパラメーターを使用する方法です。

```
from("kafka:" + TOPIC + "?brokers=localhost:{{kafkaPort}}" +
          "&groupId=A" +
          "&sslKeystoreLocation=/path/to/keystore.jks" +
          "&sslKeystorePassword=changeit" +
          "&sslKeyPassword=changeit")
      .to("mock:result");
```

2 つ目は、sslContextParameters エンドポイントパラメーターを使用することです。

```
// Configure the SSLContextParameters object
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/path/to/keystore.jks");
ksp.setPassword("changeit");
KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("changeit");
SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

// Bind this SSLContextParameters into the Camel registry
JndiRegistry registry = new JndiRegistry();
registry.bind("ssl", scp);

// Configure the camel context
DefaultCamelContext camelContext = new DefaultCamelContext(registry);
camelContext.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("kafka:" + TOPIC + "?brokers=localhost:{{kafkaPort}}" +
                "&groupId=A" +                          //
                "&sslContextParameters=#ssl")           // Reference the SSL configuration
            .to("mock:result");
   }
});
```

181.6. KAFKA のべき等リポジトリーの使用

Camel 2.19 から利用可能

camel-kafka ライブラリーは、Kafka トピックベースのべき等リポジトリーを提供します。このリポジトリーは、Kafka トピックでべき等状態（追加/削除）へのすべての変更をブロードキャストし、イベントソーシングによって各リポジトリーのプロセスインスタンスのローカルインメモリーキャッシュを生成します。

使用するトピックは、べき等リポジトリーインスタンスごとに一意である必要があります。このメカニズムには、トピックパーティションの数の要件はありません。リポジトリーはすべてのパーティションから同時に消費するためです。また、トピックのレプリケーション係数に関する要件はありません。

トピックを使用する各リポジトリーインスタンス（例： 通常、並列で実行されている異なるマシン上）は独自のコンシューマーグループを制御します。そのため、同じトピックを使用して 10 個の Camel プロセスのクラスターで、それぞれ独自のオフセットを制御します。

起動時に、インスタンスはトピックをサブスクライブし、オフセットを最初から戻し、キャッシュを最新の状態に再ビルドします。長さで poll DurationM の 1 回のポーリング が 0 レコードを返すまで、キャッシュは保留状態とみなされます。キャッシュが準備されるまで起動は完了しません。または 30 秒後に発生した場合は、コンシューマーがトピックの最後に追いつくまで、べき等リポジトリーが一貫性のない状態になる可能性があります。

KafkaIdempotentRepository には以下のプロパティーがあります。

プロパティー	説明
topic	変更のブロードキャストに使用する Kafka トピックの名前（必須）。
bootstrapServers	内部 Kafka プロデューサーおよびコンシューマーの **bootstrap.servers** プロパティー。**consumerConfig** および **producerConfig** が設定されていない場合は、これを短縮として使用します。これを使用すると、このコンポーネントはプロデューサーとコンシューマーに適切なデフォルト設定を適用します。
producerConfig	変更をブロードキャストする Kafka プロデューサーによって使用されるプロパティーを設定します。**bootstrapServers** を上書きするため、Kafka **bootstrap.servers** プロパティー自体を定義する必要があります。
consumerConfig	トピックからキャッシュを設定する Kafka コンシューマーによって使用されるプロパティーを設定します。**bootstrapServers** を上書きするため、Kafka **bootstrap.servers** プロパティー自体を定義する必要があります。
maxCacheSize	メモリーに保存される、最近使用されたキーの数（デフォルトは 1000）。

プロパティー	説明
pollDurationMs	Kafka コンシューマーのポーリング期間。ローカルキャッシュは即座に更新されます。この値は、トピックからキャッシュを更新する他のピアが、キャッシュアクションメッセージを送信したべき等コンシューマーインスタンスとの関連に影響します。デフォルト値は 100 ミリ秒です。 この値を明示的に設定する場合、リモートキャッシュの liveness と、このリポジトリーのコンシューマーと Kafka ブローカー間のネットワークトラフィックのボリュームにはトレードオフがあることに注意してください。 キャッシュウォームアッププロセスは、何も取得しないポーリングが 1 つ存在します。これは、ストリームが現在のポイントまで消費されたことを示します。トピックでメッセージが送信されるレートにポーリング期間が過剰に長くなると、キャッシュが準備できず、追いつくまでそのピアに対して一貫性のない状態で動作する可能性があります。

リポジトリーは topic および bootstrapServers を定義してインスタンス化できます。または、producerConfig および consumerConfig プロパティーセットを明示的に定義し、SSL/SASL などの機能を有効にします。

使用するには、手動または Spring/Blueprint で Bean として登録するか、CamelContext を認識しているので、このリポジトリーを Camel レジストリーに置く必要があります。

使用例は次のとおりです。

```
KafkaIdempotentRepository kafkaIdempotentRepository = new
KafkaIdempotentRepository("idempotent-db-inserts", "localhost:9091");

SimpleRegistry registry = new SimpleRegistry();
registry.put("insertDbIdemRepo", kafkaIdempotentRepository); // must be registered in the
registry, to enable access to the CamelContext
CamelContext context = new CamelContext(registry);

// later in RouteBuilder...
from("direct:performInsert")
   .idempotentConsumer(header("id")).messageIdRepositoryRef("insertDbIdemRepo")
      // once-only insert into database
   .end()
```

XML の場合：

```
<!-- simple -->
<bean id="insertDbIdemRepo"
class="org.apache.camel.processor.idempotent.kafka.KafkaIdempotentRepository">
  <property name="topic" value="idempotent-db-inserts"/>
  <property name="bootstrapServers" value="localhost:9091"/>
```

```
</bean>

<!-- complex -->
<bean id="insertDbIdemRepo"
class="org.apache.camel.processor.idempotent.kafka.KafkaIdempotentRepository">
  <property name="topic" value="idempotent-db-inserts"/>
  <property name="maxCacheSize" value="10000"/>
  <property name="consumerConfig">
    <props>
      <prop key="bootstrap.servers">localhost:9091</prop>
    </props>
  </property>
  <property name="producerConfig">
    <props>
      <prop key="bootstrap.servers">localhost:9091</prop>
    </props>
  </property>
</bean>
```

181.7. KAFKA コンシューマーでの手動コミットの使用

Camel 2.21 で利用可能

デフォルトでは、Kafka コンシューマーは自動コミットを使用します。この場合、指定の間隔を使用してオフセットをバックグラウンドで自動的にコミットします。

手動のコミットを強制する場合は、Camel Exchange から KafkaManualCommit API を使用し、メッセージヘッダーに保存されます。これは、以下の ように KafkaComponent またはエンドポイントでManualCommit を true に設定して、手動のコミットを有効にする必要があります。

```
KafkaComponent kafka = new KafkaComponent();
kafka.setAllowManualCommit(true);
...
camelContext.addComponent("kafka", kafka);
```

その後、Camel プロセッサー などの Java コードからの KafkaManualCommit を使用できます。

```
public void process(Exchange exchange) {
    KafkaManualCommit manual =
exchange.getIn().getHeader(KafkaConstants.MANUAL_COMMIT, KafkaManualCommit.class);
    manual.commitSync();
}
```

これにより、Kafka でコミットが確認されるまでブロックされる同期コミットが強制的に実行されます。または、例外がスローされた場合は例外が発生します。

KafkaManualCommit のカスタム実装を使用する場合は、カスタム実装のインスタンスを作成する KafkaComponent でカスタム KafkaManualCommitFactory を設定できます。

181.8. KAFKA ヘッダーの伝播

Camel 2.22 で利用可能

Kafka からメッセージを消費する場合、ヘッダーは自動的に Camel エクスチェンジヘッダーに伝播されます。同じ動作でサポートされるフローの生成： 特定のエクスチェンジの Camel ヘッダーは kafka メッセージヘッダーに伝播されます。

kafka ヘッダーは byte[] 値のみを許可するため、camel exchnage ヘッダーがその値を bytes[] にシリアライズされる必要があります。それ以外の場合は、ヘッダーはスキップされます。以下のヘッダー値型がサポートされます： String、Integer、Long、Double、ブール 値、バイト[]。注記： kafka から camel エクスチェンジに伝撤されるすべてのヘッダー に は、デフォルトで byte[] 値が含まれます。デフォルトの機能 uri パラメーターを上書きするには、ルート から kafkaHeaderDeserializer、ルートには kafkaHeaderSerializer を設定 する ことができます。例:

```
from("kafka:my_topic?kafkaHeaderDeserializer=#myDeserializer")
...
.to("kafka:my_topic?kafkaHeaderSerializer=#mySerializer")
```

デフォルトでは、すべてのヘッダーは KafkaHeaderFilterStrategy によってフィルターされます。strategy は、Camel または org.apache.camel プレフィックスで始まるヘッダーを除外します。デフォルトのストラテジーは、ルート に 対する/ から の headerFilterStrategy uri パラメーターを使用して上書きできます。

```
from("kafka:my_topic?headerFilterStrategy=#myStrategy")
...
.to("kafka:my_topic?headerFilterStrategy=#myStrategy")
```

myStrategy オブジェクトは HeaderFilterStrategy のサブクラスで、手動または Spring/Blueprint で Bean として登録することで、CamelContext を認識しているので Camel レジストリー内に配置する必要があります。

第182章 KESTREL コンポーネント（非推奨）

Camel バージョン 2.6 で利用可能

Kestrel コンポーネントを使用すると、メッセージを Kestrel キューに送信したり、メッセージを Kestrel キューから消費したりできます。このコンポーネントは、Kestrel サーバーとの memcached プロトコル通信に spymemcached クライアントを使用します。

警告

kestrel プロジェクトは非アクティブであり、Camel チームはこのコンポーネントが非推奨になりました。

182.1. URI 形式

```
kestrel://[addresslist/]queuename[?options]
```

queuename は、Kestrel のキューの名前です。URI の addresslist 部分には、1 つ以上の host:port ペアが含まれる場合があります。たとえば、kserver01:22133 のキュー foo に接続するには、以下を使用します。

```
kestrel://kserver01:22133/foo
```

addresslist を省略すると、localhost:22133 が想定されます。つまり、以下のようになります。

```
kestrel://foo
```

同様に、addresslist の host:port ペアからポートを省略すると、デフォルトのポート 22133 と見なされます。

```
kestrel://kserver01/foo
```

以下は、クラスター化キューの生成に使用される Kestrel エンドポイント URI の例です。

```
kestrel://kserver01:22133,kserver02:22133,kserver03:22133/massive
```

以下は、キューから同時に消費するために使用される Kestrel エンドポイント URI の例です。

```
kestrel://kserver03:22133/massive?concurrentConsumers=25&waitTimeMs=500
```

182.2. オプション

Kestrel コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	新規エンドポイントを作成するためのベースとして設定された共有設定を使用するには、以下を実行します。		KestrelConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Kestrel エンドポイントは、URI 構文を使用して設定します。

```
kestrel:addresses/queue
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

182.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
addresses	kestrel が実行されているアドレス	localhost:22133	String[]
queue	**必須**。ポーリングしているキュー		文字列

182.2.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
concurrentConsumers (common)	スレッドプールにスケジュールする同時リスナーの数	1	int
waitTimeM (common)	指定の待機期間（サーバー側）をミリ秒単位でブロックします。	100	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

182.3. SPRING XML を使用した KESTREL コンポーネントの設定

明示的な設定の最も単純な形式は以下のとおりです。

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

  <bean id="kestrel" class="org.apache.camel.component.kestrel.KestrelComponent"/>

  <camelContext xmlns="http://camel.apache.org/schema/spring">
```

```
  </camelContext>

</beans>
```

これにより、すべてのデフォルト設定で Kestrel コンポーネントが有効になります。つまり、デフォルトで localhost:22133、100ms の待機時間、および同時でないコンシューマーを 1 つ使用します。

ベース設定で特定のオプションを使用するには（ ?properties が指定されていないエンドポイントに設定を提供する）、KestrelConfiguration POJO を以下のように設定します。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

  <bean id="kestrelConfiguration" class="org.apache.camel.component.kestrel.KestrelConfiguration">
    <property name="addresses" value="kestrel01:22133"/>
    <property name="waitTimeMs" value="100"/>
    <property name="concurrentConsumers" value="1"/>
  </bean>

  <bean id="kestrel" class="org.apache.camel.component.kestrel.KestrelComponent">
    <property name="configuration" ref="kestrelConfiguration"/>
  </bean>

  <camelContext xmlns="http://camel.apache.org/schema/spring">
  </camelContext>

</beans>
```

182.4. 使用例

182.4.1. 例 1: 消費中

```
from("kestrel://kserver02:22133/massive?concurrentConsumers=10&waitTimeMs=500")
  .bean("myConsumer", "onMessage");
```

```
public class MyConsumer {
    public void onMessage(String message) {
        ...
    }
}
```

182.4.2. 例 2: 生成

```
public class MyProducer {
    @EndpointInject(uri = "kestrel://kserver01:22133,kserver02:22133/myqueue")
    ProducerTemplate producerTemplate;

    public void produceSomething() {
        producerTemplate.sendBody("Hello, world.");
    }
}
```

182.4.3. 例 3: Spring XML 設定

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="kestrel://ks01:22133/sequential?concurrentConsumers=1&waitTimeMs=500"/>
    <bean ref="myBean" method="onMessage"/>
  </route>
  <route>
    <from uri="direct:start"/>
    <to uri="kestrel://ks02:22133/stuff"/>
  </route>
</camelContext>
```

```
public class MyBean {
    public void onMessage(String message) {
        ...
    }
}
```

182.5. 依存関係

Kestrel コンポーネントには、以下の依存関係があります。

- spymemcached 2.5（以上）

182.5.1. spymemcached

クラスパスに spymemcached jar が なけれ ばなりません。以下は、pom.xml で使用できるスニペットです。

```
<dependency>
  <groupId>spy</groupId>
  <artifactId>memcached</artifactId>
  <version>2.5</version>
</dependency>
```

または、jar を直接ダウンロード できます。

警告： 制限

注記

JVM アサーションが有効な場合に、spymemcached クライアントライブラリーは kestrel で適切に動作し ません。アサーションが有効で、要求されたキーに /t=... 拡張が含まれる場合には、spymemcached には既知の問題があります（例： エンドポイント URI で waitTimeMs オプションを使用している場合は、これが推奨されます）。幸い、JVM アサーションは、明示的に有効にしない限り、デフォルトで無効 に なっているため、通常の状況では問題は発生しません。留意すべき点は、Maven の Surefire テストプラグインがアサーションを 有効 にすることです。Maven テスト環境でこのコンポーネントを使用している場合は、enableAssertions を false に設定する必要がある場合があります。詳細は、sesefire :test リファレンス を参照してください。

182.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第183章 KIE-CAMEL

183.1. 概要

KIE-Camel は、KIE(Drools)と統合する Apache Camel コンポーネント（エンドポイント）です。これにより、ルートにプルして実行できる KIE モジュール（maven GAV を使用）を指定できます。また、ファクトとしてメッセージボディーの一部を指定できます。

kie-camel コンポーネントの詳細は、「Apache Camel Integration」を参照してください。

第184章 KRATI コンポーネント（非推奨）

Camel バージョン 2.9 で利用可能

このコンポーネントでは、Camel 内で krati データストアおよびデータセットを使用できます。Krati は、非常に低レイテンシーで高スループットを備えた簡単な永続データストアです。これは、設定、パフォーマンス、および JVM ガベージコレクションの調整をほとんど行うことで、読み書き型のアプリケーションとの統合を容易にするように設計されています。

Camel は krati datastore_(key/value engine)_ のプロデューサーおよびコンシューマーを提供します。また、重複メッセージをフィルタリングするためのべき等リポジトリーも提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-krati</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

184.1. URI 形式

```
krati:[the path of the datastore][?options]
```

データストアのパス は、krati がデータストアに使用するディレクトリーの相対パスです。

URI にクエリーオプションを追加するには、?option=value&option=value&…

184.2. KRATI オプション

Krati コンポーネントにはオプションがありません。

Krati エンドポイントは、URI 構文を使用して設定します。

```
krati:path
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

184.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
path	データストアの **必要な** パスは、krati がデータストアに使用するディレクトリーの相対パスです。		文字列

184.2.2. クエリーパラメーター（29 パラメーター）：

Name	説明	デフォルト	Type
hashFunction (common)	使用するハッシュ関数。		HashFunction<byte[]>
initialCapacity（共通）	ストアの最初の上限。	100	int
keySerializer (common)	キーをシリアライズするために使用されるシリアライザー。		Object>
segmentFactory (common)	ターゲットストアのセグメントファクトリーを設定します。		SegmentFactory
segmentFileSize (common)	データストアセグメントサイズ（MB 単位）	64	int
valueSerializer (common)	値をシリアライズするために使用されるシリアライザー。		Object>
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
maxMessagesPerPoll (consumer)	1 回のポーリングで受信できるメッセージの最大数。これは、大量のデータの読み込みを回避し、メモリーを過剰に消費しないようにすることができます。		int

Name	説明	デフォルト	Type
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
キー （プロデューサー）	キー。		文字列
操作 （プロデューサー）	データストアで実行される操作のタイプを指定します。		文字列
値 （プロデューサー）	値。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int

Name	説明	デフォルト	Type
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit

Name	説明	デフォルト	Type
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

```
krati:/tmp/krati?
operation=CamelKratiGet&initialCapacity=10000&keySerializer=#myCustomSerializer
```

プロデューサーエンドポイントでは、適切なヘッダーをメッセージに渡すことで、上記の URI オプションをすべて上書きできます。

184.2.3. データストアのメッセージヘッダー

ヘッダー	説明
CamelKratiOperation	データストアで実行される操作。有効なオプションは CamelKratiAdd、CamelKratiGet、CamelKratiDeleteAll です。
CamelKratiKey	キー。
CamelKratiValue	値。

184.3. 使用状況サンプル

184.3.1. 例 1: データストアへの配置。

この例では、データストア内にすべてのメッセージを保存する方法を示しています。

```
from("direct:put").to("krati:target/test/producertest");
```

上記の例では、メッセージ上のヘッダーを使用して任意の URI パラメーターを上書きできます。上記の例は、xml を使用してルートを定義する方法を示しています。

```
    <route>
```

```
        <from uri="direct:put"/>
        <to uri="krati:target/test/producerspringtest"/>
      </route>
```

184.3.2. 例 2: データストアの取得/読み取り

この例では、データストアの contnet を読み取る方法を説明します。

```
from("direct:get")
   .setHeader(KratiConstants.KRATI_OPERATION,
constant(KratiConstants.KRATI_OPERATION_GET))
   .to("krati:target/test/producertest");
```

上記の例では、メッセージ上のヘッダーを使用して任意の URI パラメーターを上書きできます。
上記の例は、xml を使用してルートを定義する方法を示しています。

```
<route>
    <from uri="direct:get"/>
    <to uri="krati:target/test/producerspringtest?operation=CamelKratiGet"/>
</route>
```

184.3.3. 例 3: データストアの使用

この例では、指定されたデータストアの配下にあるアイテムをすべて消費します。

```
   from("krati:target/test/consumertest")
      .to("direct:next");
```

以下で示すように xml を使用して同じ目的を達成できます。

```
<route>
   <from uri="krati:target/test/consumerspringtest"/>
   <to uri="mock:results"/>
</route>
```

184.4. べき等リポジトリー

すでに説明したように、重複したメッセージのフィルターに使用できる idemptonet リポジトリーおよび idemptonet リポジトリーも提供しています。

```
from("direct://in").idempotentConsumer(header("messageId"), new
KratiIdempotentRepositroy("/tmp/idempotent").to("log://out");
```

-

184.4.1. 関連項目

- *Krati Web サイト*

第185章 KUBERNETES コンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes コンポーネントは、アプリケーションを Kubernetes スタンドアロンまたは Openshift 上に統合します。

camel-kubernetes は 13 コンポーネントで構成されています。

- *Kubernetes の ConfigMap*
- *Kubernetes Namespace*
- *Kubernetes Node*
- *Kubernetes 永続ボリューム*
- *Kubernetes Persistent Volume Claim（永続ボリューム要求、PVC）*
- *Kubernetes Pod*
- *Kubernetes Replication Controller*
- *Kubernetes Resource Quota*
- *Kubernetes の Secret*
- *Kubernetes サービスアカウント*

- *Kubernetes Service*

OpenShift では、以下も実行できます。

- *Kubernetes ビルド設定*
- *Kubernetes ビルド*

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-kubernetes</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

185.1. HEADERS

Name	タイプ	説明
CamelKubernetesOperation	文字列	プロデューサー操作
CamelKubernetesNamespaceName	文字列	名前空間名
CamelKubernetesNamespaceLabels	マップ	namespace ラベル

Name	タイプ	説明
CamelKubernetesServiceLabels	マップ	サービスのラベル
CamelKubernetesServiceName	文字列	サービス名
CamelKubernetesServiceSpec	io.fabric8.kubernetes.api.model.ServiceSpec	サービスの仕様
CamelKubernetesReplicationControllersLabels	マップ	レプリケーションコントローラーのラベル
CamelKubernetesReplicationControllerName	文字列	レプリケーションコントローラー名
CamelKubernetesReplicationControllerSpec	io.fabric8.kubernetes.api.model.ReplicationControllerSpec	レプリケーションコントローラーの仕様

Name	タイプ	説明
CamelKubernetesReplicationControllerReplicas	整数	Scale 操作時のレプリケーションコントローラーのレプリカ数
CamelKubernetesPodsLabels	マップ	Pod のラベル
CamelKubernetesPodName	文字列	Pod の名前
CamelKubernetesPodSpec	io.fabric8.kubernetes.api.model.PodSpec	Pod の仕様
CamelKubernetesPersistentVolumesLabels	マップ	永続ボリュームラベル
CamelKubernetesPersistentVolumesName	文字列	永続ボリューム名
CamelKubernetesPersistentVolumesClaimsLabels	マップ	Persistent Volume Claim（永続ボリューム要求、PVC）のラベル

Name	タイプ	説明
CamelKubernetesPersistentVolumesClaimsName	文字列	永続ボリューム要求 (PVC) の名前
CamelKubernetesPersistentVolumesClaimsSpec	io.fabric8.kubernetes.api.model.PersistentVolumeClaimSpec	Persistent Volume Claim（永続ボリューム要求、PVC）の仕様
CamelKubernetesSecretsLabels	マップ	シークレットラベル
CamelKubernetesSecretsName	文字列	Secret 名
CamelKubernetesSecret	io.fabric8.kubernetes.api.model.Secret	Secret オブジェクト
CamelKubernetesResourcesQuotaLabels	マップ	リソースクォータのラベル

Name	タイプ	説明
CamelKubernetesResourcesQuotaName	文字列	リソースクォータ名
CamelKubernetesResourceQuotaSpec	io.fabric8.kubernetes.api.model.ResourceQuotaSpec	リソースクォータの仕様
CamelKubernetesServiceAccountsLabels	マップ	サービスアカウントラベル
CamelKubernetesServiceAccountName	文字列	サービスアカウント名
CamelKubernetesServiceAccount	io.fabric8.kubernetes.api.model.ServiceAccount	サービスアカウントオブジェクト
CamelKubernetesNodesLabels	マップ	ノードラベル
CamelKubernetesNodeName	文字列	ノード名

Name	タイプ	説明
CamelKubernetesBuildsLabels	マップ	OpenShift ビルドラベル
CamelKubernetesBuildName	文字列	OpenShift ビルド名
CamelKubernetesBuildConfigsLabels	マップ	OpenShift ビルド設定ラベル
CamelKubernetesBuildConfigName	文字列	OpenShift ビルド設定名
CamelKubernetesEventAction	io.fabric8.kubernetes.client.Watcher.Action	コンシューマーによって監視されるアクション
CamelKubernetesEventTimestamp	文字列	コンシューマーによって監視されるアクションのタイムスタンプ
CamelKubernetesConfigMapName	文字列	ConfigMap 名

Name	タイプ	説明
CamelKubernetesConfigMapsLabels	マップ	ConfigMap ラベル
CamelKubernetesConfigData	マップ	ConfigMap Data

185.2. 用途

185.2.1. プロデューサーの例

ここでは、camel-kubernetes を使用したプロデューサーの例をいくつか紹介します。

185.2.2. Pod の作成

```
from("direct:createPod")
    .toF("kubernetes-pods://%s?oauthToken=%s&operation=createPod", host, authToken);
```

KubernetesConstants.KUBERNETES_POD_SPEC ヘッダーを使用すると、PodSpec を指定し、この操作に渡すことができます。

185.2.3. Pod の削除

```
from("direct:createPod")
    .toF("kubernetes-pods://%s?oauthToken=%s&operation=deletePod", host, authToken);
```

KubernetesConstants.KUBERNETES_POD_NAME ヘッダーを使用して、Pod 名を指定し、この操作に渡すことができます。

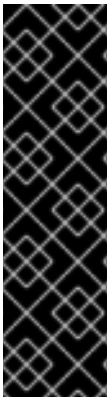
第186章 KUBERNETES コンポーネント（非推奨）

Camel バージョン 2.17 から利用可能

重要

複合 kubernetes コンポーネントは非推奨になりました。以下のように分割された個々のコンポーネントを使用します。

- Kubernetes コンポーネント
  - Kubernetes ビルド設定
  - Kubernetes ビルド
  - Kubernetes の ConfigMap
  - Kubernetes Namespace
  - Kubernetes Node
  - Kubernetes 永続ボリューム
  - Kubernetes Persistent Volume Claim（永続ボリューム要求、PVC）
  - Kubernetes Pod
  - Kubernetes Replication Controller
  - Kubernetes Resource Quota



- *Kubernetes の Secret*
- *Kubernetes サービスアカウント*
- *Kubernetes Service*

Kubernetes コンポーネントは、アプリケーションを Kubernetes スタンドアロンまたは Openshift の上部に統合するためのコンポーネントです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-kubernetes</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

186.1. URI 形式

```
kubernetes:masterUrl[?options]
```

URI にクエリーオプションを追加するには、 ?option=value&option=value&…

186.2. オプション

Kubernetes コンポーネントにはオプションがありません。

Kubernetes エンドポイントは、URI 構文を使用して設定します。

```
kubernetes:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

186.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

186.2.2. クエリーパラメーター（28 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (common)	使用する Kubernetes API バージョン		文字列
カテゴリー (common)	**必要な** Kubernetes Producer および Consumer カテゴリー		文字列
dnsDomain (common)	ServiceCall EIP に使用される dns ドメイン		文字列
kubernetesClient (common)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (common)	ServiceCall EIP に使用されるポート名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	一部のリソースを監視する際の Consumer ラベルキー		文字列
labelValue (consumer)	一部のリソースを監視する際のコンシューマーラベル値		文字列
namespace（コンシューマー）	namespace		文字列
poolSize (consumer)	コンシューマープールのサイズ	1	int

Name	説明	デフォルト	Type
resourceName (consumer)	監視するコンシューマーリソース名		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列

Name	説明	デフォルト	Type
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

186.3. HEADERS

Name	タイプ	説明
CamelKubernetesOperation	文字列	プロデューサー操作
CamelKubernetesNamespaceName	文字列	名前空間名
CamelKubernetesNamespaceLabels	マップ	namespace ラベル
CamelKubernetesServiceLabels	マップ	サービスのラベル
CamelKubernetesServiceName	文字列	サービス名

Name	タイプ	説明
CamelKubernetesServiceSpec	io.fabric8.kubernetes.api.model.ServiceSpec	サービスの仕様
CamelKubernetesReplicationControllersLabels	マップ	レプリケーションコントローラーのラベル
CamelKubernetesReplicationControllerName	文字列	レプリケーションコントローラー名
CamelKubernetesReplicationControllerSpec	io.fabric8.kubernetes.api.model.ReplicationControllerSpec	レプリケーションコントローラーの仕様
CamelKubernetesReplicationControllerReplicas	整数	Scale 操作時のレプリケーションコントローラーのレプリカ数
CamelKubernetesPodsLabels	マップ	Pod のラベル
CamelKubernetesPodName	文字列	Pod の名前

Name	タイプ	説明
CamelKubernetesPodSpec	io.fabric8.kubernetes.api.model.PodSpec	Pod の仕様
CamelKubernetesPersistentVolumesLabels	マップ	永続ボリュームラベル
CamelKubernetesPersistentVolumesName	文字列	永続ボリューム名
CamelKubernetesPersistentVolumesClaimsLabels	マップ	Persistent Volume Claim（永続ボリューム要求、PVC）のラベル
CamelKubernetesPersistentVolumesClaimsName	文字列	永続ボリューム要求 (PVC) の名前
CamelKubernetesPersistentVolumesClaimsSpec	io.fabric8.kubernetes.api.model.PersistentVolumeClaimSpec	Persistent Volume Claim（永続ボリューム要求、PVC）の仕様

Name	タイプ	説明
CamelKubernetesSecretsLabels	マップ	シークレットラベル
CamelKubernetesSecretsName	文字列	Secret 名
CamelKubernetesSecret	io.fabric8.kubernetes.api.model.Secret	Secret オブジェクト
CamelKubernetesResourcesQuotaLabels	マップ	リソースクォータのラベル
CamelKubernetesResourcesQuotaName	文字列	リソースクォータ名
CamelKubernetesResourceQuotaSpec	io.fabric8.kubernetes.api.model.ResourceQuotaSpec	リソースクォータの仕様
CamelKubernetesServiceAccountsLabels	マップ	サービスアカウントラベル

Name	タイプ	説明
CamelKubernetesServiceAccountName	文字列	サービスアカウント名
CamelKubernetesServiceAccount	io.fabric8.kubernetes.api.model.ServiceAccount	サービスアカウントオブジェクト
CamelKubernetesNodesLabels	マップ	ノードラベル
CamelKubernetesNodeName	文字列	ノード名
CamelKubernetesBuildsLabels	マップ	OpenShift ビルドラベル
CamelKubernetesBuildName	文字列	OpenShift ビルド名
CamelKubernetesBuildConfigsLabels	マップ	OpenShift ビルド設定ラベル
CamelKubernetesBuildConfigName	文字列	OpenShift ビルド設定名

Name	タイプ	説明
CamelKubernetesEventAction	io.fabric8.kubernetes.client.Watcher.Action	コンシューマーによって監視されるアクション
CamelKubernetesEventTimestamp	文字列	コンシューマーによって監視されるアクションのタイムスタンプ
CamelKubernetesConfigMapName	文字列	ConfigMap 名
CamelKubernetesConfigMapsLabels	マップ	ConfigMap ラベル
CamelKubernetesConfigData	マップ	ConfigMap Data

186.4. カテゴリー

実際には camel-kubernetes コンポーネントは以下の Kubernetes リソースをサポートします。

- ***Namespaces***
- ***Pod***
- ***レプリケーションコントローラー***

- *サービス*
- ***Persistent Volumes***
- ***Persistent Volume Claim (永続ボリューム要求、PVC)***
- ***Secret***
- *リソースクォータ*
- *サービスアカウント*
- *ノード*
- ***configmaps***

Openshift でも

- *ビルド*
- ***BuildConfig***

186.5. 用途

186.5.1. プロデューサーの例

ここでは、camel-kubernetes を使用したプロデューサーの例をいくつか紹介します。

186.5.2. Pod の作成

```
from("direct:createPod")
    .toF("kubernetes://%s?oauthToken=%s&category=pods&operation=createPod", host,
authToken);
```

KubernetesConstants.KUBERNETES_POD_SPEC ヘッダーを使用すると、PodSpec を指定し、この操作に渡すことができます。

186.5.3. Pod の削除

```
from("direct:createPod")
    .toF("kubernetes://%s?oauthToken=%s&category=pods&operation=deletePod", host,
authToken);
```

KubernetesConstants.KUBERNETES_POD_NAME ヘッダーを使用して、Pod 名を指定し、この操作に渡すことができます。

第187章 KUBERNETES CONFIGMAP COMPONENT

Camel バージョン 2.17 から利用可能

Kubernetes ConfigMap コンポーネントは、kubernetes ConfigMap 操作を実行するためのプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

187.1. コンポーネントオプション

Kubernetes ConfigMap コンポーネントにはオプションがありません。

187.2. エンドポイントオプション

Kubernetes ConfigMap エンドポイントは、URI 構文を使用して設定します。

```
kubernetes-config-maps:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

187.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

187.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	使用する Kubernetes API バージョン		文字列
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		文字列
kubernetesClient (producer)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient

Name	説明	デフォルト	Type
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値

Name	説明	デフォルト	Type
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第188章 KUBERNETES デプロイメントコンポーネント

Camel バージョン 2.20 で利用可能

Kubernetes Deployment コンポーネントは、kubernetes シークレット操作を実行するプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

188.1. コンポーネントオプション

Kubernetes Deployments コンポーネントにはオプションがありません。

188.2. エンドポイントオプション

Kubernetes Deployment エンドポイントは、URI 構文を使用して設定します。

```
kubernetes-deployments:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

188.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

188.2.2. クエリーパラメーター（27 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (common)	使用する Kubernetes API バージョン		文字列
dnsDomain (common)	ServiceCall EIP に使用される dns ドメイン		文字列
kubernetesClient (common)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient

Name	説明	デフォルト	Type
portName (common)	ServiceCall EIP に使用されるポート名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	一部のリソースを監視する際の Consumer ラベルキー		文字列
labelValue (consumer)	一部のリソースを監視する際のコンシューマーラベル値		文字列
namespace （コンシューマー）	namespace		文字列
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視するコンシューマーリソース名		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第189章 KUBERNETES NAMESPACES コンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes Namespaces コンポーネントは Kubernetes コンポーネントの 1 つで、kubernetes namespace 操作および kubernetes namespace イベントを消費するためのコンシューマーを実行するためのプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

189.1. コンポーネントオプション

Kubernetes Namespaces コンポーネントにはオプションがありません。

189.2. エンドポイントオプション

Kubernetes Namespaces エンドポイントは、URI 構文を使用して設定します。

```
kubernetes-namespaces:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

189.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

189.2.2. クエリーパラメーター（27 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (common)	使用する Kubernetes API バージョン		文字列
dnsDomain (common)	ServiceCall EIP に使用される dns ドメイン		文字列

Name	説明	デフォルト	Type
kubernetesClient (common)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (common)	ServiceCall EIP に使用されるポート名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	一部のリソースを監視する際の Consumer ラベルキー		文字列
labelValue (consumer)	一部のリソースを監視する際のコンシューマーラベル値		文字列
namespace （コンシューマー）	namespace		文字列
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視するコンシューマーリソース名		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第190章 KUBERNETES ノードのコンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes Nodes コンポーネントは Kubernetes コンポーネントの 1 つで、kubernetes ノードの操作と、kubernetes ノードイベントを使用するためにコンシューマーを実行するプロデューサーを提供する Kubernetes コンポーネントの 1 つです。 ???

190.1. コンポーネントオプション

Kubernetes Nodes コンポーネントにはオプションがありません。

190.2. エンドポイントオプション

Kubernetes ノードのエンドポイントは、URI 構文を使用して設定します。

```
kubernetes-nodes:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

190.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

190.2.2. クエリーパラメーター（27 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (common)	使用する Kubernetes API バージョン		文字列
dnsDomain (common)	ServiceCall EIP に使用される dns ドメイン		文字列

Name	説明	デフォルト	Type
kubernetesClient (common)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (common)	ServiceCall EIP に使用されるポート名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	一部のリソースを監視する際の Consumer ラベルキー		文字列
labelValue (consumer)	一部のリソースを監視する際のコンシューマーラベル値		文字列
namespace （コンシューマー）	namespace		文字列
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視するコンシューマーリソース名		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第191章 KUBERNETES PERSISTENT VOLUME CLAIM（永続ボリューム要求、PVC）コンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes Persistent Volume Claim （永続ボリューム要求、PVC）コンポーネントは、kubernetes の Persistent Volume Claim（永続ボリューム要求、PVC）の操作を実行するためのプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

191.1. コンポーネントオプション

Kubernetes Persistent Volume Claim コンポーネントにはオプションがありません。

191.2. エンドポイントオプション

Kubernetes Persistent Volume Claim エンドポイントは、URI 構文を使用して設定します。

```
kubernetes-persistent-volumes-claims:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

191.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	必要な Kubernetes マスターの URL		文字列

191.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	使用する Kubernetes API バージョン		文字列
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		文字列

Name	説明	デフォルト	Type
kubernetesClient (producer)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列

Name	説明	デフォルト	Type
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第192章 KUBERNETES 永続ボリュームコンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes 永続ボリュームコンポーネントは 、kubernetes 永続ボリューム操作 を実行するプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

192.1. コンポーネントオプション

Kubernetes Persistent Volume コンポーネントにはオプションがありません。

192.2. エンドポイントオプション

Kubernetes Persistent Volume エンドポイントは、URI 構文を使用して設定します。

```
kubernetes-persistent-volumes:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

192.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

192.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	使用する Kubernetes API バージョン		文字列
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		文字列
kubernetesClient (producer)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient

Name	説明	デフォルト	Type
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値

Name	説明	デフォルト	Type
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第193章 KUBERNETES POD コンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes Pod コンポーネントは Kubernetes コンポーネントの 1 つで、 kubernetes Pod 操作を実行するプロデューサーを提供します。

193.1. コンポーネントオプション

Kubernetes Pod コンポーネントにはオプションがありません。

193.2. エンドポイントオプション

Kubernetes Pod エンドポイントは、URI 構文を使用して設定します。

```
kubernetes-pods:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

193.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

193.2.2. クエリーパラメーター（27 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (common)	使用する Kubernetes API バージョン		文字列
dnsDomain (common)	ServiceCall EIP に使用される dns ドメイン		文字列
kubernetesClient (common)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient

Name	説明	デフォルト	Type
portName (common)	ServiceCall EIP に使用されるポート名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	一部のリソースを監視する際の Consumer ラベルキー		文字列
labelValue (consumer)	一部のリソースを監視する際のコンシューマーラベル値		文字列
namespace （コンシューマー）	namespace		文字列
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視するコンシューマーリソース名		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第194章 KUBERNETES REPLICATION CONTROLLER コンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes Replication Controller コンポーネントは Kubernetes コンポーネントの 1 つで、kubernetes レプリケーションコントローラー操作と、kubernetes レプリケーションコントローラーイベントを消費するためのコンシューマーを実行するためのプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

194.1. コンポーネントオプション

Kubernetes Replication Controller コンポーネントにはオプションがありません。

194.2. エンドポイントオプション

Kubernetes Replication Controller エンドポイントは、URI 構文を使用して設定します。

```
kubernetes-replication-controllers:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

194.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

194.2.2. クエリーパラメーター（27 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (common)	使用する Kubernetes API バージョン		文字列
dnsDomain (common)	ServiceCall EIP に使用される dns ドメイン		文字列

Name	説明	デフォルト	Type
kubernetesClient (common)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (common)	ServiceCall EIP に使用されるポート名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	一部のリソースを監視する際の Consumer ラベルキー		文字列
labelValue (consumer)	一部のリソースを監視する際のコンシューマーラベル値		文字列
namespace （コンシューマー）	namespace		文字列
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視するコンシューマーリソース名		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第195章 KUBERNETES リソースクォータコンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes Resources Quota コンポーネントは、kubernetes リソースのクォータ操作 を実行するプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

195.1. コンポーネントオプション

Kubernetes Resources Quota コンポーネントにはオプションがありません。

195.2. エンドポイントオプション

Kubernetes Resources Quota エンドポイントは URI 構文を使用して設定されます。

```
kubernetes-resources-quota:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

195.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

195.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	使用する Kubernetes API バージョン		文字列
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		文字列
kubernetesClient (producer)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient

Name	説明	デフォルト	Type
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値

Name	説明	デフォルト	Type
ユーザー名（セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第196章 KUBERNETES の SECRET コンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes Secrets コンポーネントは、kubernetes シークレット操作を実行するプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

196.1. コンポーネントオプション

Kubernetes Secrets コンポーネントにはオプションがありません。

196.2. エンドポイントオプション

Kubernetes の Secrets エンドポイントは、URI 構文を使用して設定します。

```
kubernetes-secrets:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

196.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

196.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	使用する Kubernetes API バージョン		文字列
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		文字列
kubernetesClient (producer)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient

Name	説明	デフォルト	Type
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値

Name	説明	デフォルト	Type
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第197章 KUBERNETES サービスアカウントコンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes Service Account コンポーネントは、kubernetes サービスアカウント操作 を実行するプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

197.1. コンポーネントオプション

Kubernetes Service Account コンポーネントにはオプションがありません。

197.2. エンドポイントオプション

Kubernetes サービスアカウントエンドポイントは、URI 構文を使用して設定します。

```
kubernetes-service-accounts:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

197.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

197.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	使用する Kubernetes API バージョン		文字列
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		文字列
kubernetesClient (producer)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient

Name	説明	デフォルト	Type
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値

Name	説明	デフォルト	Type
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第198章 KUBERNETES サービスコンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes サービスコンポーネントは 、kubernetes サービス操作と、kubernetes サービスイベントを消費するコンシューマーを実行するためのプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

198.1. コンポーネントオプション

Kubernetes サービスコンポーネントにはオプションがありません。

198.2. エンドポイントオプション

Kubernetes サービスエンドポイントは、URI 構文を使用して設定します。

```
kubernetes-services:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

198.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

198.2.2. クエリーパラメーター（27 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (common)	使用する Kubernetes API バージョン		文字列
dnsDomain (common)	ServiceCall EIP に使用される dns ドメイン		文字列

Name	説明	デフォルト	Type
kubernetesClient (common)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient
portName (common)	ServiceCall EIP に使用されるポート名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
labelKey (consumer)	一部のリソースを監視する際の Consumer ラベルキー		文字列
labelValue (consumer)	一部のリソースを監視する際のコンシューマーラベル値		文字列
namespace （コンシューマー）	namespace		文字列
poolSize (consumer)	コンシューマープールのサイズ	1	int
resourceName (consumer)	監視するコンシューマーリソース名		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

198.3. ECLIPSE KURA コンポーネント

Camel 2.15 から利用可能

このドキュメントページでは、Camel と Eclipse Kura M2M ゲートウェイとの統合オプションについて説明します。Camel ルートを Eclipse Kura にデプロイする一般的な理由は、エンタープライズ統合パターンと Camel コンポーネントをメッセージング M2M ゲートウェイに提供することです。たとえば、Kura を Raspberry PI にインストールし、Kura サービスを使用してその Raspberry PI にアタッチされたセンサーから温度を読み、最後に Camel EIP やコンポーネントを使用して現在の温度値をデータセンターサービスに転送するとします。

198.3.1. KuraRouter activator

Eclipse Kura にデプロイされたバンドルは通常、bundle activators として開発 されます。そのため、Apache Camel ルートを Kura にデプロイする最も簡単な方法は、org.apache.camel.kura.KuraRouter クラスを拡張するクラスを含む OSGi バンドルを作成することです。

```
public class MyKuraRouter extends KuraRouter {

  @Override
  public void configure() throws Exception {
    from("timer:trigger").
      to("netty-http:http://app.mydatacenter.com/api");
  }

}
```

KuraRouter は org.osgi.framework.BundleActivator インターフェースを実装するため、 Kura バンドルコンポーネントクラスの作成 時に 開始 および 停止 ライフサイクルメソッドを登録する必要があります。

Kura ルーターは、独自の OSGi 対応の CamelContext を起動します。つまり、KuraRouter を拡張するすべてのクラスに対して専用の CamelContext インスタンスがあります。OSGi バンドルごとに KuraRouter を 1 台デプロイすることを推奨します。

198.3.2. Deploying KuraRouter

Kura ルータークラスを含むバンドルは、OSGi マニフェストに以下のパッケージをインポートする必要があります。

```
Import-Package: org.osgi.framework;version="1.3.0",
  org.slf4j;version="1.6.4",

org.apache.camel,org.apache.camel.impl,org.apache.camel.core.osgi,org.apache.camel.builder,org.apache.camel.model,
  org.apache.camel.component.kura
```

Camel コンポーネントはランタイムレベルでサービスとして解決されるため、ルートで使用する予定の Camel コンポーネントバンドルをすべてインポートする必要がないことに注意してください。

ルーターバンドルをデプロイする前に、以下の Camel コアバンドルをデプロイ（および起動）し（Kura GoGo shell を使用）...

```
install file:///home/user/.m2/repository/org/apache/camel/camel-core/2.15.0/camel-core-2.15.0.jar
start <camel-core-bundle-id>
install file:///home/user/.m2/repository/org/apache/camel/camel-core-osgi/2.15.0/camel-core-osgi-2.15.0.jar
start <camel-core-osgi-bundle-id>
install file:///home/user/.m2/repository/org/apache/camel/camel-kura/2.15.0/camel-kura-2.15.0.jar
start <camel-kura-bundle-id>
```

また、ルートで使用する予定のすべてのコンポーネント：

```
install file:///home/user/.m2/repository/org/apache/camel/camel-stream/2.15.0/camel-stream-2.15.0.jar
start <camel-stream-bundle-id>
```

次に、最後にルーターバンドルをデプロイします。

```
install file:///home/user/.m2/repository/com/example/myrouter/1.0/myrouter-1.0.jar
start <your-bundle-id>
```

198.3.3. KuraRouter utilities

Kura ルーターベースクラスは、多くの便利なユーティリティーを提供します。このセクションでは、各項目について説明します。

198.3.3.1. SLF4J logger

Kura はロギングに SLF4J ファサードを使用します。保護されたメンバー ログ は、指定の Kura ルーターに関連付けられた SLF4J ロガーインスタンスを返します。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        log.info("Configuring Camel routes!");
```

```
        ...
    }

}
```

198.3.3.2. BundleContext

protected member bundleContext は、指定の Kura ルーターに関連付けられたバンドルコンテキストを返します。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        ServiceReference<MyService> serviceRef =
bundleContext.getServiceReference(LogService.class.getName());
        MyService myService = bundleContext.getService(serviceRef);
        ...
    }

}
```

198.3.3.3. CamelContext

保護されたメンバー camelContext は、指定の Kura ルーターに関連付けられた CamelContext です。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        camelContext.getStatus();
        ...
    }

}
```

198.3.3.4. ProducerTemplate

protected member producerTemplate は、指定の Camel コンテキストに関連付けられた ProducerTemplate インスタンスです。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        producerTemplate.sendBody("jms:temperature", 22.0);
```

```
        ...
    }

}
```

198.3.3.5. ConsumerTemplate

protected member consumerTemplate は、指定の Camel コンテキストに関連付けられた ConsumerTemplate インスタンスです。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        double currentTemperature = producerTemplate.receiveBody("jms:temperature",
Double.class);
        ...
    }

}
```

198.3.3.6. OSGi サービスリゾルバー

OSGi サービスリゾルバー（service(Class<T> serviceType)）は、OSGi バンドルコンテキストからタイプ別にサービスを簡単に取得するために使用できます。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        MyService myService = service(MyService.class);
        ...
    }

}
```

service が見つからない場合は、null 値が返されます。サービスが利用できない場合にアプリケーションが失敗する場合は、代わりに requiredService(Class) メソッドを使用します。サービスが見つからない場合、requiredService は IllegalStateException をスローします。

```
public class MyKuraRouter extends KuraRouter {

    @Override
    public void configure() throws Exception {
        MyService myService = requiredService(MyService.class);
        ...
```

```
    }

}
```

198.3.4. KuraRouter activator コールバック

Kura ルーターには、Camel ルーターの動作をカスタマイズするために使用できる lifecycle コールバックが同梱されています。たとえば、前者を起動する直前にルーターに関連付けられた CamelContext インスタンスを設定するには、KuraRouter クラスの beforeStart メソッドを上書きします。

```
public class MyKuraRouter extends KuraRouter {

 ...

 protected void beforeStart(CamelContext camelContext) {
  OsgiDefaultCamelContext osgiContext = (OsgiCamelContext) camelContext;
  osgiContext.setName("NameOfTheRouter");
 }

}
```

198.3.5. ConfigurationAdmin からの XML ルートの読み込み

サーバー設定からルートの XML 定義を読み取る必要がある場合があります。この一般的なシナリオは、OTA(over-the-air)の再デプロイメントコストが大きい可能性がある IoT ゲートウェイの一般的なシナリオです。この要件を満たすには、各 KuraRouter は OSGi ConfigurationAdmin を使用して kura.camel.BUNDLE-SYMBOLIC-NAME.route プロパティーを検索します。この方法では、デプロイされた KuraRouter ごとに Camel XML ルートファイルを定義できます。ルートを更新するには、適切な設定プロパティーを編集し、それに関連するバンドルを再起動するだけです。kura.camel.BUNDLE-SYMBOLIC-NAME.route プロパティーの内容は、以下のような Camel XML ルートファイルであることが予想されます。

```
<routes xmlns="http://camel.apache.org/schema/spring">
   <route id="loaded">
      <from uri="direct:bar"/>
      <to uri="mock:bar"/>
   </route>
</routes>
```

198.3.6. Kura ルーターを宣言型の OSGi サービスとしてデプロイ

Kura ルーターを宣言型の OSGi サービスとしてデプロイする場合は、KuraRouter が提供する アクティブ化 および 非 アクティブ化方法を使用できます。

```
<scr:component name="org.eclipse.kura.example.camel.MyKuraRouter" activate="activate"
deactivate="deactivate" enabled="true" immediate="true">
  <implementation class="org.eclipse.kura.example.camel.MyKuraRouter"/>
</scr:component>
```

198.3.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第199章 言語コンポーネント

Camel バージョン 2.5 で利用可能

language コンポーネントでは、Camel でサポートされる言語のいずれかによってスクリプトを実行するエンドポイントに Exchange を送信できます。
言語スクリプトを実行するコンポーネントを使用すると、より動的なルーティング機能が可能になります。たとえば、Routing Slip または Dynamic Router EIP を使用すると、スクリプトも動的定義されている 言語 のエンドポイントにメッセージを送信できます。

このコンポーネントは camel-core で追加設定なしで提供されるため、追加の JAR は必要ありません。Groovy や JavaScript 言語の使用など、選択した言語が義務付けられている場合に限り、追加の Camel コンポーネントを含める必要があります。

199.1. URI 形式

```
language://languageName[:script][?options]
```

Camel 2.11 以降では、Camel の他の 言語 でサポートされている表記と同じ表記を使用して、スクリプトの外部リソースを参照できます。

```
language://languageName:resource:scheme:location][?options]
```

199.2. URI オプション

Language コンポーネントにはオプションがありません。

Language エンドポイントは URI 構文を使用します。

```
language:languageName:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

199.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
languageName	**Required**: 使用する言語の名前を設定します。		文字列
resourceUri	リソースへのパス、またはリソースとして使用するレジストリーで Bean を検索するための参照		文字列

199.2.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
バイナリー（プロデューサー）	スクリプトがバイナリーコンテンツかテキストコンテンツであるか。デフォルトでは、スクリプトはテキストコンテンツとして読み取られます（例：java.lang.String）。	false	boolean
cacheScript (producer)	コンパイルされたスクリプトをキャッシュし、スクリプトを再使用すると、1 つの Camel org.apache.camel.Exchange を次の org.apache.camel.Exchange に処理できない可能性があります。	false	boolean
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
スクリプト（プロデューサー）	実行するスクリプトを設定します。		文字列
変換（プロデューサー）	スクリプトの結果をメッセージボディーとして使用するかどうか。このオプションはデフォルトの true です。	true	boolean
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

199.3. メッセージヘッダー

以下のメッセージヘッダーを使用して、コンポーネントの動作に影響を与えることができます。

ヘッダー	説明
CamelLanguageScript	ヘッダーで提供されるスクリプトを実行するスクリプト。エンドポイントに設定されたスクリプトよりも優先されます。

199.4. 例

たとえば、Simple 言語を使用して、メッセージをメッセージ変換できます。

メッセージボディーの型を変換する場合は、以下も実行できます。

また、入力メッセージが 2 で乗算される例など、Groovy 言語を使用することもできます。

また、以下のようにスクリプトをヘッダーとして提供することもできます。ここでは、XPath 言語を使用して < foo > タグからテキストを抽出します。

```
Object out = producer.requestBodyAndHeader("language:xpath", "<foo>Hello World</foo>",
Exchange.LANGUAGE_SCRIPT, "/foo/text()");
assertEquals("Hello World", out);
```

199.5. リソースからのスクリプトの読み込み

Camel 2.9 で利用可能

エンドポイント URI または Exchange.gitopsUAGE_SCRIPT ヘッダーのいずれかで、スクリプトを読み込むリソース URI を指定できます。
uri は、file:、classpath:、または http のいずれかのスキームのいずれかで開始する必要があります。

たとえば、クラスパスからスクリプトを読み込むには、次のコマンドを実行します。

デフォルトでは、スクリプトは 1 度ロードされ、キャッシュされます。ただし、contentCache オプションを無効にして、各評価にスクリプトを読み込むことができます。
たとえば、ディスク上で myscript.txt ファイルを変更すると、更新されたスクリプトが使用されます。

Camel 2.11 以降では、以下のように「resource :」のプレフィックスを指定して、Camel の他の 言語に似たリソース を参照できます。

第200章 LDAP コンポーネント

Camel バージョン 1.5 で利用可能

ldap コンポーネントでは、フィルターをメッセージペイロードとして使用して LDAP サーバーで検索を実行できます。
このコンポーネントは、標準の JNDI（javax.naming パッケージ）を使用してサーバーにアクセスします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ldap</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

200.1. URI 形式

```
ldap:ldapServerBean[?options]
```

URI の ldapServerBean の部分はレジストリーの DirContext Bean を参照します。LDAP コンポーネントはプロデューサーエンドポイントのみをサポートします。つまり、ldap URI はルートの開始時に表示できません。

URI にクエリーオプションを追加するには、?option=value&option=value&…

200.2. オプション

LDAP コンポーネントにはオプションがありません。

LDAP エンドポイントは URI 構文を使用します。

```
ldap:dirContextName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

200.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
dirContextName	javax.naming.directory.DirContext または java.util.Hashtable のいずれかの **必須** 名、または Bean をレジストリーで検索するためにマッピングします。Bean が Hashtable または Map のいずれかである場合は、使用ごとに新しい javax.naming.directory.DirContext インスタンスが作成されます。Bean が javax.naming.directory.DirContext の場合、Bean は指定されたものとして使用されます。javax.naming.directory.DirContext を共有できないすべての状況では、後者では不可能ではない可能性があります。このような場合には、代わりに java.util.Hashtable または Map を使用することが推奨されます。		文字列

200.2.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
ベース （プロデューサー）	検索用のベース DN。	ou=system	文字列
pageSize (producer)	ldap モジュールがページングを使用してすべての結果を取得し（ほとんどの LDAP サーバーは、1 つのクエリーで 1000 を超えるエントリーを取得しようとすると例外をスローします）。これを使用できるようにするには、LdapContext（DirContext のサブクラス）を ldapServerBean として渡す必要があります（それ以外の場合は例外が発生します）。		整数
returnedAttributes (producer)	結果の各エントリーに設定すべき属性のコンマ区切りリスト		文字列
scope (producer)	ベース DN から順にエントリーのツリーを検索する方法を指定します。	subtree	文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

200.3. 結果

結果は、Out ボディーで ArrayList<javax.naming.directory.SearchResult > オブジェクトとして返

されます。

200.4. DIRCONTEXT

URI ldap:ldapserver は、ldapserver の ID で Spring Bean を参照します。ldapserver Bean は以下のように定義できます。

```
<bean id="ldapserver" class="javax.naming.directory.InitialDirContext" scope="prototype">
  <constructor-arg>
    <props>
      <prop key="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</prop>
      <prop key="java.naming.provider.url">ldap://localhost:10389</prop>
      <prop key="java.naming.security.authentication">none</prop>
    </props>
  </constructor-arg>
</bean>
```

上記の例では、ローカルでホストされる LDAP サーバーに匿名で接続する通常の Sun ベースの LDAP DirContext を宣言します。

注記

DirContext オブジェクトは、コントラクトによる同時実行のサポートには必要 ありません。そのため、ディレクトリーコンテキストは Bean 定義で設定、または コンテキストが 同時実行をサポートする状態で宣言することが重要です。Spring フレームワークでは、プロトタイプ スコープオブジェクトはルックアップされるたびにインスタンス化されます。

200.5. サンプル

上記の Spring 設定から、以下のコードサンプルは、グループでメンバーを検索するための LDAP リクエストを送信します。その後、Common Name が応答から抽出されます。

```
ProducerTemplate<Exchange> template = exchange
  .getContext().createProducerTemplate();

Collection<?> results = (Collection<?>) (template
  .sendBody(
    "ldap:ldapserver?base=ou=mygroup,ou=groups,ou=system",
    "(member=uid=huntc,ou=users,ou=system)"));

if (results.size() > 0) {
  // Extract what we need from the device's profile

  Iterator<?> resultIter = results.iterator();
```

```
SearchResult searchResult = (SearchResult) resultIter
    .next();
Attributes attributes = searchResult
    .getAttributes();
Attribute deviceCNAttr = attributes.get("cn");
String deviceCN = (String) deviceCNAttr.get();

...
```

特定のフィルターが必要ない場合（たとえば、単一のエントリーを検索するだけで、ワイルドカードフィルター式を指定する必要があります）。たとえば、LDAP エントリーに Common Name がある場合は、以下のようなフィルター式を使用します。

```
(cn=*)
```

200.5.1. 認証情報を使用したバインディング

Camel エンドユーザーは、クレデンシャルを使用して ldap サーバーにバインドするために使用されるサンプルコードを提供していました。

```
Properties props = new Properties();
props.setProperty(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
props.setProperty(Context.PROVIDER_URL, "ldap://localhost:389");
props.setProperty(Context.URL_PKG_PREFIXES, "com.sun.jndi.url");
props.setProperty(Context.REFERRAL, "ignore");
props.setProperty(Context.SECURITY_AUTHENTICATION, "simple");
props.setProperty(Context.SECURITY_PRINCIPAL, "cn=Manager");
props.setProperty(Context.SECURITY_CREDENTIALS, "secret");

SimpleRegistry reg = new SimpleRegistry();
reg.put("myldap", new InitialLdapContext(props, null));

CamelContext context = new DefaultCamelContext(reg);
context.addRoutes(
   new RouteBuilder() {
      public void configure() throws Exception {
         from("direct:start").to("ldap:myldap?base=ou=test");
      }
   }
);
context.start();

ProducerTemplate template = context.createProducerTemplate();

Endpoint endpoint = context.getEndpoint("direct:start");
Exchange exchange = endpoint.createExchange();
exchange.getIn().setBody("(uid=test)");
Exchange out = template.send(endpoint, exchange);

Collection<SearchResult> data = out.getOut().getBody(Collection.class);
```

```
assert data != null;
assert !data.isEmpty();

System.out.println(out.getOut().getBody());

context.stop();
```

200.6. SSL の設定

必要なのは、カスタムソケットファクトリーを作成し、InitialDirContext Bean で参照することです。以下の例を参照してください。

SSL 設定

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
              http://camel.apache.org/schema/blueprint http://camel.apache.org/schema/blueprint/camel-
blueprint.xsd">


   <sslContextParameters xmlns="http://camel.apache.org/schema/blueprint"
                       id="sslContextParameters">
      <keyManagers
            keyPassword="{{keystore.pwd}}">
         <keyStore
               resource="{{keystore.url}}"
               password="{{keystore.pwd}}"/>
      </keyManagers>
   </sslContextParameters>

   <bean id="customSocketFactory" class="zotix.co.util.CustomSocketFactory">
      <argument ref="sslContextParameters" />
   </bean>
   <bean id="ldapserver" class="javax.naming.directory.InitialDirContext" scope="prototype">
      <argument>
         <props>
            <prop key="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
            <prop key="java.naming.provider.url" value="ldaps://lab.zotix.co:636"/>
            <prop key="java.naming.security.protocol" value="ssl"/>
            <prop key="java.naming.security.authentication" value="simple" />
            <prop key="java.naming.security.principal" value="cn=Manager,dc=example,dc=com"/>
            <prop key="java.naming.security.credentials" value="passw0rd"/>
            <prop key="java.naming.ldap.factory.socket"
                 value="zotix.co.util.CustomSocketFactory"/>
         </props>
      </argument>
   </bean>
</blueprint>
```

カスタムソケットファクトリー

```
import org.apache.camel.util.jsse.SSLContextParameters;

import javax.net.SocketFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.TrustManagerFactory;
import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.security.KeyStore;

/**
 * The CustomSocketFactory. Loads the KeyStore and creates an instance of
SSLSocketFactory
 */
public class CustomSocketFactory extends SSLSocketFactory {

    private static SSLSocketFactory socketFactory;

    /**
     * Called by the getDefault() method.
     */
    public CustomSocketFactory() {

    }

    /**
     * Called by Blueprint DI to initialise an instance of SocketFactory
     *
     * @param sslContextParameters
     */
    public CustomSocketFactory(SSLContextParameters sslContextParameters) {
        try {
            KeyStore keyStore =
sslContextParameters.getKeyManagers().getKeyStore().createKeyStore();
            TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
            tmf.init(keyStore);
            SSLContext ctx = SSLContext.getInstance("TLS");
            ctx.init(null, tmf.getTrustManagers(), null);
            socketFactory = ctx.getSocketFactory();
        } catch (Exception ex) {
            ex.printStackTrace(System.err);  /* handle exception */
        }
    }

    /**
     * Getter for the SocketFactory
     *
     * @return
     */
    public static SocketFactory getDefault() {
        return new CustomSocketFactory();
```

```
    }

    @Override
    public String[] getDefaultCipherSuites() {
        return socketFactory.getDefaultCipherSuites();
    }

    @Override
    public String[] getSupportedCipherSuites() {
        return socketFactory.getSupportedCipherSuites();
    }

    @Override
    public Socket createSocket(Socket socket, String string, int i, boolean bln) throws
IOException {
        return socketFactory.createSocket(socket, string, i, bln);
    }

    @Override
    public Socket createSocket(String string, int i) throws IOException {
        return socketFactory.createSocket(string, i);
    }

    @Override
    public Socket createSocket(String string, int i, InetAddress ia, int i1) throws IOException {
        return socketFactory.createSocket(string, i, ia, i1);
    }

    @Override
    public Socket createSocket(InetAddress ia, int i) throws IOException {
        return socketFactory.createSocket(ia, i);
    }

    @Override
    public Socket createSocket(InetAddress ia, int i, InetAddress ia1, int i1) throws IOException
{
        return socketFactory.createSocket(ia, i, ia1, i1);
    }
}
```

200.7. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*

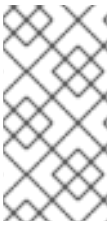
- *エンドポイント*
- *はじめに*

第201章 LDIF コンポーネント

Camel バージョン 2.20 で利用可能

ldif コンポーネントを使用すると、LDIF ボディーコンテンツから LDAP サーバーの更新を実行できます。

このコンポーネントは、基本的な URL 構文を使用してサーバーにアクセスします。Apache DS LDAP ライブラリーを使用して LDIF を処理します。LDIF の処理後、応答本体は各エントリーの成功/失敗のステータスの一覧です。



注記

Apache LDAP API は、LDIF 構文エラーに非常に機密です。不明な場合は、ユニットテストを参照して、それぞれの変更タイプの例を確認してください。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ldif</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

201.1. URI 形式

```
ldap:ldapServerBean[?options]
```

URI の ldapServerBean の部分は LdapConnection を参照します。これは、接続タイムアウトを回避するために、使用される時点のファクトリーから構築する必要があります。LDIF コンポーネントはプロデューサーエンドポイントのみをサポートします。つまり、ldif URI はルートの開始時に表示できません。

SSL 設定については、カスタム SocketFactory インスタンスの設定例がある camel-ldap コンポーネントを参照してください。

URI にクエリーオプションを追加するには、?option=value&option=value&...

201.2. オプション

LDIF コンポーネントにはオプションがありません。

LDIF エンドポイントは、URI 構文を使用して設定します。

```
ldif:ldapConnectionName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

201.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
ldapConnectionName	レジストリーからプルする LdapConnection Bean の名前が **必要です**。スレッド間で共有されないようにしたり、タイムアウトした接続を使用したりするには、スコーププロトタイプである必要があることに注意してください。		文字列

201.2.2. クエリーパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

201.3. ボディーのタイプ：

ボディーは LDIF ファイルまたはインライン LDIF ファイルの URL です。本文タイプの相違点を示すには、インライン LDIF を開始する必要があります。

```
version: 1
```

そうでない場合には、コンポーネントはボディーを URL として解析しようとします。

201.4. 結果

結果は、Out ボディーで ArrayList<java.lang.String& gt; オブジェクトとして返されます。これには、各 LDIF エントリーの "success" または Exception メッセージのいずれかが含まれます。

201.5. LDAPCONNECTION

URI ldif:ldapConnectionName は、ldapConnectionName の ID を持つ Bean を参照します。ldapConnection は LdapConnectionConfig Bean を使用して設定できます。接続が共有または古くなった接続を取得しないように、スコープには プロトタイプ の範囲を指定する必要があることに注意してください。

LdapConnection Bean は Spring XML で以下のように定義できます。

```
<bean id="ldapConnectionOptions"
class="org.apache.directory.ldap.client.api.LdapConnectionConfig">
  <property name="ldapHost" value="${ldap.host}"/>
  <property name="ldapPort" value="${ldap.port}"/>
  <property name="name" value="${ldap.username}"/>
  <property name="credentials" value="${ldap.password}"/>
  <property name="useSsl" value="false"/>
  <property name="useTls" value="false"/>
</bean>

<bean id="ldapConnectionFactory"
class="org.apache.directory.ldap.client.api.DefaultLdapConnectionFactory">
  <constructor-arg index="0" ref="ldapConnectionOptions"/>
</bean>

<bean id="ldapConnection" factory-bean="ldapConnectionFactory" factory-
method="newLdapConnection" scope="prototype"/>
```

または、OSGi の blueprint.xml で以下を行います。

```
<bean id="ldapConnectionOptions"
class="org.apache.directory.ldap.client.api.LdapConnectionConfig">
  <property name="ldapHost" value="${ldap.host}"/>
  <property name="ldapPort" value="${ldap.port}"/>
  <property name="name" value="${ldap.username}"/>
  <property name="credentials" value="${ldap.password}"/>
  <property name="useSsl" value="false"/>
  <property name="useTls" value="false"/>
```

```
</bean>

<bean id="ldapConnectionFactory"
class="org.apache.directory.ldap.client.api.DefaultLdapConnectionFactory">
  <argument ref="ldapConnectionOptions"/>
</bean>

<bean id="ldapConnection" factory-ref="ldapConnectionFactory" factory-
method="newLdapConnection" scope="prototype"/>
```

201.6. サンプル

上記の Spring 設定から、以下のコードサンプルは、グループでメンバーを検索するための LDAP リクエストを送信します。その後、Common Name が応答から抽出されます。

```
ProducerTemplate<Exchange> template = exchange.getContext().createProducerTemplate();

List<?> results = (Collection<?>) template.sendBody("ldap:ldapConnection, "LDiff goes
here");

if (results.size() > 0) {
  // Check for no errors

  for (String result : results) {
    if ("success".equalTo(result)) {
      // LDIF entry success
    } else {
      // LDIF entry failure
    }
  }
}
```

201.7. LEVELDB

Camel 2.10 で利用可能

leveldb は、非常に軽量で埋め込み可能なキー値データベースです。これにより、Camel とともに Aggregator などのさまざまな Camel 機能に永続的なサポートを提供できます。

提供されている現在の機能：

- LevelDBAggregationRepository

201.7.1. LevelDBAggregationRepository の使用

LevelDBAggregationRepository は AggregationRepository で、集約されたメッセージを即座に永続化します。これにより、デフォルトのアグリゲーターは AggregationRepository のみのメモリーで使用されるため、メッセージを緩めることはありません。

このオプションには以下のオプションが含まれます。

オプション	型	説明
repositoryName	文字列	必須のリポジトリー名。複数のリポジトリーに共有 **LevelDBFile** を使用できます。
persistentFileName	文字列	永続ストレージのファイル名。起動時にファイルが存在しない場合には、新しいファイルが作成されます。
levelDBFile	LevelDBFile	既存の設定された **org.apache.camel.component.leveldb.LevelDBFile** インスタンスを使用します。
sync	boolean	**Camel 2.12:** LevelDBFile が書き込み時に同期するかどうか。デフォルトは false です。書き込み時に同期することで、すべての書き込みがディスクにスプーピングされるのを常に待機し、アップデートが悪化しないようにします。非同期と同期書き込み の詳細は、levelDB ドキュメント を参照してください。
returnOldExchange	boolean	get 操作によって古い既存の Exchange が返されるかどうか。デフォルトでは、このオプションは最適化時に古いエクスチェンジを必要としないため、最適化するために **false** になっています。
useRecovery	boolean	リカバリーが有効になっているかどうか。このオプションはデフォルトで **true** です。Camel Aggregator の自動リカバリーの失敗されたエクスチェンジのリカバリーを有効にして、再提出します。
recoveryInterval	Long	リカバリーが有効になっている場合、バックグラウンドタスクは、失敗したエクスチェンジをスキャンして再送信するために、失敗したエクスチェンジをスキャンするたびに実行されます。デフォルトでは、この間隔は 5000 ミリ秒です。
maximumRedeliveries	int	リカバリーされたエクスチェンジの再配信試行の最大数を制限できます。有効にすると、再配信の試行に失敗した場合、エクスチェンジはデッドレターチャネルに移動します。デフォルトでは、このオプションは無効になっています。このオプションを使用する場合は、**deadLetterUri** オプションも指定する必要があります。

オプション	型	説明
deadLetterUri	文字列	リカバリーされたエクスチェンジが移動される Dead Letter Channel のエンドポイント URI。このオプションを使用する場合は、max **Redeliveries** オプションも指定する必要があります。

repositoryName オプションを指定する必要があります。次に、persistentFileName または levelDBFile のいずれかを指定する必要があります。

201.7.2. 永続時に保持されるもの

LevelDBAggregationRepository は、概念実証と互換性のある メッセージ のボディーデータタイプのみを保持します。メッセージヘッダーは、プリミティブ/文字列/数字の / などにする必要があります。データ型がドロップされたタイプではなく、WARN がログに記録されます。そして、メッセージ本文とメッセージヘッダーのみが永続化 さ れます。エクスチェンジプロパティー は 永続化され ません。

201.7.3. 復元

LevelDBAggregationRepository はデフォルトで失敗したエクスチェンジを復元します。これは、永続ストアで失敗したエクスチェンジの有無をスキャンするバックグラウンドタスクを設定することによって行われます。checkInterval オプションを使用して、このタスクの実行頻度を設定できます。リカバリーはトランザクションとして機能し、Camel が失敗したエクスチェンジのリカバリーおよび再配信を試行します。復元されたエクスチェンジは永続ストアから復元され、再送信されて再度送信されます。

Exchange がリカバリーまたは再配信されるときに、以下のヘッダーが設定されます。

ヘッダー	Type	説明
Exchange.REDELIVERED	ブール値	エクスチェンジが再配信されることを示すには、true に設定します。

ヘッダー	Type	説明
Exchange.REDELIVERY_COUNTER	整数	1 から開始する再配信の試行。

Exchange が正常に処理された場合にのみ、完了とマークされます。これは、確認 メソッドが AggregationRepository で呼び出されると発生します。つまり、同じエクスチェンジが再び失敗すると、成功するまで再試行されます。

maximumRedeliveries オプションを使用して、特定のリカバリーされたエクスチェンジの再配信の最大試行回数を制限できます。また、maximumRedeliveries に達したときに Camel が Exchange を送信する場所を認識できるように deadLetterUri オプションも設定する必要があります。

この テストなど、camel-leveldb のユニットテストにはいくつかの例があります。

201.7.3.1. Java DSL での LevelDBAggregationRepository の使用

この例では、target/data/leveldb.dat ファイルで集約されたメッセージを永続化します。

201.7.3.2. Spring XML での LevelDBAggregationRepository の使用

同じ例で、代わりに Spring XML を使用します。

201.7.4. 依存関係

camel ルートで LevelDB を使用するには、camel-leveldb の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-leveldb</artifactId>
```

```
  <version>2.10.0</version>
</dependency>
```

201.7.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- アグリゲーター
- HawtDB
- コンポーネント

第202章 LINKEDIN コンポーネント

Camel バージョン 2.14 から利用可能

LinkedIn コンポーネントは、 https://developer.linkedin.com/rest で文書化されたすべての LinkedIn REST API へのアクセスを提供します。

LinkedIn は、すべてのクライアントアプリケーション認証に OAuth2.0 を使用します。アカウントで camel-linkedin を使用するには、https://www.linkedin.com/secure/developer で LinkedIn の新しいアプリケーションを作成する必要があります。LinkedIn アプリケーションのクライアント ID およびシークレットは、現行ユーザーを必要とする LinkedIn REST API へのアクセスを許可します。ユーザーのアクセストークンは、エンドユーザーのコンポーネントで生成および管理されます。または、Camel アプリケーションは org.apache.camel.component.linkedin.api.OAuthSecureStorage の実装を登録し、org.apache.camel.component.linkedin.apiOAuthToken OAuth トークンを提供できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-linkedin</artifactId>
  <version>${camel-version}</version>
</dependency>
```

202.1. URI 形式

```
linkedin://endpoint-prefix/endpoint?[options]
```

エンドポイントプレフィックスは以下のいずれかになります。

- コメント
- 企業
- groups

- *ジョブ*
- *ユーザー*
- ***posts***
- ***search***

202.2. LINKEDINCOMPONENT

Linkedin コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有設定の使用		LinkedInConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Linkedin エンドポイントは URI 構文を使用して設定します。

```
linkedin:apiName/methodName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

202.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
apiName	**必要な** 操作の種類		LinkedInApiName
methodName	選択した操作に使用するサブ操作が **必要**		文字列

Name	説明	デフォルト	Type

202.2.2. クエリーパラメーター（14 パラメーター）：

Name	説明	デフォルト	Type
clientId (common)	LinkedIn アプリケーションクライアント ID		文字列
clientSecret (common)	LinkedIn アプリケーションクライアントシークレット		文字列
httpParams (common)	カスタム HTTP パラメーター（プロキシーホストおよびポートなど）は、AllClientPNames の定数を使用します。		マップ
inBody (common)	エクスチェンジ In Body で渡されるパラメーターの名前を設定します。		文字列
lazyAuth (common)	レイジー OAuth を有効または無効にするフラグ。デフォルトは true です。有効にされる場合、OAuth トークンの取得または生成は最初の REST 呼び出しになるまで実行されません。	true	boolean
redirectUri (common)	アプリケーションのリダイレクト URI は、正常に機能するリダイレクトサーバーを持たないため、このページにはリダイレクトされません。そのため、テストには https://localhostを使用できます。		文字列
scopes (common)	https://developer.linkedin.com/documents/authenticationgrantingで指定される LinkedIn スコープのリスト		OAuthScope[]
secureStorage (common)	OAuth トークンを提供するコールバックインターフェース、またはコンポーネントによって生成されたトークンを保存するコールバックインターフェース。コールバックは最初の呼び出しで null を返し、作成されたトークンを saveToken（）コールバックに保存します。コールバックが初めて null を返す場合は、userPassword を指定する必要があります。		OAuthSecureStorage
userName (common)	LinkedIn ユーザーアカウント名。指定する必要があります		文字列
userPassword (common)	LinkedIn アカウントパスワード		文字列

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

202.3. プロデューサーエンドポイント：

プロデューサーエンドポイントは、エンドポイント接頭辞の後にエンドポイント名および関連するオプションを使用できます。省略形のエイリアスは、一部のエンドポイントに使用できます。エンドポイント URI には接頭辞が含まれている必要があります。

必須ではないエンドポイントオプションは [] で表されます。エンドポイントに必須オプションがない場合は、[] オプションのセットの 1 つを指定する必要があります。プロデューサーエンドポイントは、特別なオプション inBody を使用することもできます。そのオプションには、値が Camel Exchange In メッセージに含まれる endpoint オプションの名前が含まれる必要があります。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は CamelLinkedIn.<option> 形式である必要があります。inBody オプションはメッセージヘッダーを上書きします（例： body = option のエンドポイントオプションは CamelLinkedIn.option ヘッダーを上書きします）。

エンドポイントおよびオプションの詳細は、 https://developer.linkedin.com/rest の LinkedIn REST API ドキュメントを参照してください。

202.3.1. エンドポイント接頭辞の コメント

以下のエンドポイントは、以下のように接頭辞の コメント を使用して呼び出すことができます。

linkedin://comments/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
getComment	comment	comment_id, fields	org.apache.camel.component.linkedin.api.model.Comment
removeComment	comment	comment_id	

コメントの URI オプション

Name	Type
comment_id	文字列
fields	文字列

202.3.2. エンドポイント接頭辞の 企業

以下のエンドポイントは、以下のように、プレフィックス company で呼び出すことができます。

linkedin://companies/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
addCompanyUpdateComment	companyUpdateComment	company_id, update_key, updatecomment	
addCompanyUpdateCommentAsCompany	companyUpdateCommentAsCompany	company_id, update_key, updatecomment	
addShare	share	company_id, share	

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
getCompanies	企業	email_domain, fields, is_company_admin	org.apache.camel.component.linkedin.api.model.Companies
getCompanyById	companyById	company_id, fields	org.apache.camel.component.linkedin.api.model.Company
getCompanyByName	companyByName	fields, universal_name	org.apache.camel.component.linkedin.api.model.Company
getCompanyUpdateComments	companyUpdateComments	company_id, fields, secure_urls, update_key	org.apache.camel.component.linkedin.api.model.Comments
getCompanyUpdateLikes	companyUpdateLikes	company_id, fields, secure_urls, update_key	org.apache.camel.component.linkedin.api.model.Likes
getCompanyUpdates	companyUpdates	company_id、count、event_type、field、start	org.apache.camel.component.linkedin.api.model.Updates
getHistoricalFollowStatistics	historicalFollowStatistics	company_id, end_timestamp, start_timestamp, time_granularity	org.apache.camel.component.linkedin.api.model.HistoricalFollowStatistics
getHistoricalStatusUpdateStatistics	historicalStatusUpdateStatistics	company_id, end_timestamp, start_timestamp, time_granularity, update_key	org.apache.camel.component.linkedin.api.model.HistoricalStatusUpdateStatistics
getNumberOfFollowers	numberOfFollowers	companySizes, company_id, geos, industries, jobFunc, seniorities	org.apache.camel.component.linkedin.api.model.NumFollowers
getStatistics	statistics	company_id	org.apache.camel.component.linkedin.api.model.CompanyStatistics
isShareEnabled		company_id	org.apache.camel.component.linkedin.api.model.IsCompanyShareEnabled

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
isViewerShareEnabled		company_id	org.apache.camel.component.linkedin.api.model.IsCompanyShareEnabled
likeCompanyUpdate		company_id, isliked, update_key	

企業の URI オプション

[companySizes, count, email_domain, end_timestamp, event_type, geos, geos, is_company_admin, jobFunc, secure_urls, highities, start, start_timestamp, time_granularity] のいずれかのオプションに値が指定されてい ない場合、 null と見なされます。null 値 は、他のオプションが一致するエンドポイントを満たさない場合にのみ使用されることに注意してください。

Name	Type
companySizes	java.util.List
company_id	Long
count	Long
email_domain	文字列
end_timestamp	Long
event_type	org.apache.camel.component.linkedin.api.Eventtype
fields	文字列
GEO	java.util.List
業界	java.util.List
is_company_admin	ブール値
isliked	org.apache.camel.component.linkedin.api.model.IsLiked
jobFunc	java.util.List

Name	Type
secure_urls	ブール値
上級者	java.util.List
share	org.apache.camel.component.linkedin.api.model.Share
start	Long
start_timestamp	Long
time_granularity	org.apache.camel.component.linkedin.api.Timegranularity
universal_name	文字列
update_key	文字列
updatecomment	org.apache.camel.component.linkedin.api.model.UpdateComment

202.3.3. エンドポイント接頭辞 グループ

以下のエンドポイントは、以下のようにプレフィックス グループ で呼び出すことができます。

linkedin://groups/endpoint?[options]

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
addPost	POST	group_id, post	
getGroup	group	group_id	org.apache.camel.component.linkedin.api.model.Group

グループの URI オプション

Name	Type
group_id	Long

Name	Type
POST	org.apache.camel.component.linkedin.api.model.Post

202.3.4. エンドポイントプレフィックス ジョブ

以下のエンドポイントは、以下のようにプレフィックス ジョブ で呼び出すことができます。

```
linkedin://jobs/endpoint?[options]
```

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
addJob	job	job	
editJob		job, partner_job_id	
getJob	job	フィールド job_id	org.apache.camel.component.linkedin.api.model.Job

ジョブの URI オプション

Name	Type
fields	文字列
job	org.apache.camel.component.linkedin.api.model.Job
job_id	Long
partner_job_id	Long

202.3.5. エンドポイントの接頭辞

以下のエンドポイントは、以下のように接頭辞の people で呼び出すことができます。

```
linkedin://people/endpoint?[options]
```

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
addActivity	activity	activity	
addGroupMembership	groupMembership	groupmembership	
addInvite	招待	mailboxitem	
addJobBookmark	jobBookmark	jobbookmark	
addUpdateComment	updateComment	update_key, updatecomment	
followCompany		会社	
getConnections	connections	フィールド secure_urls	org.apache.camel.component.linkedin.api.model.Connections
getConnectionsById	connectionsById	fields、person_id、secure_urls	org.apache.camel.component.linkedin.api.model.Connections
getConnectionsByUrl	connectionsByUrl	fields、public_profile_url、secure_urls	org.apache.camel.component.linkedin.api.model.Connections
getFollowedCompanies	followedCompanies	fields	org.apache.camel.component.linkedin.api.model.Companies
getGroupMembershipSettings	groupMembershipSettings	count、field、group_id、start	org.apache.camel.component.linkedin.api.model.GroupMemberships
getGroupMemberships	groupMemberships	count, fields, membership_state, start	org.apache.camel.component.linkedin.api.model.GroupMemberships
getJobBookmarks	jobBookmarks		org.apache.camel.component.linkedin.api.model.JobBookmarks
getNetworkStats	networkStats		org.apache.camel.component.linkedin.api.model.NetworkStats

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
getNetworkUpdates	networkUpdates	after, before, count, fields, scope, secure_urls, show_hidden_members, start, type	org.apache.camel.component.linkedin.api.model.Updates
getNetworkUpdatesById	networkUpdatesById	after, before, count, fields, person_id, scope, secure_urls, show_hidden_members, start, type	org.apache.camel.component.linkedin.api.model.Updates
getPerson	person	フィールド secure_urls	org.apache.camel.component.linkedin.api.model.Person
getPersonById	personById	fields、person_id、secure_urls	org.apache.camel.component.linkedin.api.model.Person
getPersonByUrl	personByUrl	fields、public_profile_url、secure_urls	org.apache.camel.component.linkedin.api.model.Person
getPosts	posts	category、count、field、group_id、modified_since、order、role、start	org.apache.camel.component.linkedin.api.model.Posts
getSuggestedCompanies	suggestedCompanies	fields	org.apache.camel.component.linkedin.api.model.Companies
getSuggestedGroupPosts	suggestedGroupPosts	category、count、field、group_id、modified_since、order、role、start	org.apache.camel.component.linkedin.api.model.Posts
getSuggestedGroups	suggestedGroups	fields	org.apache.camel.component.linkedin.api.model.Groups
getSuggestedJobs	suggestedJobs	fields	org.apache.camel.component.linkedin.api.model.JobSuggestions
getUpdateComments	updateComments	fields、secure_urls、update_key	org.apache.camel.component.linkedin.api.model.Comments

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
getUpdateLikes	updateLikes	fields、secure_urls、update_key	org.apache.camel.component.linkedin.api.model.Likes
likeUpdate		isliked, update_key	
removeGroupMembership	groupMembership	group_id	
removeGroupSuggestion	groupSuggestion	group_id	
removeJobBookmark	jobBookmark	job_id	
share		share	org.apache.camel.component.linkedin.api.model.Update
stopFollowingCompany		company_id	
updateGroupMembership		group_id、groupmembership	

ユーザーの URI オプション

オプションのいずれかの値が指定されていない場合、[after、before、category、count、membership_state、modified_since、order、public_profile_url、role、scope、secure_urls、show_hidden_members、start、type] をエンドポイント URI またはメッセージヘッダーのいずれかで指定し、 null と見なされます。null 値 は、他のオプションが一致するエンドポイントを満たさない場合にのみ使用されることに注意してください。

Name	Type
activity	org.apache.camel.component.linkedin.api.model.Activity
after	Long
before	Long

Name	Type
category	org.apache.camel.component.linkedin.api.Category
会社	org.apache.camel.component.linkedin.api.model.Company
company_id	Long
count	Long
fields	文字列
group_id	Long
groupmembership	org.apache.camel.component.linkedin.api.model.GroupMembership
isliked	org.apache.camel.component.linkedin.api.model.IsLiked
job_id	Long
jobbookmark	org.apache.camel.component.linkedin.api.model.JobBookmark
mailboxitem	org.apache.camel.component.linkedin.api.model.MailboxItem
membership_state	org.apache.camel.component.linkedin.api.model.MembershipState
modified_since	Long
order	org.apache.camel.component.linkedin.api.Order
person_id	文字列
public_profile_url	文字列
role	org.apache.camel.component.linkedin.api.Role
scope	文字列
secure_urls	ブール値
share	org.apache.camel.component.linkedin.api.model.Share

Name	Type
show_hidden_members	ブール値
start	Long
type	org.apache.camel.component.linkedin.api.Type
update_key	文字列
updatecomment	org.apache.camel.component.linkedin.api.model.UpdateComment

202.3.6. エンドポイントプレフィックスの 投稿

以下のエンドポイントは、以下のようにプレフィックス post を使用して 呼び出す ことができます。

> ***linkedin://posts/endpoint?[options]***

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
addComment	comment	comment, post_id	
flagCategory		post_id, postcategorycode	
followPost		isfollowing, post_id	
getPost	POST	count、field、post_id、start	org.apache.camel.component.linkedin.api.model.Post
getPostComments	postComments	count、field、post_id、start	org.apache.camel.component.linkedin.api.model.Comments
likePost		isliked, post_id	
removePost	POST	post_id	

postsの *URI* オプション

オプション [count, start] のいずれかに値が指定されていない場合、エンドポイント URI またはメッセージヘッダーのいずれかで値が指定されていない場合、 null と見なされます。null 値 は、他のオプションが一致するエンドポイントを満たさない場合にのみ使用されることに注意してください。

Name	Type
comment	org.apache.camel.component.linkedin.api.model.Comment
count	Long
fields	文字列
フォロー中	org.apache.camel.component.linkedin.api.model.IsFollowing
isliked	org.apache.camel.component.linkedin.api.model.IsLiked
post_id	文字列
postcategorycode	org.apache.camel.component.linkedin.api.model.PostCategoryCode
start	Long

202.3.7. エンドポイント接頭辞の 検索

以下のエンドポイントは、以下のように接頭辞 search で呼び出すことができます。

> ***linkedin://search/endpoint?[options]***

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
searchCompanies	企業	count, facet, facets, fields, hq_only, keywords, sort, start	org.apache.camel.component.linkedin.api.model.CompanySearch

エンドポイント	短縮エイリアス	オプション	結果ボディーのタイプ
searchJobs	ジョブ	company_name、count、country_code、distance、fett、Fast、field、job_title、keyword、postal_code、sort、start	org.apache.camel.component.linkedin.api.model.JobSearch
searchPeople	ユーザー	company_name, count, country_code, current_company, current_school, current_title, distance, facet, facets, fields, first_name, keywords, last_name, postal_code, school_name, sort, start, title	org.apache.camel.component.linkedin.api.model.PeopleSearch

検索の URI オプション

[company_name, count, country_code, current_company, current_school, current_title, distance, facet, facets, first_name, hq_only, job_title, keywords, last_name, postal_code, school_name, sort, start, title] のいずれかのオプションに値が指定されていない場合は、エンドポイント URI またはメッセージヘッダーのいずれかで null と見なされます。null 値 は、他のオプションが一致するエンドポイントを満たさない場合にのみ使用されることに注意してください。

Name	Type
company_name	文字列
count	Long
country_code	文字列
current_company	文字列
current_school	文字列
current_title	文字列
distance	org.apache.camel.component.linkedin.api.model.Distance

Name	Type
facet	文字列
facets	文字列
fields	文字列
first_name	文字列
hq_only	文字列
job_title	文字列
keywords	文字列
last_name	文字列
postal_code	文字列
school_name	文字列
sort	文字列
start	Long
title	文字列

202.4. コンシューマーエンドポイント

*プロデューサーエンドポイントはいずれもコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、 **consumer.** プレフィックスと共に Scheduled Poll Consumer オプション を使用してエンドポイント呼び出しをスケジュールできます。デフォルトでは、配列またはコレクションを返すコンシューマーエンドポイントは要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 度実行されます。この動作を変更するには、**consumer.splitResults=true** プロパティーを使用してリストまたはアレイ全体の単一のエクスチェンジを返します。*

202.5. メッセージヘッダー

*すべての URI オプションは、プロデューサーエンドポイントのメッセージヘッダーに **CamelLinkedIn.** プレフィックスで指定できます。*

202.6. メッセージボディー

すべての結果メッセージ本文は、Apache CXF JAX-RS を使用してビルドされた Camel LinkedIn API SDK によって提供されるオブジェクトを使用します。プロデューサーエンドポイントは、inBody エンドポイントパラメーターに受信メッセージボディーのオプション名を指定できます。

202.7. ユースケース

以下のルートはユーザーのプロファイルを取得します。

```
from("direct:foo")
    .to("linkedin://people/person");
```

以下のルートは、30 秒ごとにユーザーの接続をポーリングします。

```
from("linkedin://people/connections?consumer.timeUnit=SECONDS&consumer.delay=30")
    .to("bean:foo");
```

以下のルートは、動的ヘッダーオプションでプロデューサーを使用します。 personId ヘッダーには LinkedIn 人 ID があるため、以下のように CamelLinkedIn.person_id ヘッダーに割り当てられます。

```
from("direct:foo")
    .setHeader("CamelLinkedIn.person_id", header("personId"))
    .to("linkedin://people/connectionsById")
    .to("bean://bar");
```

第203章 ログコンポーネント

Camel バージョン 1.1 で利用可能

log: コンポーネントは、メッセージを基礎となるロギングメカニズムに記録します。

Camel は sfl4j を使用します。これにより、以下のようなロギングを設定できます。

- Log4j
- Logback
- Java Util Logging

203.1. URI 形式

```
log:loggingCategory[?options]
```

loggingCategory は、使用するロギングカテゴリーの名前です。URI にクエリーオプションを追加するには、?option=value&option=value&... を使用します。

INFO:* レジストリーからのロガーインスタンスの使用* Camel 2.12.4/2.13.1 としての Logger インスタンスの使用。レジストリーにある org.slf4j.Logger の単一インスタンスがある場合、ロギングカテゴリー はロガーインスタンスの作成に使用されなくなりました。代わりに登録されたインスタンスが使用されます。また、特定のロガーインスタンスは ?logger=#my Logger URI パラメーターを使用して参照することもできます。最終的に、登録済みおよび URI ロガー パラメーターがない場合、ロギングカテゴリー を使用してロガーインスタンスが作成されます。

たとえば、ログエンドポイントは、通常以下のように level オプションを使用してログ レベル を指定します。

```
log:org.apache.camel.example?level=DEBUG
```

デフォルトのロガーはすべてのエクスチェンジ（通常のロギング）をログに記録します。しかし、Camel には Throughput ロガーが同梱さ れ ます。これは、groupSize オプションが指定されている場

合は常に使用されます。

*TIP:*DSL のログには直接ログがあります* DSL に直接 ログ がありますが、目的は異なります。軽量ログおよびヒューマンログ用詳細は「LogEIP」を参照してください。*

203.2. オプション

Log コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
exchangeFormatter (advanced)	カスタム ExchangeFormatter を設定して、Exchange をロギングに適した String に変換します。指定のない場合は、デフォルトで DefaultExchangeFormatter になります。		ExchangeFormatter
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Log エンドポイントは URI 構文を使用します。

```
log:loggerName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

203.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
loggerName	使用するロガー名が **必要です**。		文字列

203.2.2. クエリーパラメーター（26 パラメーター）：

Name	説明	デフォルト	Type
groupActiveOnly (producer)	true の場合、新しいメッセージが間隔（false の場合）受信されない場合の統計を非表示にし、メッセージトラフィックに関係なく統計を表示します。	true	ブール値
groupDelay (producer)	統計の初期遅延を設定します（ミリ秒単位）。		Long
groupInterval (producer)	指定された場合は、この時間間隔でメッセージ統計をグループ化します（ミリ秒単位）。		Long
groupSize (producer)	スループットロギングにグループサイズを指定する整数。		整数
レベル （プロデューサー）	使用するロギングレベル。デフォルト値は INFO です。	INFO	文字列
logMask (producer)	true の場合、パスワードやパスフレーズなどの機密情報をログでマスクします。		ブール値
marker (producer)	使用するオプションのマーカー名。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
maxChars （形式）	行ごとにログに記録する文字数を制限します。	10000	int
複数行 （形式）	有効にすると、各情報は改行で出力されます。	false	boolean
showAll (formatting)	すべてのオプションをオンにするクイックオプション（マルチライン、maxChars を使用する場合は手動で設定する必要があります）	false	boolean
showBody (formatting)	メッセージのボディーを表示します。	true	boolean
showBodyType (formatting)	ボディーの Java タイプを表示します。	true	boolean

Name	説明	デフォルト	Type
showCaughtException (formatting)	エクスチェンジにキャッチされた例外があり、例外メッセージ（スタックトレースなし）を表示します。キャッチされた例外はエクスチェンジのプロパティーとして保存されます（キーリンク org.apache.camel.ExchangeEXCEPTION_CAUGHT を使用し、たとえば doCatch は例外をキャッチできます）。	false	boolean
showException (formatting)	エクスチェンジに例外がある場合は、例外メッセージを表示します（スタックトレースなし）。	false	boolean
showExchangeId (formatting)	一意のエクスチェンジ ID を表示します。	false	boolean
showExchangePattern (formatting)	メッセージ交換パターン（略して MEP）を表示します。	true	boolean
showFiles (formatting)	有効な Camel がファイルを出力する場合	false	boolean
showFuture (formatting)	有効にすると、Camel が Future オブジェクトで、ペイロードがログに記録されるのを待ちます。	false	boolean
showHeaders (formatting)	メッセージヘッダーを表示します。	false	boolean
showOut (formatting)	エクスチェンジにメッセージがある場合は、out メッセージを表示します。	false	boolean
showProperties (formatting)	エクスチェンジプロパティーを表示します。	false	boolean
showStackTrace (formatting)	エクスチェンジに例外がある場合に、スタックトレースを表示します。showAll、showException、または showCaughtException のいずれかが有効な場合のみ有効です。	false	boolean
showStreams (formatting)	Camel がストリーム本文を表示するかどうか（例：java.io.InputStream）。このオプションを有効にすると、ストリームがすでにこのロガーによって読み込まれているため、後でメッセージボディーにアクセスできなくなる可能性があります。これに対応するには、Stream Caching を使用する必要があります。	false	boolean

Name	説明	デフォルト	Type
skipBodyLineSeparator (formatting)	メッセージのボディーをログに記録する際に改行記号をスキップするかどうか。これにより、メッセージボディーを 1 行に記録できます。このオプションを false に設定すると、本文と改行文字が保持され、ボディーはそのままログに記録されます。	true	boolean
スタイル（形式）	使用する出力スタイルを設定します。	デフォルト	OutputStyle

203.3. 通常のロガーの例

以下のルートでは、注文を処理する前に、DEBUG レベルで受信順序をログに記録します。

```
from("activemq:orders").to("log:com.mycompany.order?
level=DEBUG").to("bean:processOrder");
```

または、Spring XML を使用してルートを定義します。

```
<route>
  <from uri="activemq:orders"/>
  <to uri="log:com.mycompany.order?level=DEBUG"/>
  <to uri="bean:processOrder"/>
</route>
```

203.4. フォーマッターサンプルを含む通常のロガー

以下のルートでは、注文を処理する前に INFO レベルで受信順序をログに記録します。

```
from("activemq:orders").
    to("log:com.mycompany.order?showAll=true&multiline=true").to("bean:processOrder");
```

203.5. GROUPSIZE サンプルでのスループットロガー

以下のルートでは、10 メッセージごとにグループ化された DEBUG レベルで受信順序のスループットをログに記録します。

```
from("activemq:orders").
    to("log:com.mycompany.order?level=DEBUG&groupSize=10").to("bean:processOrder");
```

203.6. GROUPINTERVAL サンプルでのスループットロガー

このルートにより、メッセージ統計が 10s ごとにログに記録され、メッセージトラフィックがなくても統計が表示されます。

```
from("activemq:orders").
    to("log:com.mycompany.order?
level=DEBUG&groupInterval=10000&groupDelay=60000&groupActiveOnly=false").to("bean:pr
ocessOrder");
```

以下のログに記録されます。

```
"Received: 1000 new messages, with total 2000 so far. Last group took: 10000 millis which is: 100
messages per second. average: 100"
```

203.7. パスワードなどの機密情報のマスク

Camel 2.19 から利用可能

logMask フラグを true に設定すると、ロギングのセキュリティーマスクを有効にできます。このオプションは Log EIP にも影響することに注意してください。

CamelContext レベルで Java DSL でマスクを有効にするには、以下を実行します。

```
camelContext.setLogMask(true);
```

XML では、以下のようになります。

```
<camelContext logMask="true">
```

また、エンドポイントレベルでオンオフにすることもできます。エンドポイントレベルで Java DSL でマスクを有効にするには、ログエンドポイントの URI に logMask=true オプションを追加します。

```
from("direct:start").to("log:foo?logMask=true");
```

XML では、以下のようになります。

```
<route>
  <from uri="direct:foo"/>
  <to uri="log:foo?logMask=true"/>
</route>
```

org.apache.camel.processor.DefaultMaskingFormatter は、デフォルトでマスクに使用されます。カスタムマスクフォーマッターを使用する場合は、CamelCustomLogMask の名前でレジストリーに配置します。マスクフォーマッターは org.apache.camel.spi.MaskingFormatter を実装する必要があることに注意してください。

203.8. ロギング出力の完全なカスタマイズ

Camel 2.11 から利用可能

#Formatting セクションで説明されているオプションを使用して、ロガーの出力量を制御できます。ただし、ログ行は常に以下の構造に従います。

```
Exchange[Id:ID-machine-local-50656-1234567901234-1-2, ExchangePattern:InOut,
Properties:{CamelToEndpoint=log://org.apache.camel.component.log.TEST?showAll=true,
CamelCreatedTimestamp=Thu Mar 28 00:00:00 WET 2013},
Headers:{breadcrumbId=ID-machine-local-50656-1234567901234-1-1}, BodyType:String, Body:Hello
World, Out: null]
```

場合によっては、この形式は適切ではありません。

- *...: 出力されるヘッダーとプロパティーをフィルターして、洞察と詳細のバランスを深めます。*
- *常に読み取り可能なものにログメッセージを調節します。*
- *... ログのマイニングによるログメッセージのダイジェストの調整（例： Splunk）。*
- *... 特定のボディーのタイプを異なる出力します。*
- *...*

絶対的なカスタマイズが必要な場合は、ExchangeFormatter インターフェースを実装するクラスを作成できます。完全 Exchange へのアクセスが可能な形式(Exchange) メソッド内で必要な正確な情報を選択して抽出し、カスタム方法でフォーマットし、返すことができます。戻り値は最終的なログメッセージになります。

Log コンポーネントは、以下のいずれかの方法でカスタム ExchangeFormatter を取得できます。

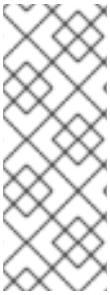
レジストリーで LogComponent を明示的にインスタンス化します。

```
<bean name="log" class="org.apache.camel.component.log.LogComponent">
  <property name="exchangeFormatter" ref="myCustomFormatter" />
</bean>
```

203.8.1. 設定に関する規則：*

logFormatter という名前の Bean を登録するだけです。Log コンポーネントは、自動的に選択できるインテリジェントなコンポーネントです。

```
<bean name="logFormatter" class="com.xyz.MyCustomExchangeFormatter" />
```



注記

ExchangeFormatter は、Camel Context 内のすべての Log エンドポイント に適用されます。異なるエンドポイントに異なる ExchangeFormatters が必要な場合は、LogComponent を必要な回数だけインスタンス化し、関連する Bean 名をエンドポイント接頭辞として使用します。

カスタムログフォーマッターの使用時に Camel 2.11.2/2.12 以降では、カスタムログフォーマッターに設定されるログ URI でパラメーターを指定できます。ただし、パラメーターが異なる場合は共有されないように、logFormatter をプロトタイプスコープとして定義する必要があります（例：

```
<bean name="logFormatter" class="com.xyz.MyCustomExchangeFormatter" scope="prototype"/>
```

その後、異なるオプションを持つログ URI を使用して Camel ルートを設定できます。

```
<to uri="log:foo?param1=foo&amp;param2=100"/>

<to uri="log:bar?param1=bar&amp;param2=200"/>
```

203.9. OSGI でのログコンポーネントの使用

Camel 2.12.4/2.13.1 の時点での改善

OSGi 内で Log コンポーネントを使用する場合（Karaf で）、基礎となるロギングメカニズムは PAX ロギングによって提供されます。org.slf4j.LoggerFactory.getLogger（） メソッドを呼び出すバンドルを検索し、バンドルをロガーインスタンスに関連付けます。カスタム org.sfl4j.Logger インスタンスを指定しないと、Log コンポーネントによって作成されたロガーは camel-core バンドルに関連付けられます。

シナリオによっては、ロガーに関連付けられたバンドルをルート定義が含まれるバンドルにする必要があります。これを実行するには、レジストリーで org.slf4j.Logger の単一インスタンスを登録するか、 ロガー URI パラメーターを使用して参照します。

203.10. 関連項目

- ヒューマンログ用に DSL で直接 ログ を使用するための LogEIP

第204章 LUCENE コンポーネント

Camel バージョン 2.2 で利用可能

lucene コンポーネントは Apache Lucene プロジェクトをベースにしています。Apache Lucene は、Java で完全に書かれた強力な高パフォーマンスのフル機能のテキスト検索エンジンライブラリーです。Lucene の詳細は、以下のリンクを参照してください。

- http://lucene.apache.org/java/docs/
- http://lucene.apache.org/java/docs/features.html

camel の lucene コンポーネントは、エンタープライズ統合パターンおよびシナリオでの Lucene エンドポイントの統合および使用を容易にします。lucene コンポーネントは以下を行います。

- ペイロードが Lucene エンドポイントに送信されるときにドキュメントの検索可能なインデックスを構築します。
- Camel でのインデックス化された検索の実行を容易に実行

このコンポーネントはプロデューサーエンドポイントのみをサポートします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-lucene</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

204.1. URI 形式

```
lucene:searcherName:insert[?options]
lucene:searcherName:query[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&...

204.2. オプションの挿入

Lucene コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
config (advanced)	共有された lucene 設定を使用するには、以下を行います。		LuceneConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Lucene エンドポイントは URI 構文を使用して設定します。

```
lucene:host:operation
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

204.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
host	lucene サーバーの URL が **必要**		文字列
operation	挿入またはクエリーなどの **必要な操作です**。		LuceneOperation

204.2.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
Analyzer （プロデューサー）	Analyzer は、テキストを解析する TokenStreams を構築します。これは、テキストからインデックス用語を抽出するポリシーを表します。アナライザーの値は、抽象クラス org.apache.lucene.analysis.Analyzer を拡張する任意のクラスになります。Lucene は、追加設定なしで豊富なアナライザーも提供します。		アナライザー
indexDir (producer)	指定したアナライザーによるドキュメント分析時にインデックスファイルが作成されるファイルシステムディレクトリー		ファイル
maxHits (producer)	検索操作の結果セットを制限する整数値。		int
srcDir (producer)	プロデューサー起動時にインデックスに分析および追加するために使用されるファイルを含むオプションのディレクトリー。		ファイル
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

204.3. キャッシュへのメッセージの送受信

204.3.1. メッセージヘッダー

ヘッダー	説明
QUERY	インデックスで実行される Lucene Query。クエリーにはワイルドカードおよびフレーズが含まれることがあります。
RETURN_LUCENE_DOCS	camel 2.15: ヒット情報を返す際に実際の Lucene ドキュメントを含めるには、このヘッダーを true に設定します。

204.3.2. Lucene プロデューサー

このコンポーネントは、2 つのプロデューサーエンドポイントをサポートします。

insert: insert プロデューサーは、受信エクスチェンジのボディーを分析し、トークン("content")に関連付けることで、検索可能なインデックスを構築します。query: クエリープロデューサーは、事前作成されたインデックスで検索を実行します。クエリーは、検索可能なインデックスを使用してスコアおよび再配置ベースの検索を実行します。クエリーは受信エクスチェンジ経由で送信されます。'QUERY' というヘッダープロパティー名が含まれます。ヘッダープロパティー 'QUERY' の値は Lucene Query です。Lucene クエリーの作成方法に関する詳細は、http://lucene.apache.org/java/3_0_0/queryparsersyntax.html を参照してください。

204.3.3. Lucene プロセッサー

プロデューサーを作成せずに lucene に対してクエリーを実行するための LuceneQueryProcessor と呼ばれるプロセッサーがあります。

204.4. LUCENE の使用状況に関するサンプル

204.4.1. 例 1: Lucene インデックスの作成

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("direct:start").
            to("lucene:whitespaceQuotesIndex:insert?
                analyzer=#whitespaceAnalyzer&indexDir=#whitespace&srcDir=#load_dir").
            to("mock:result");
    }
};
```

204.4.2. 例 2: Camel コンテキストの JNDI レジストリーにプロパティーの読み込み

```
@Override
protected JndiRegistry createRegistry() throws Exception {
  JndiRegistry registry =
        new JndiRegistry(createJndiContext());
  registry.bind("whitespace", new File("./whitespaceIndexDir"));
  registry.bind("load_dir",
        new File("src/test/resources/sources"));
  registry.bind("whitespaceAnalyzer",
        new WhitespaceAnalyzer());
  return registry;
}
...
CamelContext context = new DefaultCamelContext(createRegistry());
```

204.4.3. 例 2: クエリープロデューサーを使用した検索の実行

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("direct:start").
```

```
        setHeader("QUERY", constant("Seinfeld")).
        to("lucene:searchIndex:query?
          analyzer=#whitespaceAnalyzer&indexDir=#whitespace&maxHits=20").
        to("direct:next");

      from("direct:next").process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          Hits hits = exchange.getIn().getBody(Hits.class);
          printResults(hits);
        }

        private void printResults(Hits hits) {
          LOG.debug("Number of hits: " + hits.getNumberOfHits());
          for (int i = 0; i < hits.getNumberOfHits(); i++) {
            LOG.debug("Hit " + i + " Index Location:" + hits.getHit().get(i).getHitLocation());
            LOG.debug("Hit " + i + " Score:" + hits.getHit().get(i).getScore());
            LOG.debug("Hit " + i + " Data:" + hits.getHit().get(i).getData());
          }
        }
      }).to("mock:searchResult");
    }
  };
```

204.4.4. 例 3: クエリプロセッサーを使用した検索の実行

```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
      try {
        from("direct:start").
          setHeader("QUERY", constant("Rodney Dangerfield")).
          process(new LuceneQueryProcessor("target/stdindexDir", analyzer, null, 20)).
          to("direct:next");
      } catch (Exception e) {
        e.printStackTrace();
      }

      from("direct:next").process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          Hits hits = exchange.getIn().getBody(Hits.class);
          printResults(hits);
        }

        private void printResults(Hits hits) {
          LOG.debug("Number of hits: " + hits.getNumberOfHits());
          for (int i = 0; i < hits.getNumberOfHits(); i++) {
            LOG.debug("Hit " + i + " Index Location:" + hits.getHit().get(i).getHitLocation());
            LOG.debug("Hit " + i + " Score:" + hits.getHit().get(i).getScore());
            LOG.debug("Hit " + i + " Data:" + hits.getHit().get(i).getData());
          }
        }
      }).to("mock:searchResult");
    }
  };
```

第205章 LUMBERJACK コンポーネント

Camel バージョン 2.18 から利用可能

Lumberjack コンポーネントは、インスタンスの Filebeat から Lumberjack プロトコルを使用してネットワーク上で送信されるログを取得します。ネットワーク通信は SSL でセキュリティー保護できます。

このコンポーネントは、コンシューマーエンドポイントのみをサポートします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-lumberjack</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

205.1. URI 形式

```
lumberjack:host
lumberjack:host:port
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

205.2. オプション

Lumberjack コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
sslContextParameters (security)	すべてのエンドポイントに使用するデフォルトの SSL 設定を設定します。また、エンドポイントレベルで直接設定することもできます。		SSLContextParameters

Name	説明	デフォルト	Type
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Lumberjack エンドポイントは、URI 構文を使用して設定します。

```
lumberjack:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

205.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
host	Lumberjack をリッスンするために **必要な** ネットワークインターフェース		文字列
port	Lumberjack をリッスンするネットワークポート	5044	int

205.2.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
sslContextParameters (consumer)	SSL 設定		SSLContextParameters
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

205.3. 結果

結果ボディーは Map<String, Object> オブジェクト です。

205.4. LUMBERJACK の使用方法サンプル

205.4.1. 例 1: ログメッセージのストリーミング

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
    from("lumberjack:0.0.0.0").              // Listen on all network interfaces using the default
port
      setBody(simple("${body[message]}")).    // Select only the log message
      to("stream:out");                       // Write it into the output stream
  }
};
```

第206章 LZF の圧縮データフォーマットの調整

Camel バージョン 2.17 から利用可能

LZF データフォーマット は、メッセージ圧縮と圧縮解除形式です。LZF deflate アルゴリズムを使用します。 LZF 圧縮を使用してマーシャリングされたメッセージは、エンドポイントで消費される直前に LZF のデ圧縮を使用してアンマーシャリングできます。圧縮機能は、大規模な XML およびテキストベースのペイロードを扱う場合や、以前に LZF algotithm を使用して開始したメッセージを読み取る場合に非常に便利です。

206.1. オプション

LZF の圧縮データ形式は、以下に示す 2 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
usingParallelCompression	**false**	**ブール値**	複数の処理コアを使用してエンコーディング（圧縮）を有効にします。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

206.2. マーシャリング

この例では、LZF 圧縮形式を使用する圧縮ペイロードに通常のテキスト/XML ペイロードをマーシャリングし、MY_QUEUE という ActiveMQ キューに送信します。

```
from("direct:start").marshal().lzf().to("activemq:queue:MY_QUEUE");
```

206.3. アンマーシャリング

この例では、MY_QUEUE という ActiveMQ キューから LZF ペイロードを元のフォーマットにアンマーシャリングし、これを UnGZippedMessageProcessor に転送します。

```
from("activemq:queue:MY_QUEUE").unmarshal().lzf().process(new
UnCompressedMessageProcessor());
```

206.4. 依存関係

camel ルートでLZF 圧縮を使用するには、このデータ形式を実装する camel-lzf の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加できます。バージョン番号は最新の最新のリリースに置き換えてください（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-lzf</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第207章 メールコンポーネント

Camel バージョン 1.0 で利用可能

メールコンポーネントは、Spring の Mail サポートと基盤の JavaMail システムを介して Email へのアクセスを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mail</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

警告

geronimo メール .jar

アタッチメントのあるメールをポーリングする際に、geronimo メール .jar (v1.6)にバグがあることが発見されました。Content-Type を正しく特定できません。そのため、.jpeg ファイルをメールにアタッチし、ポーリングする場合、Content-Type は image/jpeg としてではなく text/plain として解決されます。そのため、独自の実装を提供し、ファイル名に基づいて正しい Mime タイプを返すことで、org.apache.camel.component.ContentTypeResolver SPI インターフェースを追加し、このバグを修正しました。そのため、ファイル名が jpeg/jpg で終わる場合、イメージ/jpeg を返すことができます。

MailComponent インスタンスまたは MailEndpoint インスタンスにカスタムリゾルバーを設定できます。

ヒント

POP3 または IMAP POP3 にはいくつかの制限があります。可能であれば、エンドユーザーは IMAP を使用することが推奨されます。

INFO: テストに mock-mail を使用します。ユニットテスト にはモックフレームワークを使用できます。これにより、実際のメールサーバーを必要とせずにテストできます。ただし、実際のメールサーバーにメールを送信する必要がある実稼働環境または他の環境に移行する必要がある場合には、モックメールを必ず含めないようにしてください。クラスパスに mock-javamail.jar が存在するだけで、起動してメールの送信を回避することができます。

207.1. URI 形式

メールエンドポイントには、以下の URI 形式のいずれかを使用できます（プロトコル、SMTP、POP3、または IMAP 用）。

```
smtp://[username@]host[:port][?options]
pop3://[username@]host[:port][?options]
imap://[username@]host[:port][?options]
```

mail コンポーネントは、これらのプロトコル（SSL を通じて）のセキュアなバリアントもサポートします。スキームに s を追加して、セキュアなプロトコルを有効にできます。

```
smtps://[username@]host[:port][?options]
pop3s://[username@]host[:port][?options]
imaps://[username@]host[:port][?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

207.2.

Mail コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	メール設定を設定します。		MailConfiguration
contentTypeResolver (advanced)	ファイル添付の Content-Type を決定するリゾルバー。		ContentTypeResolver
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

207.3.

Mail エンドポイントは、URI 構文を使用して設定します。

```
imap:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

207.3.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
host	***必須***。メールサーバーのホスト名		文字列
port	メールサーバーのポート番号		int

207.3.2. クエリーパラメーター（62 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
closeFolder (consumer)	ポーリング後にコンシューマーがフォルダーを終了するかどうか。このオプションを false に設定して disconnect=false を設定し、その後、コンシューマーはポーリング間でフォルダーを開いたままにします。	true	boolean
copyTo (consumer)	メールメッセージの処理後、指定した名前のメールフォルダーにコピーできます。この設定値を上書きするには、キー copyTo を持つヘッダーを使用して、実行時に設定されたフォルダー名にメッセージをコピーできます。		文字列
Delete （コンシューマー）	メッセージの処理後にメッセージを削除します。これは、メールメッセージに DELETED フラグを設定して行います。false の場合、SEEN フラグが代わりに設定されます。Camel 2.10 以降では、キー delete でヘッダーを設定し、メールを削除するかどうかを決定することにより、この設定オプションを上書きできます。	false	boolean
disconnect (consumer)	ポーリング後にコンシューマーが切断されるかどうか。これを有効にすると、Camel が各ポーリングで強制的に接続されます。	false	boolean
handleFailedMessage (consumer)	メールコンシューマーが指定のメールメッセージを取得できない場合、このオプションではコンシューマーのエラーハンドラーによって原因となった例外を処理できます。コンシューマーでブリッジエラーハンドラーを有効にすると、Camel ルーティングエラーハンドラーが代わりに例外を処理できます。デフォルトの動作はコンシューマーによって例外がスローされ、バッチからのメールは Camel によってルーティングできません。	false	boolean
maxMessagesPerPoll (consumer)	ポーリングごとに収集するメッセージの最大数を指定します。デフォルトでは、最大値は設定されません。サーバーの起動時に数千ものファイルをダウンロードしないように 1000 などの制限を設定できます。このオプションを無効にするには、0 または negative の値を設定します。		int
mimeDecodeHeaders (consumer)	このオプションを使用すると、メールヘッダーの透過的な MIME デコードおよびアンロックが可能になります。	false	boolean

Name	説明	デフォルト	Type
peek （コンシューマー）	メールメッセージを処理する前に、javax.mail.Message を peeked とマークします。これは IMAPMessage メッセージタイプのみに適用されます。peek を使用すると、メールはメールサーバーで SEEN としてマークされません。これにより、Camel にエラー処理がある場合は、メールメッセージをロールバックできます。	true	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
skipFailedMessage (consumer)	メールコンシューマーが指定のメールメッセージを取得できない場合、このオプションではメッセージを省略し、次のメールメッセージの取得を許可します。デフォルトの動作はコンシューマーによって例外がスローされ、バッチからのメールは Camel によってルーティングできません。	false	boolean
Unseen (consumer)	参照されていないメールによってのみ制限するかどうか。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
fetchSize (consumer)	ポーリング中に消費するメッセージの最大数を設定します。メールボックスのフォルダーに多くのメッセージが含まれる場合、これを使用するとメールサーバーのオーバーロードを防ぐことができます。デフォルト値の -1 はフェッチサイズがなく、すべてのメッセージが消費されます。値を 0 に設定することは特別なもので、Camel はメッセージを全く消費しません。	-1	int
folderName (consumer)	ポーリングするフォルダー。	INBOX	文字列
mailUidGenerator (consumer)	カスタムロジックを使用してメールメッセージの UUID を生成できるようにするプラグ可能な MailUidGenerator。		MailUidGenerator

Name	説明	デフォルト	Type
mapMailMessage (consumer)	Camel が受信したメールメッセージを Camel の本文/ヘッダーにマッピングするかどうかを指定します。true に設定すると、メールメッセージのボディーが Camel IN メッセージのボディーにマッピングされ、メールヘッダーは IN ヘッダーにマッピングされます。このオプションを false に設定すると、IN メッセージには raw javax.mail.Message が含まれます。exchange.getIn（）.getBody(javax.mail.Message.class)を呼び出すことで、この未加工メッセージを取得できます。	true	boolean
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
postProcessAction (consumer)	通常の処理が終了したら、メールボックスで後続処理を行う MailBoxPostProcessAction を参照します。		MailBoxPostProcess Action
bcc （プロデューサー）	BCC メールアドレスを設定します。複数のメールアドレスはコンマで区切ります。		文字列
cc (producer)	CC のメールアドレスを設定します。複数のメールアドレスはコンマで区切ります。		文字列
from (producer)	from メールアドレス	camel@localhost	文字列
replyTo (producer)	Reply-To 受信者（応答メールの受信側）。複数のメールアドレスはコンマで区切ります。		文字列
Subject (producer)	送信されるメッセージの件名。注記： ヘッダーにサブジェクトを設定することは、このオプションよりも優先されます。		文字列
to （プロデューサー）	To email address を設定します。複数のメールアドレスはコンマで区切ります。		文字列
javaMailSender (producer)	メールの送信にカスタムの org.apache.camel.component.mail.JavaMailSender を使用します。		JavaMailSender

Name	説明	デフォルト	Type
additionalJavaMail Properties (advanced)	その他のすべてのオプションに基づいて設定されるデフォルトのプロパティーを追記/無効化する追加の java メールプロパティーを設定します。これは、特別なオプションを追加する必要があるものの、他のオプションをそのまま維持する必要がある場合に便利です。		プロパティー
alternativeBodyHeader (advanced)	別のメールボディーが含まれる IN メッセージヘッダーへのキーを指定します。たとえば、text/html 形式でメールを送信し、HTML 以外の電子メールクライアントの代替メール本文を提供する場合は、このキーを持つ別のメールボディをヘッダーとして設定します。	Camel MailAlternativeBody	文字列
attachmentsContentTransferEncodingResolver (advanced)	カスタムの 接続先ContentTransferEncodingResolver を使用して添付に使用する content-type-encoding を解決します。		disconnectContent TransferEncoding Resolver
バインディング（詳細）	Camel メッセージからメールメッセージとの間で変換するために使用されるバインディングを設定します。		MailBinding
connectionTimeout (advanced)	接続のタイムアウト（ミリ秒単位）。	30000	int
contentType (advanced)	メールメッセージのコンテンツタイプ。HTML メールには text/html を使用します。	text/plain	文字列
contentTypeResolver (advanced)	ファイル添付の Content-Type を決定するリゾルバー。		ContentTypeResolver
debugMode (advanced)	基礎となるメールフレームワークでデバッグモードを有効にします。SUN Mail フレームワークは、デフォルトでデバッグメッセージを System.out にロギングします。	false	boolean
headerFilterStrategy (advanced)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用してヘッダーをフィルターします。		HeaderFilterStrategy
ignoreUnsupportedCharset (advanced)	メールの送信時に Camel がローカル JVM でサポートされない文字セットを無視するオプション。charset がサポートされない場合は、charset=XXX（XXX はサポートされない文字セットを表します）が、コンテンツタイプから削除され、代わりにプラットフォームのデフォルトに依存します。	false	boolean

Name	説明	デフォルト	Type
ignoreUriScheme (advanced)	メールの送信時に Camel がローカル JVM でサポートされない文字セットを無視するオプション。charset がサポートされない場合は、charset=XXX（XXX はサポートされない文字セットを表します）が、コンテンツタイプから削除され、代わりにプラットフォームのデフォルトに依存します。	false	boolean
セッション （詳細）	camel がすべてのメール対話に使用するメールセッションを指定します。メールセッションが他のリソースによって作成および管理されるシナリオで役に立ちます。これが指定されていない場合、Camel はメールセッションを自動的に作成します。		Session
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
useInlineAttachments (advanced)	配置のインラインまたは添付を使用するかどうか。	false	boolean
idempotentRepository (filter)	同じメールボックスから消費できるプラグ可能なリポジトリー org.apache.camel.spi.IdempotentRepository。デフォルトでは、使用中のリポジトリーはありません。		String>
idempotentRepositoryRemoveOnCommit (filter)	べき等リポジトリーを使用する場合、メールメッセージが正常に処理され、コミットされると、メッセージ ID がべき等リポジトリー（デフォルト）から削除されるか、リポジトリーに保持される必要があります。デフォルトでは、メッセージ ID が一意であることを前提としており、メールメッセージが認識/移動または削除され、再度消費されないようにするため、リポジトリーに値が保存されていません。そのため、メッセージ ID がべき等リポジトリーに保存されていると、値がほとんどありません。ただし、このオプションを使用すると、何らかの理由でメッセージ ID を保存できます。	true	boolean
searchTerm (filter)	javax.mail.search.SearchTerm を参照します。		SearchTerm
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int

Name	説明	デフォルト	Type
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。	60000	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

Name	説明	デフォルト	Type
sortTerm (sort)	メッセージのソート順序。IMAP ではネイティブでしかサポートされません。POP3 を使用する場合や、IMAP サーバーに SORT 機能がない場合、ある程度エミュレートされます。		文字列
dummyTrustManager (security)	すべての証明書を信頼するためにダミーのセキュリティー設定を使用します。開発モードにのみ使用し、実稼働モードにのみ使用してください。	false	boolean
パスワード （セキュリティー）	ログインのパスワード		文字列
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定します。		SSLContextParameters
ユーザー名 （セキュリティー）	ログインのユーザー名		文字列

207.3.3. サンプルエンドポイント

通常、ログインクレデンシャルを使用して URI を指定します（例として SMTP を置き換えます）。

```
smtp://[username@]host[:port][?password=somepwd]
```

または、ユーザー名とパスワードの両方をクエリーオプションとして指定できます。

```
smtp://host[:port]?password=somepwd&username=someuser
```

以下に例を示します。

```
smtp://mycompany.mailserver:30?password=tiger&username=scott
```

207.4. コンポーネント

- *IMAP*
- *IMAPS*

- *POP3s*
- *POP3s*
- *SMTP*
- *SMTPs*

207.4.1. デフォルトのポート

デフォルトのポート番号がサポートされます。ポート番号を省略すると、Camel はプロトコルに基づいて使用するポート番号を決定します。

プロトコル	デフォルトのポート番号
SMTP	25
SMTPS	465
POP3	110
POP3S	995
IMAP	143
IMAPS	993

207.5. SSL サポート

基礎となるメールフレームワークは、SSL サポートを提供します。 必要な Java Mail API 設定オプションを完全に指定して SSL/TLS サポートを設定するか、コンポーネントまたはエンドポイント設定を介して設定済みの SSLContextParameters を提供できます。

207.5.1. JSSE 設定ユーティリティーの使用

Camel 2.10 以降、mail コンポーネントは Camel JSSE 設定ユーティリティーを介して SSL/TLS 設定をサポートします。 このユーティリティーは、エンドポイントおよびコンポーネントレベルで記述し、設定する必要のあるコンポーネント固有のコードの量を大幅に削減します。 以下の例は、mail コンポーネントでユーティリティーを使用する方法を示しています。

エンドポイントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/truststore.jks");
ksp.setPassword("keystorePassword");
TrustManagersParameters tmp = new TrustManagersParameters();
tmp.setKeyStore(ksp);
SSLContextParameters scp = new SSLContextParameters();
scp.setTrustManagers(tmp);
Registry registry = ...
registry.bind("sslContextParameters", scp);
...
from(...)
&amp;nbsp; &amp;nbsp; .to("smtps://smtp.google.com?
username=user@gmail.com&password=password&sslContextParameters=#sslContextParam
eters");
```

エンドポイントの Spring DSL ベースの設定

```
...
<camel:sslContextParameters id="sslContextParameters">
  <camel:trustManagers>
    <camel:keyStore resource="/users/home/server/truststore.jks" password="keystorePassword"/>
  </camel:trustManagers>
</camel:sslContextParameters>...
...
<to uri="smtps://smtp.google.com?
username=user@gmail.com&password=password&sslContextParameters=#sslContextParameters"/
>...
```

207.5.2. JavaMail の直接的な設定

Camel は SUN JavaMail を使用します。これは、よく知られた認証局（デフォルトの JVM 信頼設定）が発行する証明書のみを信頼します。独自の証明書を発行する場合は、CA 証明書を JVM の Java 信頼/キーストアファイルにインポートし、デフォルトの JVM 信頼/キーストアファイルを上書きする必要があります（詳細は、JavaMail の SSLNOTES.txt を参照してください）。

207.6. メールメッセージの内容

Camel はメッセージエクスチェンジの IN ボディーを MimeMessage テキストコンテンツとして使用します。ボディーは String.class に変換されます。

Camel はすべてのエクスチェンジの IN ヘッダーを MimeMessage ヘッダーにコピーします。

MimeMessage のサブジェクトは、IN メッセージのヘッダープロパティーを使用して設定できます。以下のコードは、以下を示しています。

recipients などの他の MimeMessage ヘッダーにも適用されるため、To でヘッダープロパティーを使用できます。

Camel 2.11 以降、MailProducer を使用する場合は、Camel メッセージヘッダーから CamelMailMessage のキーを持つ MimeMessage のメッセージ ID を 取得できるはずです。

207.7. ヘッダーが事前に設定された受信者よりも優先されます。

メッセージヘッダーで指定された受信側は、エンドポイント URI で事前設定された受信者よりも優先されます。メッセージヘッダーに受信側を提供する場合は、取得するものになります。エンドポイント URI で事前設定された受信側はフォールバックとして処理されます。

以下のサンプルコードでは、メールメッセージは事前設定された受信者 davsclaus@apache.org より優先されるため、info@mycompany.com に送信されます。エンドポイント URI の CC および BCC 設定も無視され、これらの受信者はメールを受信しません。ヘッダーと事前設定の設定の中から選択肢は、すべてまたは何もありません。mail コンポーネントは、ヘッダーからのみ受信者を取る か、事前に設定された設定から排他的に受け取ります。ヘッダーと事前設定済みの設定を混在させたり、一致させることはできません。

```
        Map<String, Object> headers = new HashMap<String, Object>();
        headers.put("to", "davsclaus@apache.org");

        template.sendBodyAndHeaders("smtp://admin@localhost?to=info@mycompany.com",
    "Hello World", headers);
```

207.8. 設定を容易にする複数の受信者

コンマ区切りのリストまたはセミコロン区切りのリストを使用して、複数の受信者を設定できます。これは、ヘッダー設定とエンドポイント URI の設定の両方に適用されます。以下に例を示します。

```
        Map<String, Object> headers = new HashMap<String, Object>();
        headers.put("to", "davsclaus@apache.org ; jstrachan@apache.org ;
    ningjiang@apache.org");
```

上記の例では、セミコロン ; を区切り文字として使用します。

207.9. 送信者名とメールの設定

受信者は、< email> という形式 で指定し、受信者の名前とメールアドレスの両方を含めることができます。

たとえば、以下のヘッダーをメッセージに定義します。

```
Map headers = new HashMap();
map.put("To", "Claus Ibsen <davsclaus@apache.org>");
map.put("From", "James Strachan <jstrachan@apache.org>");
map.put("Subject", "Camel is cool");
```

207.10. JAVAMAIL API（SUN JAVAMAIL など）

JavaMail API は、メールの消費と生成の可能性下で使用されます。
POP3 または IMAP プロトコルの使用時にエンドユーザーがこれらの参照を参照することが推奨されます。特に、POP3 の機能セットは IMAP よりも限られています。

- JavaMail POP3 API
- JavaMail IMAP API
- 一般的には、MAIL フラグについて

207.11. サンプル

最初に、JMS キューから受信したメッセージをメールとして送信する単純なルートから開始します。メールアカウントは、mymailserver.com の管理 者 アカウントです。

```
from("jms://queue:subscription").to("smtp://admin@mymailserver.com?password=secret");
```

次の例では、1 分ごとに新しい電子メールのメールボックスをポーリングします。ポーリングの間隔 consumer.delay を 60000 ミリ秒 = 60 秒として、特別なコンシューマーオプションを使用することに

注意してください。

```
from("imap://admin@mymailserver.com
    password=secret&unseen=true&consumer.delay=60000")
  .to("seda://mails");
```

この例では、複数の受信者にメールを送信します。

207.12. 添付サンプルを使用したメールの送信

警告

アタッチメントは Camel コンポーネントすべてによってサポートされません。接続 API は Java Activation Framework をベースとしており、通常は Mail API によってのみ使用されます。他の Camel コンポーネントの多くはアタッチメントをサポートしないため、添付ファイルはルート全体で伝播されるため、失われる可能性があります。そのため、thumb のルールは、mail エンドポイントにメッセージを送信する直前に添付を追加します。

メールコンポーネントは添付をサポートします。以下の例では、ロゴファイル添付を含むプレーンテキストのメッセージが含まれるメールメッセージを送信します。

207.13. SSL の例

この例では、メールの Google メール受信トレイをポーリングします。メールをローカルメールクライアントにダウンロードするには、Google メールで SSL を有効にして設定する必要があります。これは、Google メールアカウントにログインし、IMAP アクセスを許可するように設定を変更します。Google には、これを行うための幅広いドキュメントがあります。

```
from("imaps://imap.gmail.com?
username=YOUR_USERNAME@gmail.com&password=YOUR_PASSWORD"
    + "&delete=false&unseen=true&consumer.delay=60000").to("log:newmail");
```

前述のルートは、新しいメールを 1 分ごとに新しいメールで Google メール受信トレイにポーリングし、受信したメッセージを 新しいメール ロガーカテゴリーに記録します。
DEBUG ロギングを有効にしてサンプルを実行すると、ログで進捗を監視できます。

```
2008-05-08 06:32:09,640 DEBUG MailConsumer - Connecting to MailStore
imaps//imap.gmail.com:993 (SSL enabled), folder=INBOX
2008-05-08 06:32:11,203 DEBUG MailConsumer - Polling mailfolder:
imaps//imap.gmail.com:993 (SSL enabled), folder=INBOX
2008-05-08 06:32:11,640 DEBUG MailConsumer - Fetching 1 messages. Total 1 messages.
2008-05-08 06:32:12,171 DEBUG MailConsumer - Processing message: messageNumber=
[332], from=[James Bond <007@mi5.co.uk>], to=YOUR_USERNAME@gmail.com], subject=[...
2008-05-08 06:32:12,187 INFO  newmail - Exchange[MailMessage: messageNumber=[332],
from=[James Bond <007@mi5.co.uk>], to=YOUR_USERNAME@gmail.com], subject=[...
```

207.14. 添付サンプルでのメールの消費

この例では、メールボックスをポーリングし、メールからすべての添付ファイルをファイルとして格納します。まず、メールボックスをポーリングするルートを定義します。このサンプルは google メールをベースとしているため、SSL サンプルと同じルートを使用します。

```
from("imaps://imap.gmail.com?
username=YOUR_USERNAME@gmail.com&password=YOUR_PASSWORD"
    + "&delete=false&unseen=true&consumer.delay=60000").process(new MyMailProcessor());
```

メールをロギングするのではなく、java コードからメールを処理できるプロセッサーを使用します。

```
public void process(Exchange exchange) throws Exception {
    // the API is a bit clunky so we need to loop
    Map<String, DataHandler> attachments = exchange.getIn().getAttachments();
    if (attachments.size() > 0) {
        for (String name : attachments.keySet()) {
            DataHandler dh = attachments.get(name);
            // get the file name
            String filename = dh.getName();

            // get the content and convert it to byte[]
            byte[] data = exchange.getContext().getTypeConverter()
                            .convertTo(byte[].class, dh.getInputStream());

            // write the data to a file
            FileOutputStream out = new FileOutputStream(filename);
            out.write(data);
            out.flush();
            out.close();
        }
    }
}
```

アタッチメントを処理する API はビットの明確ですが、javax.activation.DataHandler を取得できるため、標準 API を使用して添付ファイルを処理できるようになります。

207.15. アタッチメントでメールメッセージを分割する方法

この例では、多くの添付ファイルがあるメールメッセージを消費します。後は、添付ファイルを個別に処理するために、個別の添付ファイルごとに Splitter EIP を使用します。たとえば、メールメッセージに 5 つの添付ファイルがある場合、Splitter が 5 つのメッセージを処理するようにし、それぞれが 1 つの添付ファイルを持つようにします。これを行うには、Splitter にカスタム式を提供する必要があります。ここで、1 つの添付ファイルを持つ 5 つのメッセージが含まれる List<Message> を提供します。

このコードは Camel 2.10 以降の camel-mail コンポーネントで追加設定なしで提供されます。コードは org.apache.camel.component.mail.SplitAttachmentsExpression クラスにあり、ここにソースコードを見つけることができます。

Camel ルートでは、以下のようにルートでこの式を使用する必要があります。

XML DSL を使用する場合、以下のように Splitter でメソッド呼び出し式を宣言する必要があります。

```
<split>
  <method beanType="org.apache.camel.component.mail.SplitAttachmentsExpression"/>
  <to uri="mock:split"/>
</split>
```

Camel 2.16 以降では、添付ファイルを byte[] として分割し、メッセージボディーとして保存することもできます。これは、ブール値 true の式を作成して行います。

```
SplitAttachmentsExpression split = SplitAttachmentsExpression(true);
```

そして、Splitter eip で式を使用します。

207.16. カスタム検索の使用

Camel 2.11 から利用可能

MailEndpoint で searchTerm を設定できます。これにより、不要なメールを除外できます。

たとえば、Subject または Text のいずれかに Camel が含まれるメールをフィルターするには、以下のように行います。

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm.subjectOrBody=Camel"/>
  <to uri="bean:myBean"/>
</route>
```

"searchTerm.subjectOrBody" をパラメーターキーとして使用し、「Camel」という単語を含めるようメールサブジェクトまたはボディーで検索したいことを示すパラメーターキーであることに注意してください。
org.apache.camel.component.mail.SimpleSearchTerm クラスには、設定可能な数多くのオプションがあります。

または、24 時間遡った新しい未参照メールを取得するには、実施することができます。「now-24h」構文に注意してください。詳細は以下の表を参照してください。

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm.fromSentDate=now-24h"/>
  <to uri="bean:myBean"/>
</route>
```

エンドポイント URI 設定に複数の searchTerm を指定できます。その後、AND 演算子を使用して組み合わせるので、両方の条件が一致する必要があります。たとえば、メールの件名に Camel がある 24 時間返される最後の未確認メールを取得するには、以下を行うことができます。

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm.subject=Camel&searchTerm.fromSentDate=now-
24h"/>
  <to uri="bean:myBean"/>
</route>
```

SimpleSearchTerm は POJO から簡単に設定可能なように設計されているため、XML の <bean> スタイルを使用して設定することもできます。

```
<bean id="mySearchTerm" class="org.apache.camel.component.mail.SimpleSearchTerm">
  <property name="subject" value="Order"/>
  <property name="to" value="acme-order@acme.com"/>
  <property name="fromSentDate" value="now"/>
 </bean>
```

その後、以下のように Camel ルートの #beanId を使用してこの Bean を参照できます。

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm=#mySearchTerm"/>
  <to uri="bean:myBean"/>
</route>
```

Java には、org.apache.camel.component.mail.SearchTermBuilder クラスを使用して compound SearchTerms を構築するビルダークラスがあります。これにより、以下のような複雑な用語を構築できます。

```
// we just want the unseen mails which is not spam
SearchTermBuilder builder = new SearchTermBuilder();

builder.unseen().body(Op.not, "Spam").subject(Op.not, "Spam")
  // which was sent from either foo or bar
  .from("foo@somewhere.com").from(Op.or, "bar@somewhere.com");
  // .. and we could continue building the terms

SearchTerm term = builder.build();
```

207.17. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第208章 マスターコンポーネント

Camel バージョン 2.20 で利用可能

camel-master: エンドポイントは、クラスター内の単一のコンシューマーのみを指定のエンドポイントから消費させる方法を提供します。その JVM が停止している場合には、自動フェイルオーバーを使用します。

これは、同時消費をサポートしない一部のレガシーバックエンドから消費する必要がある場合や、商用または安定性の理由上、ある時点では単一の接続のみを持つことができる場合に非常に便利です。

208.1. マスターエンドポイントの使用

camel エンドポイントの前に **master:someName:** を付けます。ここで、**someName** は論理名で、マスターロックの取得に使用されます。

```
from("master:cheese:jms:foo").to("activemq:wine");
```

上記は ActiveMQ の [Exclusive Consumers](http://activemq.apache.org/exclusive-consumer.html)タイプ機能をシミュレートしますが、排他コンシューマーをサポートしない可能性があるサードパーティーの JMS プロバイダーでシミュレートします。

208.2. URI 形式

```
master:namespace:endpoint[?options]
```

ここでの **endpoint** は、マスター/スレーブモードで実行する Camel エンドポイントです。

208.3. オプション

Master コンポーネントは、以下に示す **3** つのオプションをサポートします。

Name	説明	デフォルト	Type
サービス （詳細）	使用するサービスを注入します。		CamelClusterService

Name	説明	デフォルト	Type
serviceSelector (advanced)	使用する CamelClusterService を検索するために使用されるサービスセレクターを挿入します。		セレクター
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

マスターエンドポイントは URI 構文を使用して設定します。

```
master:namespace:delegateUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

208.3.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
namespace	**必要な** クラスター namespace の名前		文字列
delegateUri	マスター/スレーブモードで使用するエンドポイント URI が **必要**		文字列

208.3.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

208.4. 例

クラスター化された Camel アプリケーションを保護することで、1 つのアクティブなノードからのみファイルを消費できます。

```
// the file endpoint we want to consume from
String url = "file:target/inbox?delete=true";

// use the camel master component in the clustered group named myGroup
// to run a master/slave mode in the following Camel url
from("master:myGroup:" + url)
    .log(name + " - Received file: ${file:name}")
    .delay(delay)
    .log(name + " - Done file:     ${file:name}")
    .to("file:target/outbox");
```

マスターコンポーネントは、以下を使用して設定できる CamelClusterService を活用します。

- *Java*

```
ZooKeeperClusterService service = new ZooKeeperClusterService();
service.setId("camel-node-1");
service.setNodes("myzk:2181");
service.setBasePath("/camel/cluster");

context.addService(service)
```

- ***XML(Spring/Blueprint)***

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://camel.apache.org/schema/spring
    http://camel.apache.org/schema/spring/camel-spring.xsd">


  <bean id="cluster"
class="org.apache.camel.component.zookeeper.cluster.ZooKeeperClusterService">
    <property name="id" value="camel-node-1"/>
    <property name="basePath" value="/camel/cluster"/>
    <property name="nodes" value="myzk:2181"/>
  </bean>

  <camelContext xmlns="http://camel.apache.org/schema/spring" autoStartup="false">
    ...
  </camelContext>

</beans>
```

- ***Spring Boot***

```
camel.component.zookeeper.cluster.service.enabled   = true
camel.component.zookeeper.cluster.service.id        = camel-node-1
camel.component.zookeeper.cluster.service.base-path = /camel/cluster
camel.component.zookeeper.cluster.service.nodes     = myzk:2181
```

208.5. 実装

Camel は以下の ClusterService 実装を提供します。

- ***camel-atomix***
- ***camel-consul***
- ***camel-file***

- *camel-kubernetes*
- *camel-zookeeper*

208.6. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第209章 メトリクスコンポーネント

209.1. メトリクスコンポーネント

metrics: コンポーネントを使用すると、Camel ルートから直接各種のメトリクスを収集できます。サポートされるメトリックタイプは、 カウンター、 ヒストグラム、メーター、 および ゲージ です。 メトリクス を使用すると、アプリケーションの動作を簡単に測定できます。設定可能なレポートバックエンドは、統計を収集し、可視化するためのさまざまな統合オプションを有効にします。また、コンポーネントは Dropwizard Metrics を使用してルート統計を公開できる MetricsRoutePolicyFactory を提供します。詳細は、ページ下部 を参照してください。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-metrics</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

209.2. URI 形式

```
metrics:[ meter | counter | histogram | timer | gauge ]:metricname[?options]
```

209.3. オプション

Metrics コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
metricRegistry (advanced)	カスタムが設定された MetricRegistry を使用するには、以下を実行します。		MetricRegistry
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

メトリクスエンドポイントは URI 構文を使用して設定します。

```
metrics:metricsType:metricsName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

209.3.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
metricsType	**必要な** メトリクスのタイプ		MetricsType
metricsName	メトリクスの **必須** 名		文字列

209.3.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
アクション （プロデューサー）	タイマータイプを使用する際のアクション		MetricsTimerAction
decrement (producer)	カウンタータイプを使用する場合のデクリメント値		Long
インクリメント （プロデューサー）	カウンタータイプを使用する場合の値のインクリメント		Long
マーク （プロデューサー）	メータータイプを使用するときのマーク		Long
Subject (producer)	ゲージタイプを使用する際のサブジェクト値		オブジェクト
値 （プロデューサー）	ヒストグラムタイプを使用する際の値		Long
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

209.4. METRIC REGISTRY

Camel Metrics コンポーネントは、デフォルトで 60 秒のレポーティング間隔を持つ Slf4jReporter を持つ MetricRegistry インスタンスを使用します。このデフォルトレジストリーは、MetricRegistry

Bean を指定してカスタムのレジストリーに置き換えることができます。複数の MetricRegistry Bean がアプリケーションに存在する場合は、metricRegistry という名前の Bean が使用されます。

Spring Java 設定の使用例：

```
@Configuration
public static class MyConfig extends SingleRouteCamelConfiguration {

    @Bean
    @Override
    public RouteBuilder route() {
        return new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                // define Camel routes here
            }
        };
    }

    @Bean(name = MetricsComponent.METRIC_REGISTRY_NAME)
    public MetricRegistry getMetricRegistry() {
        MetricRegistry registry = ...;
        return registry;
    }
}
```

または、CDI を使用します。

```
class MyBean extends RouteBuilder {

    @Override
    public void configure() {
      from("...")
          // Register the 'my-meter' meter in the MetricRegistry below
          .to("metrics:meter:my-meter");
    }

    @Produces
    // If multiple MetricRegistry beans
    // @Named(MetricsComponent.METRIC_REGISTRY_NAME)
    MetricRegistry registry() {
        MetricRegistry registry = new MetricRegistry();
        // ...
        return registry;
    }
}
```

注意

MetricRegistry は、レポートに内部スレッドを使用します。ユーザーが終了時にクリーンアップするには、バージョン DropWizard 3.0.1 にはパブリック API はありません。そのため、Camel Metrics コンポーネントを使用すると Java クラスローダーのリークが発生し、場合によっては OutOfMemoryErrors が発生します。

209.5. 用途

各メトリクスには type と name があります。サポートされるタイプは、カウンター、ヒストグラム、メーター、タイマー、ゲージ です。メトリクス名は単純な文字列です。メトリクスタイプが指定されていない場合、タイプメーターはデフォルトで使用されます。

209.5.1. Headers

URI で定義されたメトリクス名は、CamelMetricsName という名前のヘッダーを使用して上書きできます。

たとえば、以下のようになります。

```
from("direct:in")
    .setHeader(MetricsConstants.HEADER_METRIC_NAME, constant("new.name"))
    .to("metrics:counter:name.not.used")
    .to("direct:out");
```

name.not.used ではなく new.name でカウンターを更新します。

メトリクスエンドポイントがエクスチェンジの処理を終了すると、メトリクス固有のヘッダーはすべてメッセージから削除されます。エクスチェンジメトリクスエンドポイントを処理すると、レベル warn を使用してすべての例外を取得し、ログエントリーを書き込みます。

209.6. メトリクスタイプカウンター

```
metrics:counter:metricname[?options]
```

209.6.1. オプション

Name	デフォルト	説明
インクリメント	-	カウンターに追加する長い値
decrement	-	カウンターから減算する long 値

インクリメント または デクリメント が定義されていない場合は、カウンター値は 1 つずつ増えます。インクリメントと デクリメント の両方がインクリメント 操作のみが定義されている場合。

```
// update counter simple.counter by 7
from("direct:in")
    .to("metric:counter:simple.counter?increment=7")
    .to("direct:out");
```

```
// increment counter simple.counter by 1
from("direct:in")
    .to("metric:counter:simple.counter")
    .to("direct:out");
```

```
// decrement counter simple.counter by 3
from("direct:in")
    .to("metric:counter:simple.counter?decrement=3")
    .to("direct:out");
```

209.6.2. Headers

メッセージヘッダーを使用して、Metrics コンポーネント URI に指定された インクリメント 値および デクリメント 値を上書きできます。

Name	説明	想定されるタイプ
Camel Metrics CounterIncrement	URI のインクリメント値の上書き	Long

Name	説明	想定されるタイプ
Camel Metrics CounterDecrement	URI のデクリメント値の上書き	Long

```
// update counter simple.counter by 417
from("direct:in")
    .setHeader(MetricsConstants.HEADER_COUNTER_INCREMENT, constant(417L))
    .to("metric:counter:simple.counter?increment=7")
    .to("direct:out");
```

```
// updates counter using simple language to evaluate body.length
from("direct:in")
    .setHeader(MetricsConstants.HEADER_COUNTER_INCREMENT, simple("${body.length}"))
    .to("metrics:counter:body.length")
    .to("mock:out");
```

209.7. METRIC TYPE HISTOGRAM

```
metrics:histogram:metricname[?options]
```

209.7.1. オプション

Name	デフォルト	説明
value	-	ヒストグラムで使用する値

値 が設定されていないと、ヒストグラムには何も追加されず、警告がログに記録されます。

```
// adds value 9923 to simple.histogram
from("direct:in")
    .to("metric:histogram:simple.histogram?value=9923")
    .to("direct:out");
```

```
// nothing is added to simple.histogram; warning is logged
from("direct:in")
    .to("metric:histogram:simple.histogram")
    .to("direct:out");
```

209.7.2. Headers

メッセージヘッダーは、Metrics コンポーネント URI で指定された値を上書きするために使用できます。

Name	説明	想定されるタイプ
Camel Metrics HistogramValue	URI のヒストグラム値の上書き	Long

```
// adds value 992 to simple.histogram
from("direct:in")
    .setHeader(MetricsConstants.HEADER_HISTOGRAM_VALUE, constant(992L))
    .to("metric:histogram:simple.histogram?value=700")
    .to("direct:out")
```

209.8. METRIC TYPE METER

```
metrics:meter:metricname[?options]
```

209.8.1. オプション

Name	デフォルト	説明
mark	-	マークとして使用する長い値

mark が設定されていない場合、 メーター.mark（） は引数なしで呼び出されます。

```
// marks simple.meter without value
from("direct:in")
    .to("metric:simple.meter")
    .to("direct:out");
```

```
// marks simple.meter with value 81
from("direct:in")
    .to("metric:meter:simple.meter?mark=81")
    .to("direct:out");
```

209.8.2. Headers

メッセージヘッダーを使用して、Metrics コンポーネント URI で指定された マーク 値を上書きできます。

Name	説明	想定されるタイプ
Camel Metrics Meter Mark	URI でマーク値を上書きする	Long

```
// updates meter simple.meter with value 345
from("direct:in")
    .setHeader(MetricsConstants.HEADER_METER_MARK, constant(345L))
    .to("metric:meter:simple.meter?mark=123")
    .to("direct:out");
```

209.9. メトリクスタイプのタイマー

```
metrics:timer:metricname[?options]
```

209.9.1. オプション

Name	デフォルト	説明
action	-	起動または停止

アクション または無効な値が指定されていない場合、警告はタイマーの更新なしでログに記録されます。アクション 開始 が実行中のタイマーまたは停止で呼び出されたときにタイマーまたは 停止 が呼び出されていない場合は、何も更新されず、警告がログに記録されます。

```
// measure time taken by route "calculate"
from("direct:in")
    .to("metrics:timer:simple.timer?action=start")
    .to("direct:calculate")
    .to("metrics:timer:simple.timer?action=stop");
```

TimerContext オブジェクトは、異なるメトリクスコンポーネント呼び出し間でエクスチェンジプロパティーとして保存されます。

209.9.2. Headers

メッセージヘッダーは、Metrics コンポーネント URI で指定されたアクション値を上書きするために使用できます。

Name	説明	想定されるタイプ
Camel Metrics TimerAction	URI でのタイマーアクションの上書き	**org.apache.camel.component.metrics.timer.TimerEndpoint.TimerAction**

```
// sets timer action using header
from("direct:in")
    .setHeader(MetricsConstants.HEADER_TIMER_ACTION, TimerAction.start)
    .to("metric:timer:simple.timer")
    .to("direct:out");
```

209.10. METRIC TYPE GAUGE

```
metrics:gauge:metricname[?options]
```

209.10.1. オプション

Name	デフォルト	説明
subject	-	ゲージによって観察されるオブジェクト

サブジェクト が定義されていない場合は単に無視します。つまり、ゲージは登録されません。

```
// update gauge "simple.gauge" by a bean "mySubjectBean"
from("direct:in")
```

```
    .to("metric:gauge:simple.gauge?subject=#mySubjectBean")
    .to("direct:out");
```

209.10.2. Headers

メッセージヘッダーを使用して、Metrics コンポーネント URI で指定された サブジェクト 値を上書きできます。注記： CamelMetricsName ヘッダーが指定されている場合には、URI で指定されるデフォルトのゲージに加えて、新しいゲージが登録されます。

Name	説明	想定されるタイプ
Camel Metrics Gauge Subject	URI のサブジェクト値を上書きする	オブジェクト

```
// update gauge simple.gauge by a String literal "myUpdatedSubject"
from("direct:in")
    .setHeader(MetricsConstants.HEADER_GAUGE_SUBJECT, constant("myUpdatedSubject"))
    .to("metric:counter:simple.gauge?subject=#mySubjectBean")
    .to("direct:out");
```

209.11. METRICSROUTEPOLICYFACTORY

このファクトリーでは、Dropwizard メトリクスを使用してルート使用状況の統計を公開する各ルートに RoutePolicy を追加できます。このファクトリーは、以下の例のように Java および XML で使用できます。

注記

MetricsRoutePolicyFactory を使用する代わりに、インストルメント化するルートごとに MetricsRoutePolicy を定義できます。この場合、一部の選択したルートのみをインストルメント化できます。

Java の場合は、以下のようにファクトリーを CamelContext に追加します。

```
context.addRoutePolicyFactory(new MetricsRoutePolicyFactory());
```

XML DSL の場合、以下のように <bean> を定義します。

```
<!-- use camel-metrics route policy to gather metrics for all routes -->
<bean id="metricsRoutePolicyFactory"
class="org.apache.camel.component.metrics.routepolicy.MetricsRoutePolicyFactory"/>
```

MetricsRoutePolicyFactory および MetricsRoutePolicy は以下のオプションをサポートします。

Name	デフォルト	説明
useJmx	false	**com.codahale.metrics.JmxReporter** を使用して詳細な統計を JMX に報告するかどうか。 CamelContext で JMX が有効になっている場合、JMX ツリーのサービスタイプの下に **MetricsRegistryService** mbean が登録されていることに注意してください。この mbean には、統計を JSON 出力する 1 つのオペレーションがあります。**useJmx** を true に設定する必要があるのは、統計タイプごとに詳細な mbeans が必要な場合のみです。
jmxDomain	org.apache.camel.metrics	JMX ドメイン名
prettyPrint	false	統計を json 形式で出力する際に Pretty print を使用するかどうか。
metricsRegistry		共有 **com.codahale.metrics.MetricRegistry** の使用を許可します。指定しない場合は、Camel はこの CamelContext によって使用される共有インスタンスを作成します。
rateUnit	TimeUnit.SECONDS	メトリクスレポーターまたは統計を json 出力するときのレートに使用する単位。
durationUnit	TimeUnit.MILLISECONDS	メトリクスレポーターまたは統計を json 出力するときの期間に使用する単位。
namePattern	**name.routeId.type**	Camel 2.17: 使用する名前パターン。区切り文字としてドットを使用しますが、変更できます。値 **名**、**routeId**、および **type** は実際の値に置き換えられます。**name** は、CamelContext の名前です。**routeId** はルートの名前です。and **type** は応答の値です。

Java コード tou から、以下のように org.apache.camel.component.metrics.routepolicy.MetricsRegistryService から com.codahale.metrics.MetricRegistry を保持することができます。

```
MetricRegistryService registryService = context.hasService(MetricsRegistryService.class);
if (registryService != null) {
  MetricsRegistry registry = registryService.getMetricsRegistry();
  ...
}
```

209.12. METRICSMESSAGEHISTORYFACTORY

Camel 2.17 から利用可能

このファクトリーでは、メッセージのルーティング中にメトリクスを使用して Message History パフォーマンス統計を取得できます。これは、すべてのルートの各ノードにメトリクス Timer を使用して機能します。 このファクトリーは、以下の例のように Java および XML で使用できます。

Java では、以下のようにファクトリーを CamelContext に設定します。

```
context.setMessageHistoryFactory(new MetricsMessageHistoryFactory());
```

XML DSL の場合、以下のように <bean> を定義します。

```
<!-- use camel-metrics message history to gather metrics for all messages being routed -->
  <bean id="metricsMessageHistoryFactory"
class="org.apache.camel.component.metrics.messagehistory.MetricsMessageHistoryFactory"/>
```

ファクトリーでは以下のオプションがサポートされます。

Name	デフォルト	説明
useJmx	false	**com.codahale.metrics.JmxReporter** を使用して詳細な統計を JMX に報告するかどうか。 CamelContext で JMX が有効になっている場合、JMX ツリーのサービスタイプの下に **MetricsRegistryService** mbean が登録されていることに注意してください。この mbean には、統計を JSON 出力する 1 つのオペレーションがあります。**useJmx** を true に設定する必要があるのは、統計タイプごとに詳細な mbeans が必要な場合のみです。

Name	デフォルト	説明
jmxDomain	org.apache.camel.metrics	JMX ドメイン名
prettyPrint	false	統計を json 形式で出力する際に Pretty print を使用するかどうか。
metricsRegistry		共有 **com.codahale.metrics.MetricRegistry** の使用を許可します。指定しない場合は、Camel はこの CamelContext によって使用される共有インスタンスを作成します。
rateUnit	TimeUnit.SECONDS	メトリクスレポーターまたは統計を json 出力するときのレートに使用する単位。
durationUnit	TimeUnit.MILLISECONDS	メトリクスレポーターまたは統計を json 出力するときの期間に使用する単位。
namePattern	**name.routeId.id.type**	使用する名前パターン。区切り文字としてドットを使用しますが、変更できます。値 **名**、**routeId**、**type**、および **id** は実際の値に置き換えられます。**name** は、CamelContext の名前です。**routeId** はルートの名前です。**id** パターンはノード ID を表します。and **type** は履歴の値です。

実行時にメトリクスには Java API または JMX からアクセスでき、データを json 出力として収集できます。

Java コードから、以下のように CamelContext からサービスを取得できます。

```
MetricsMessageHistoryService service =
context.hasService(MetricsMessageHistoryService.class);
String json = service.dumpStatisticsAsJson();
```

また、JMX API では、Bean は name=MetricsMessageHistoryService の type=services ツリーに登録されます。

209.13. INSTRUMENTEDTHREADPOOLFACTORY

Camel 2.18 から利用可能

このファクトリーを使用すると、Camel 内から情報を収集する InstrumentedThreadPoolFactory を注入することで、Camel スレッドプールのパフォーマンス情報を収集できます。詳細は「Spring を使用した CamelContext の高度な設定」を参照してください。

209.14. 関連項目

- *Camel、Metrics、CDI 間のインテグレーションを示す camel-example-cdi-metrics の例。*

第210章 OPC UA クライアントコンポーネント

Camel バージョン 2.19 から利用可能

Milo Client コンポーネントは、Eclipse Milo™ 実装を使用して OPC UA サーバーへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-milo</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

OPC UA クライアントコンポーネントは、以下に示す 6 つのオプションをサポートします。

Name	説明	デフォルト	Type
defaultConfiguration (common)	クライアントのデフォルトオプションすべて		MiloClientConfiguration
applicationName (common)	デフォルトのアプリケーション名		文字列
applicationUri (common)	デフォルトのアプリケーション URI		文字列
productUri (common)	デフォルトのプロダクト URI		文字列
reconnectTimeout (common)	デフォルトの再接続タイムアウト		Long
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

210.1. URI 形式

エンドポイントの URI 構文は以下のようになります。

```
milo-client:tcp://[user:password@]host:port/path/to/service?node=RAW(nsu=urn:foo:bar;s=item-1)
```

サーバーがパスを使用しない場合は、単に省略することができます。

```
milo-client:tcp://[user:password@]host:port?node=RAW(nsu=urn:foo:bar;s=item-1)
```

ユーザー認証情報が指定されていない場合、クライアントは匿名モードに切り替わります。

210.2. URI オプション

グループクライアント内のすべての設定オプションは、共有クライアントインスタンスに適用されます。エンドポイントは、エンドポイント URI ごとにクライアントインスタンスを共有します。そのため、そのエンドポイント URI のリクエストが最初に行われると、クライアントグループのオプションが適用されます。これ以降のインスタンスはすべて無視されます。

同じエンドポイント URI に代替オプションが必要な場合、別の共有接続インスタンスを選択するためにエンドポイント URI に内部に追加される clientId オプションを設定することは可能です。つまり、エンドポイント URI とクライアント ID の組み合わせによって位置する共有接続です。

OPC UA クライアントエンドポイントは、URI 構文を使用して設定します。

```
milo-client:endpointUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

210.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
endpointUri	**必須**: OPC UA サーバーエンドポイント		文字列

210.2.2. クエリーパラメーター（24 パラメーター）：

Name	説明	デフォルト	Type
clientId (common)	新しい接続インスタンスの作成を強制する仮想クライアント ID		文字列
defaultAwaitWrites (common)	書き込みのデフォルト await 設定	false	boolean
ノード (common)	ノード定義（ノード ID を参照）		ExpandedNodeId
samplingInterval (common)	サンプリング間隔（ミリ秒単位）		double
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
allowedSecurityPolicies (client)	許可されるセキュリティーポリシー URI のセット。デフォルトではすべてを受け入れ、最も高いものを使用します。		文字列
applicationName (client)	アプリケーション名	Eclipse Milo 用の Apache Camel アダプター	文字列

Name	説明	デフォルト	Type
applicationUri (client)	アプリケーション URI	http://camel.apache.org/EclipseMilo/Client	文字列
channelLifetime (client)	チャネルの有効期間（ミリ秒単位）		Long
keyAlias (client)	キーストアファイルのキーの名前		文字列
keyPassword (client)	鍵のパスワード		文字列
keyStorePassword (client)	キーストアパスワード		文字列
keyStoreType (client)	キーストアのタイプ		文字列
keyStoreUrl (client)	キーの読み込み元の URL		URL
maxPendingPublishRequests (client)	保留中の公開要求の最大数		Long
maxResponseMessageSize (client)	レスポンスメッセージが持つことができる最大バイト数		Long
overrideHost (client)	サーバーが報告されたエンドポイントホストを、エンドポイント URI のホストで上書きします。	false	boolean
productUri (client)	製品 URI	http://camel.apache.org/EclipseMilo	文字列
requestTimeout (client)	要求タイムアウト（ミリ秒単位）		Long

Name	説明	デフォルト	Type
sessionName (client)	セッション名		文字列
sessionTimeout (client)	セッションタイムアウト（ミリ秒単位）		Long

210.2.3. ノード ID

ターゲットノードを定義するには、namespace とノード ID が必要です。以前のバージョンでは、nodeId および namespaceUri または namespaceIndex を指定することにより、これができました。ただし、これは文字列ベースのノード ID を使用する場合にのみ許可されます。また、この設定はまだ可能ですが、新しい設定が推奨されます。

新しいアプローチは、ns=1;i =1 の形式で完全な namespace+node ID を指定し、その他のノード ID 形式（数値、GUID/UUID、不透明な形式など）を使用することも許可しています。ノード パラメーターを使用する場合は、古いパラメーターを使用しないでください。このノードフォーマットの構文は、セミコロン(;)で区切られた キーと値 のペアのセットです。

namespace と 1 つのノード ID キーのみを使用する必要があります。可能なキーについては、以下の表を参照してください。

キー	Type	説明
ns	namespace	数値の namespace インデックス
nsu	namespace	名前空間 URI
s	node	string node ID
i	node	数値ノード ID
g	node	GUID/UUID ノード ID
b	node	不透明なノード ID の base64 でエンコードされた文字列

構文によって生成された値は URI パラメーターの値に透過的にエンコードできないため、それらをエスケープする必要があります。しかし、Camel では実際の値を RAW(...)内でラップすることができます。これにより、エスケープが不必要になります。以下に例を示します。

```
milo-client://user:password@localhost:12345?node=RAW(nsu=http://foo.bar;s=foo/bar)
```

210.2.4. セキュリティーポリシー

許可されるセキュリティーポリシーを設定する場合は、よく知られた OPC UA URI（例：http://opcfoundation.org/UA/SecurityPolicy#Basic128Rsa15）を使用するか、Milo 列挙リテラル（例： None）を使用できます。不明なセキュリティーポリシー URI または enum の指定はエラーです。

既知のセキュリティーポリシー URI と enum リテラルは、SecurityPolicy.javaで確認できます。

注記： セキュリティーポリシーの場合は、大文字と小文字が区別されると見なされます。

210.3. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第211章 OPC UA サーバーコンポーネント

Camel バージョン 2.19 から利用可能

Milo Server コンポーネントは、Eclipse Milo™ 実装を使用して OPC UA サーバーを提供します。

Java 8: このコンポーネントには、ランタイム時に Java 8 が必要です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-milo</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Camel からエンドポイントに送信されたメッセージは、OPC UA サーバーから OPC UA クライアントで利用できます。OPC UA クライアントからの値の書き込みリクエストが Apache Camel に送信されるメッセージをトリガーします。

OPC UA Server コンポーネントは 19 個のオプションをサポートします。これらのオプションは以下のとおりです。

Name	説明	デフォルト	Type
namespaceUri (common)	名前空間の URI。デフォルトは urn:org:apache:camel です。		文字列
applicationName (common)	アプリケーション名		文字列
applicationUri (common)	アプリケーション URI		文字列
productUri (common)	製品 URI		文字列

Name	説明	デフォルト	Type
bindPort (common)	サーバーがバインドする TCP ポート		int
strictEndpointUrls Enabled (common)	厳密なエンドポイント URL を適用するかどうかを設定します。	false	boolean
serverName (common)	サーバー名		文字列
hostname (common)	サーバーのホスト名		文字列
securityPolicies (common)	セキュリティーポリシー		Set
securityPoliciesById (common)	URI または名前別のセキュリティーポリシー		String>
ユーザー認証の認証情報 （共通）	user1:pwd1,user2:pwd2 の形式でユーザーパスワードの組み合わせを設定し、パスワードは URL でデコードされます。		文字列
enableAnonymous Authentication (common)	匿名認証の有効化、デフォルトでは無効になっています。	false	boolean
bindAddresses (common)	サーバーがバインドする必要のあるローカルアドレスのアドレスを設定します。		文字列
buildInfo (common)	サーバービルド情報		BuildInfo
serverCertificate (common)	サーバー証明書		結果
certificateManager (common)	サーバー証明書マネージャー		CertificateManager
certificateValidator (common)	クライアント証明書のバリデーター		CertificateValidator>
defaultCertificateValidator (common)	デフォルトのファイルベースのアプローチを使用したクライアント証明書のバリデーター		ファイル

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

211.1. URI 形式

```
milo-server:itemId[?options]
```

211.2. URI オプション

OPC UA サーバーエンドポイントは、URI 構文を使用して設定します。

```
milo-server:itemId
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

211.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
itemId	項目に **必要な** ID		文字列

211.2.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

211.3. 関連項目

- ***Configuring Camel (Camel の設定)***
- ***コンポーネント***
- ***エンドポイント***
- ***はじめに***

第212章 MIME MULTIPART DATAFORMAT

Camel バージョン 2.17 から利用可能

MIME 分割メッセージをメッセージボディーとして持つ Camel メッセージに添付された Camel メッセージを変換できるこのデータ形式（アタッチメントなし）。

これのユースケースとして、ユーザーは、特別なプロトコル実装（HTTP エンドポイント経由での MIME マルチパート送信）またはトンネリングソリューション（例： camel-jms はアタッチメントをサポート）として、直接サポートしないエンドポイントを介してアタッチメントを送信できるようにすることです。 JMS キューへメッセージを送信し、JMS キューからメッセージを受信して再度マーシャリングします（アタッチメントのあるメッセージボディーに）。

mime-multipart データフォーマットの marshal オプションは、添付ファイルを持つメッセージを MIME-Multipart メッセージに変換します。パラメーター「multipartWithoutAttachment」を true に設定すると、1 つの部分を持つマルチパートメッセージに添付せずにメッセージをマーシャルします。パラメーターが false に設定されている場合は、メッセージをそのまま残します。

mulitpart の MIME ヘッダー（"MIME-Version" および "Content-Type" は camel ヘッダーとしてメッセージに設定されます）。パラメーター "headersInline" を true に設定すると、いつでも MIME マルチパートメッセージも作成されます。
さらに、マルチパートの MIME ヘッダーは、Camel ヘッダーとしてではなく、メッセージボディーの一部として記述されます。

mime-multipart データフォーマットの unmarshal オプションは、MIME-Multipart メッセージを添付ファイルと共に camel メッセージに変換し、他のメッセージをそのまま残します。MIME-Multipart メッセージの MIME ヘッダーは Camel ヘッダーとして設定する必要があります。アンマーシャリングは、「Content-Type」ヘッダーが「multipart」タイプに設定されている場合にのみ行われます。オプション "headersInline" が true に設定されている場合、ボディーは常に MIME メッセージとして解析されます。メッセージボディーがストリームで、ストリームキャッシュが有効になっていない場合、メッセージボディーに MIME ヘッダーを持つ MIME メッセージがないメッセージのボディーは空のメッセージに置き換えられます。Camel バージョン 2.17.1 まで、ボディータイプとストリームキャッシュ設定に関係なく、MIME マルチパートメッセージを含まないすべてのメッセージボディーが発生します。

212.1. オプション

MIME Multipart データフォーマットは、以下に示す 6 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
multipartSubType	**mixed**	**文字列**	MIME Multipart のサブタイプを指定します。デフォルトは混合です。
multipartWithoutAttachment	**false**	**ブール値**	アタッチメントのないメッセージも MIME マルチパートにマーシャルされるかどうかを定義します（1 つのボディーの一部のみ）。デフォルトは false です。
headersInline	**false**	**ブール値**	MIME-Multipart ヘッダーがメッセージボディー(true)の一部であるか、または Camel ヘッダー(false)として設定されるかどうかを定義します。デフォルトは false です。
includeHeaders		**文字列**	MIME ヘッダーとして MIME マルチパートにも含まれる Camel ヘッダーを定義する正規表現。これは、headerInline が true に設定されている場合にのみ機能します。デフォルトではヘッダーは含まれません。
binaryContent	**false**	**ブール値**	MIME マルチパートのバイナリー部分のコンテンツがバイナリー(true)または Base-64 でエンコードされた(false)のデフォルトであるかを定義します。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

212.2. メッセージヘッダー(MARSHAL)

Name	タイプ	説明
message-Id	文字列	マーシャリング操作は、「headersInline」パラメーターが false に設定されている場合に、生成された MIME メッセージ ID に設定します。
MIME-Version	文字列	マーシャリング操作は、「headersInline」パラメーターが false に設定されている場合、このパラメーターを適用された MIME バージョン(1.0)に設定します。
Content-Type	文字列	このヘッダーの内容は、メッセージボディーの部分のコンテンツタイプとして使用されます。コンテンツタイプが設定されていない場合、「application/octet-stream」が想定されます。マーシャリング操作の後に、「headersInline」パラメーターが true に設定されている場合、コンテンツタイプは「multipart/related」に設定されます。

Name	タイプ	説明
Content-Encoding	文字列	受信コンテンツタイプが "text/*" の場合、コンテンツエンコーディングはボディー部分の Content-Type MIME ヘッダーの encoding パラメーターに設定されます。さらに、テキストのために指定の文字セットがバイナリー変換に適用されます。

212.3. メッセージヘッダー（アンマーシャリング）

Name	タイプ	説明
Content-Type	文字列	このヘッダーが「multipart/*"」に設定されていない場合、アンマーシャリング操作は何もしません。他の場合、multipart は attachments で camel メッセージに解析され、ヘッダーはボディーの部分の Content-Type ヘッダーに設定されます（ただし、これが application/octet-stream である場合を除く）。後者の場合、ヘッダーが削除されます。
Content-Encoding	文字列	ボディー部分の content-type に encoding パラメーターが含まれる場合、このヘッダーはこのエンコーディングパラメーターの値に設定されます（MIME 終了記述子から Java エンコーディング記述子への変換）。
MIME-Version	文字列	unmarshal 操作はこのヘッダーを読み取り、MIME マルチパートの解析に使用します。ヘッダーの後で削除

212.4. 例

```
from(...).marshal().mimeMultipart()
```

Content-Type ヘッダーが設定されていないメッセージでは、以下のメッセージ Camel ヘッダーで Message を作成します。

Camel メッセージヘッダー

```
Content-Type=multipart/mixed; \n boundary="----=_Part_0_14180567.1447658227051"
Message-Id=<...>
MIME-Version=1.0
```

```
The message body will be:
```

Camel メッセージボディー

```
------=_Part_0_14180567.1447658227051
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
Qm9keSB0ZXh0
------=_Part_0_14180567.1447658227051
Content-Type: application/binary
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Attachment File Name"
AAECAwQFBgc=
------=_Part_0_14180567.1447658227051--
```

ヘッダー Content-Type がルートに送信される "text/plain" に設定されたメッセージ

```
from("...").marshal().mimeMultipart("related", true, true, "(included|x-.*)", true);
```

特定の MIME ヘッダーを Camel ヘッダーとして設定せずにメッセージを作成します（Content-Type ヘッダーが Camel メッセージから削除されます）と、"x-" で始まる元のメッセージのヘッダーがすべて含まれ、名前に "included" のヘッダーも含まれる以下のメッセージボディーが作成されます。

Camel メッセージボディー

```
Message-ID: <...>
MIME-Version: 1.0
Content-Type: multipart/related;
    boundary="----=_Part_0_1134128170.1447659361365"
x-bar: also there
included: must be included
x-foo: any value

------=_Part_0_1134128170.1447659361365
Content-Type: text/plain
Content-Transfer-Encoding: 8bit

Body text
------=_Part_0_1134128170.1447659361365
Content-Type: application/binary
Content-Transfer-Encoding: binary
Content-Disposition: attachment; filename="Attachment File Name"

[binary content]
------=_Part_0_1134128170.1447659361365
```

212.5. 依存関係

Camel ルートで MIME-Multipart を使用するには、このデータ形式を実装する camel-mail の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加できます。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mail</artifactId>
  <version>x.x.x</version> <!-- use the same version as your Camel core version -->
</dependency>
```

第213章 MINA2 コンポーネント

Camel バージョン 2.10 で利用可能

mina2: コンポーネントとは、Apache MINA 2.xと連携するトランスポートです。

ヒント

Netty を Netty として使用することが推奨されますが、現在 Apache Mina よりもよりアクティブで一般的なプロジェクトです。

INFO: コンシューマーエンドポイントでは sync=false と注意してください。camel-mina2 以降、すべてのコンシューマーエクスチェンジは InOut になります。これは camel-mina とは異なります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mina2</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

213.1. URI 形式

```
mina2:tcp://hostname[:port][?options]
mina2:udp://hostname[:port][?options]
mina2:vm://hostname[:port][?options]
```

codec オプションを使用して、レジストリーに コーデック を指定できます。TCP を使用し、コーデックが指定されていない場合は、テキストライン ベースのコーデーションまたはオブジェクトのシリアライズを代わりに使用するかどうかを決定します。デフォルトでは、オブジェクトのシリアライズが使用されます。

codec が指定されていない場合、デフォルトで基本的な ByteBuffer ベースの codec が使用されます。

VM プロトコルは、同じ JVM のダイレクト転送メカニズムとして使用されます。

Mina プロデューサーにはデフォルトのタイムアウト値 30 秒があり、リモートサーバーからの応答を待機します。

通常の使用では、camel-mina は本文 content-message ヘッダーおよびエクスチェンジプロパティーのマーシャリングのみをサポートします。
ただし、オプション transferExchange により、ネットワークを介してエクスチェンジ自体を転送できます。以下のオプションを参照してください。

URI にクエリーオプションを追加するには、?option=value&option=value&...

213.2. オプション

Mina2 コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有の最小設定を使用するには、以下を実行します。		Mina2Configuration
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Mina2 エンドポイントは、URI 構文を使用して設定します。

```
mina2:protocol:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

213.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
protocol	使用する **必須** プロトコル		文字列
host	**使用** するホスト名。ローカルサーバーに localhost または 0.0.0.0 をコンシューマーとして使用します。プロデューサーには、リモートサーバーのホスト名または IP アドレスを使用します。		文字列
port	**必要な** ポート番号		int

213.2.2. クエリーパラメーター（27 パラメーター）：

Name	説明	デフォルト	Type
切断 （共通）	使用後、最小セッションから切断される（クローズ）かどうか。コンシューマーとプロデューサーの両方に使用できます。	false	boolean
minaLogger (common)	Apache MINA ロギングフィルターを有効にできます。Apache MINA は INFO レベルで slf4j ロギングを使用して、すべての入出力をログに記録します。	false	boolean
同期 （共通）	endpoint を一方向または request-response として設定するための設定。	true	boolean
タイムアウト (common)	リモートサーバーからの応答を待つ期間を指定するタイムアウトを設定できます。タイムアウトの単位はミリ秒単位であるため、60000 は 60 秒です。	30000	Long
writeTimeout (common)	MINA セッションにデータを送信するのにかかった最大時間。デフォルトは 10000 ミリ秒です。	10000	Long
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
clientMode (consumer)	clientMode が true の場合、mina コンシューマーはアドレスを TCP クライアントとして接続します。	false	boolean

Name	説明	デフォルト	Type
disconnectOnNoReply (consumer)	同期が有効な場合には、送信先の応答がない場合に、このオプションにより MinaConsumer が指示されます。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
noReplyLogLevel (consumer)	同期が有効な場合、ロギングに使用するロギングレベルが MinaConsumer に指定されます。	WARN	LoggingLevel
cachedAddress (producer)	InetAddress を一度作成し、再利用するかどうか。これを false に設定すると、ネットワークで DNS の変更を取得できます。	true	boolean
lazySessionCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合に、例外を回避するためにセッションを遅延的に作成できます。	true	boolean
maximumPoolSize (advanced)	TCP および UDP のワーカープール内のワーカースレッドの数	16	int
orderedThreadPoolExecutor (advanced)	順序付けされたスレッドプールを使用して、同じチャネルでイベントが順序付け処理されるかどうか。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
transferExchange (advanced)	TCP にのみ使用されます。ボディーだけでなく、ネットワーク上でエクスチェンジを転送できます。次のフィールドが転送されます。本文、Out ボディー、フォールトボディー、ヘッダー、送信ヘッダー、エクスチェンジプロパティー、エクスチェンジ例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。	false	boolean

Name	説明	デフォルト	Type
allowDefaultCodec (codec)	mina コンポーネントは、codec が null で、textline が false の場合、デフォルトのコーデックをインストールします。allowDefaultCodec を false に設定すると、mina コンポーネントがフィルターチェーンの最初の要素としてデフォルトのコーデックをインストールできなくなります。これは、SSL フィルターなど、別のフィルターがフィルターチェーンの最初のフィルターである必要がある場合に役立ちます。	true	boolean
codec (codec)	コーデック実装を使用するには、以下を行います。		ProtocolCodecFactory
decoderMaxLineLength (codec)	テキストラインプロトコルデコーダーの最大ライン長を設定するには、以下を行います。デフォルトでは Mina 自体のデフォルト値が 1024 が使用されています。	1024	int
encoderMaxLineLength (codec)	テキストラインプロトコルエンコーダーの最大ライン長を設定するには、以下を行います。デフォルトでは、Integer.MAX_VALUE である Mina 自体のデフォルト値が使用されます。	-1	int
encoding (codec)	TCP テキストラインコーデックおよび UDP プロトコルに使用するエンコーディング（charset 名）を設定できます。指定されていない場合、Camel は JVM のデフォルト Charset を使用します。		文字列
フィルター（コード）	使用する Mina IoFilters の一覧を設定できます。		リスト
テキストライン（コード）	TCP にのみ使用されます。codec が指定されていない場合、このフラグを使用してテキスト行ベースのコーデックを指定できます。指定されていない場合や、値が false の場合は、Object Serialization は TCP を介して想定されます。	false	boolean
textlineDelimiter (codec)	TCP および textline=true にのみ使用されます。使用するテキスト行区切り文字を設定します。指定しないと、Camel は DEFAULT を使用します。この区切り文字は、テキストの最後をマークするために使用されます。		Mina2TextLineDelimiter
autoStartTls (security)	SSL ハンドシェイクを自動起動するかどうか。	true	boolean
sslContextParameters (security)	SSL セキュリティーを設定します。		SSLContextParameters

213.3. カスタムコーデックの使用

独自のコーデックの作成方法 を参照してください。camel-mina でカスタムコーデックを使用するには、たとえば Spring XML ファイルに Bean を作成してレジストリーにコーデックを登録する必要があります。次に、codec オプションを使用してコーデックの Bean ID を指定します。カスタムコーデックを持つ HL7 を参照してください。

213.4. SYNC=FALSE のあるサンプル

この例では、Camel はポート 6200 で TCP 接続をリッスンするサービスを公開します。テキストライン コーデックを使用します。ルートでは、ポート 6200 をリッスンする Mina コンシューマーエンドポイントを作成します。

```
from("mina2:tcp://localhost:" + port1 + "?textline=true&sync=false").to("mock:result");
```

サンプルはユニットテストの一部であるため、ポート 6200 で一部のデータを送信してテストします。

```
MockEndpoint mock = getMockEndpoint("mock:result");
mock.expectedBodiesReceived("Hello World");

template.sendBody("mina2:tcp://localhost:" + port1 + "?textline=true&sync=false", "Hello
World");

assertMockEndpointsSatisfied();
```

213.5. SYNC=TRUE のあるサンプル

次の例では、ポート 6201 で TCP サービスを公開するより一般的なユースケースもあります。このユースケースでは、テキストラインコーデックも使用します。ただし、今回は応答を返すので、コンシューマーで sync オプションを true に設定します。

```
from("mina2:tcp://localhost:" + port2 + "?textline=true&sync=true").process(new Processor() {
    public void process(Exchange exchange) throws Exception {
        String body = exchange.getIn().getBody(String.class);
        exchange.getOut().setBody("Bye " + body);
    }
});
```

次に、一部のデータを送信し、template.requestBody（） メソッドを使用して応答を取得してサンプルをテストします。応答は String であるため、これを String にキャストし、レスポンスがプロセッサーコードロジックに動的に設定されていることをアサートできます。

```
String response = (String)template.requestBody("mina2:tcp://localhost:" + port2 + "?
textline=true&sync=true", "World");
assertEquals("Bye World", response);
```

213.6. SPRING DSL を使用した例

Spring DSL は MINA にも使用することができます。以下の例では、TCP サーバーをポート 5555 で公開します。

```
<route>
  <from uri="mina2:tcp://localhost:5555?textline=true"/>
  <to uri="bean:myTCPOrderHandler"/>
</route>
```

上記のルートでは、テキストラインコーデックを使用して TCP サーバーをポート 5555 で公開します。ID myTCPOrderHandler を持つ Spring Bean にリクエストを処理し、返信を返します。たとえば、ハンドラー Bean は以下のように実装できます。

```
public String handleOrder(String payload) {
    ...
    return "Order: OK"
}
```

213.7. 完了後にセッションを閉じる

サーバーとして動作する場合は、クライアント変換の完了時にセッションを閉じることがあります。Camel にセッションを閉じるよう指示するには、CamelMinaCloseSessionWhenComplete キーでヘッダーをブール値 true に追加する必要があります。

たとえば、以下の例では bye メッセージをクライアントに書き直した後にセッションを閉じます。

```
    from("mina2:tcp://localhost:8080?sync=true&textline=true").process(new Processor() {
        public void process(Exchange exchange) throws Exception {
            String body = exchange.getIn().getBody(String.class);
            exchange.getOut().setBody("Bye " + body);

exchange.getOut().setHeader(Mina2Constants.MINA_CLOSE_SESSION_WHEN_COMPLETE,
true);
        }
    });
```

213.8. メッセージの IOSESSION を取得します。

このキー Mina2Constants.MINA_IOSESSION を使用してメッセージヘッダーから IoSession を取得し、さらに Mina2Constants.MINA_LOCAL_ADDRESS キーおよびリモートホストアドレスのキー Mina2Constants.MINA_REMOTE_ADDRESS を持つローカルホストアドレスを取得することもできます。

213.9. MINA フィルターの設定

フィルターを使用すると、SslFilter などの一部の Mina フィルターを使用できます。カスタマイズしたフィルターを実装することもできます。codec および logger は、IoFilter タイプの Mina フィルターとしても実装されていることに注意してください。定義したフィルターは、フィルターチェーンの最後に追加されます。つまり、codec および logger の後に追加されます。

213.10. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- Netty

第214章 MLLP コンポーネント

Camel バージョン 2.17 から利用可能

MLLP コンポーネントは、MLLP プロトコルの nuances を処理し、Healthcare プロバイダーが MLLP プロトコルを使用して他のシステムと通信するのに必要な機能を提供します。 MLLP コンポーネントは、単純な設定 URI、自動化された HL7 承認生成および自動承認インターテグレーションを提供します。

MLLP プロトコルは通常、多数の同時 TCP 接続を使用しません。単一のアクティブな TCP 接続は通常です。 そのため、MLLP コンポーネントは標準の Java Sockets をベースとした簡単な thread-per-connection モデルを使用します。これにより、実装がシンプルになり、Camel 自体以外の依存関係がなくなります。

コンポーネントは以下をサポートします。

- TCP サーバーを使用する Camel コンシューマー
- TCP クライアントを使用した Camel プロデューサー

MLLP コンポーネントは byte[] ペイロードを使用し、Camel Type Conversion を使用して byte[] を別のタイプに変換します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mllp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

214.1. MLLP オプション

MLLP コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
logPhi (advanced)	PHI データをログに記録するにはコンポーネントを設定します。	true	ブール値
logPhiMaxBytes (advanced)	ログエントリーにログインする PHI の最大バイト数を設定します。	5120	整数
defaultCharset (advanced)	String 変換へのバイト変換に使用するデフォルトの文字セットを設定します。	ISO-8859-1	文字列
設定 （共通）	MLLP エンドポイントの作成時に使用するデフォルト設定を設定します。		MllpConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

MLLP エンドポイントは、URI 構文を使用して設定します。

```
mllp:hostname:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

214.1.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
hostname	TCP 接続に **必要な** ホスト名または IP。デフォルト値は null で、すべてのローカル IP アドレスを意味します。		文字列
port	TCP 接続に **必要な** ポート番号		int

214.1.2. クエリーパラメーター（27 パラメーター）：

Name	説明	デフォルト	Type
autoAck (common)	MLLP Acknowledgement MLLP コンシューマーのみの自動生成の有効化/無効化	true	boolean

Name	説明	デフォルト	Type
bufferWrites (common)	**非推奨**: ソケットに書き込む前に HL7 ペイロードのバッファー処理を有効/無効にします。	false	boolean
hl7Headers (common)	HL7 Message MLLP コンシューマーのみからのメッセージヘッダーの自動生成を有効または無効にします。	true	boolean
requireEndOfData (common)	MLLP 標準に対する厳密なコンプライアンスの有効化/無効化。MLLP 標準は、START_OF_BLOCKhl7 payloadEND_OF_BLOCKEND_OF_DATA を指定しますが、一部のシステムでは最終的な END_OF_DATA バイトが送信されません。この設定は、最終的な END_OF_DATA バイトが必要かどうか、または任意になるかどうかを制御します。	true	boolean
stringPayload (common)	ペイロードを String に変換するか、有効化/無効にします。有効にすると、外部システムから受信した HL7 ペイロードは String に変換されます。charsetName プロパティーを設定すると、その文字セットが変換に使用されます。charsetName プロパティーが設定されていない場合、MSH-18 の値は、適切な文字セットを決定するために使用されます。MSH-18 が設定されていない場合は、デフォルトの ISO-8859-1 文字セットが使用されます。	true	boolean
validatePayload (common)	HL7 Payloads の検証を有効/無効にします。有効にされている場合、外部システムから受信した HL7 Payloads が検証されます（検証の詳細は Hl7Util.generateInvalidPayloadExceptionMessage を参照してください）。ペイロードが検出されると、MllpInvalidMessageException（コンシューマーの場合）または MllpInvalidAcknowledgementException がスローされます。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。つまり、コンシューマーが受信メッセージを受信しようしている間に発生した例外や、ルーティングエラーハンドラーによって処理されます。無効な場合、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して WARN または ERROR レベルでロギングし、無視されます。	true	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。	InOut	ExchangePattern
同期 （詳細）	同期処理を厳密に使用するかどうかを設定します（このコンポーネントは同期操作のみをサポートします）。	true	boolean
backlog (tcp)	受信接続のindication（接続リクエスト）の最大キューの長さが、backlog パラメーターに設定されます。キューが満杯になると接続が到達すると、接続は拒否されます。	5	整数
lenientBind (tcp)	TCP サーバーのみ： TCP ServerSocket がバインドされる前にエンドポイントを起動できます。一部の環境では、TCP ServerSocket をバインドする前にエンドポイントを開始できるようにする必要がある場合があります。	false	boolean
maxConcurrentConsumers (tcp)	許可される同時 MLLP コンシューマー接続の最大数。新しい接続が受信され、最大数が確立されている場合、新しい接続は即座にリセットされます。	5	int
reuseAddress (tcp)	SO_REUSEADDR ソケットオプションを有効化/無効化します。	false	ブール値
acceptTimeout (timeout)	TCP 接続 TCP サーバーのみを待機する際のタイムアウト（ミリ秒単位）	60000	int
bindRetryInterval (timeout)	TCP サーバーのみ： バインドの試行まで待機する時間（ミリ秒単位）。	5000	int
bindTimeout (timeout)	TCP サーバーのみ： サーバーポートへのバインディングを再試行する時間（ミリ秒単位）。	30000	int
connectTimeout (timeout)	TCP 接続 TCP クライアントのみを確立するタイムアウト（ミリ秒単位）	30000	int
idleTimeout (timeout)	Client TCP Connection をリセットする前に許可される概算時間。null 値またはゼロ以下の値を指定すると、アイドルタイムアウトが無効になります。		整数

Name	説明	デフォルト	Type
maxReceiveTimeouts (timeout)	**非推奨**。TCP 接続をリセットする前に許可されるタイムアウトの最大数（receiveTimeout で指定）。		整数
keepAlive (tcp)	SO_KEEPALIVE ソケットオプションを有効化/無効化します。	true	ブール値
receiveBufferSize (tcp)	SO_RCVBUF オプションを指定された値に設定します（バイト単位）。	8192	整数
sendBufferSize (tcp)	SO_SNDBUF オプションを指定された値（バイト単位）に設定します。	8192	整数
tcpNoDelay (tcp)	TCP_NODELAY ソケットオプションを有効化/無効にします。	true	ブール値
readTimeout (timeout)	MLLP フレームの開始後に使用される SO_TIMEOUT 値（ミリ秒単位）。	5000	int
receiveTimeout (timeout)	MLLP フレームの開始を待機する際に使用される SO_TIMEOUT 値（ミリ秒単位）。	15000	int
charsetName (codec)	エクスチェンジに CamelCharsetName プロパティーを設定します。		文字列

214.2. MLLP コンシューマー

MLLP コンシューマーは、MLLP-framed メッセージの受信と、HL7 Acknowledgement の送信をサポートします。 MLLP コンシューマーは、HL7 Acknowledgement（HL7 アプリケーション承認のみ（AA、AE および AR）のみ）を自動的に生成できます。または、CamelMllpAcknowledgement 交換プロパティーを使用して承認を指定できます。 さらに、生成される確認応答のタイプは、CamelMllpAcknowledgementType エクスチェンジプロパティーを設定することで制御できます。

214.3. メッセージヘッダー

MLLP コンシューマーは、これらのヘッダーを Camel メッセージに追加します。

キー	説明	
CamelMllpLocalAddress	ソケットのローカル TCP アドレス	

CamelMllpRemoteAddress	ソケットのローカル TCP アドレス	
CamelMllpSendingApplication	MSH-3 値	
CamelMllpSendingFacility	MSH-4 の値	
CamelMllpReceivingApplication	MSH-5 の値	
CamelMllpReceivingFacility	MSH-6 値	
CamelMllpTimestamp	MSH-7 の値	
CamelMllpSecurity	MSH-8 の値	
CamelMllpMessageType	MSH-9 の値	
CamelMllpEventType	MSH-9-1 値	
CamelMllpTriggerEvent	MSH-9-2 値	
CamelMllpMessageControlId	MSH-10 の値	
CamelMllpProcessingId	MSH-11 の値	
CamelMllpVersionId	MSH-12 の値	
CamelMllpCharset	MSH-18 の値	

すべてのヘッダーは String 型です。ヘッダーの値がない場合、その値は null になります。

214.4. エクスチェンジプロパティー

MLLP コンシューマーは TCP ソケットの生成と状態を生成および確認応答のタイプは、Camel エクスチェンジのこれらのプロパティーで制御できます。

キー	Type	説明
CamelMllpAcknowledgement	byte[]	このプロパティーが存在する場合、このプロパティーは MLLP Acknowledgement としてクライアントに送信されます。

CamelMllpAcknowledgementString	文字列	存在し、CamelMllpAcknowledgement が存在しない場合、このプロパティーは MLLP Acknowledgement としてクライアントに送信されます。
CamelMllpAcknowledgementMsaText	文字列	CamelMllpAcknowledgement または CamelMllpAcknowledgementString が存在しない場合、このプロパティーを使用して生成された HL7 確認で MSA-3 の内容を指定できます。
CamelMllpAcknowledgementType	文字列	CamelMllpAcknowledgement または CamelMllpAcknowledgementString が存在しない場合、このプロパティーを使用して HL7 の確認タイプ（AA、AE、AR など）を指定できます。
CamelMllpAutoAcknowledge	ブール値	autoAck クエリーパラメーターを上書きします。
CamelMllpCloseConnectionBeforeSend	ブール値	true の場合、データを送信する前にソケットが閉じられます。
CamelMllpResetConnectionBeforeSend	ブール値	true の場合、データを送信する前にソケットがリセットされます。
CamelMllpCloseConnectionAfterSend	ブール値	true の場合、データ送信直後にソケットが閉じられます。
CamelMllpResetConnectionAfterSend	ブール値	true の場合、データ送信直後にソケットがリセットされます。

214.5. MLLP PRODUCER

MLLP Producer は、MLLP-framed メッセージの送信および HL7 承認の受信をサポートします。MLLP Producer interrogates the HL7 Acknowledgment（負の確認応答が受信されると例外が発生します） 受信した確認応答が補間され、負の確認が行われると例外が発生します。

214.6. メッセージヘッダー

MLLP Producer は以下のヘッダーを Camel メッセージに追加します。

キー	説明	
CamelMllpLocalAddress	ソケットのローカル TCP アドレス	
CamelMllpRemoteAddress	ソケットのリモート TCP アドレス	
CamelMllpAcknowledgement	受信された HL7 Acknowledgment byte[]	
CamelMllpAcknowledgementString	HL7 Acknowledgment が受信され、String に変換されます。	

214.7. エクスチェンジプロパティー

TCP ソケットの状態は、Camel エクスチェンジのこれらのプロパティーで制御できます。

キー	Type	説明
CamelMllpCloseConnectionBeforeSend	ブール値	true の場合、データを送信する前にソケットが閉じられます。
CamelMllpResetConnectionBeforeSend	ブール値	true の場合、データを送信する前にソケットがリセットされます。
CamelMllpCloseConnectionAfterSend	ブール値	true の場合、データ送信直後にソケットが閉じられます。
CamelMllpResetConnectionAfterSend	ブール値	true の場合、データ送信直後にソケットがリセットされます。

第215章 モックコンポーネント

Mock コンポーネントのドキュメントは、現在利用できません。

第216章 MONGODB コンポーネント

Camel バージョン 2.10 で利用可能

Wikipedia に従って、「NoSQL is a moving promote a loosely defined class of non-rational data store that breaks of an long history ofational databases and ACID guarantees.」 NoSQL ソリューションは過去数年に人気があり、Facebook、LinkedIn、Twitter などの主要サイトやサービスを利用することが知られています。これらのソリューションは、それらを広範囲に使用してスケーラビリティーや生産性を実現していることが知られています。

基本的に、NoSQL ソリューションは従来の faillock(Relational Database Management Systems)とは異なります。これは、SQL をクエリー言語として使用しておらず、一般的に ACID のようなトランザクション動作やリレーショナルデータベースデータを提供していないことです。代わりに、柔軟なデータ構造とスキーマの概念に基づいて設計されており（固定スキーマを持つデータベーステーブルの従来の概念がドロップされると）、商用ハードウェアや高速処理に極端なスケーラビリティーが発生します。

MongoDB は非常に人気のあるNoSQL ソリューションであり、camel-mongodb コンポーネントは Camel と MongoDB を統合するため、プロデューサーとして（コレクションにおける同等の操作）およびコンシューマー（MongoDB コレクションからドキュメントを多用）の両方と対話できます。

MongoDB は、（オフィスドキュメントではなく）ドキュメントの概念に関するもので、JSON/BSON で定義された階層データです。このコンポーネントページは、それらに精通していることを前提としています。そうでない場合は、http://www.mongodb.org/ にアクセスします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mongodb</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

216.1. URI 形式

```
mongodb:connectionBean?
database=databaseName&collection=collectionName&operation=operationName[&moreOptions...]
```

216.2. MONGODB オプション

MongoDB コンポーネントにはオプションがありません。

MongoDB エンドポイントは、URI 構文を使用して設定します。

```
mongodb:connectionBean
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

216.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
connectionBean	使用する **必須** の com.mongodb.Mongo の名前。		文字列

216.2.2. クエリーパラメーター（23 パラメーター）：

Name	説明	デフォルト	Type
コレクション （共通）	このエンドポイントにバインドする MongoDB コレクションの名前を設定します。		文字列
collectionIndex (common)	コレクションインデックスを設定します(JSON FORMAT : field1 : order1, field2 : order2)。		文字列
createCollection (common)	初期中にコレクションが存在しない場合は作成します。デフォルトは true です。	true	boolean
データベース （共通）	MongoDB データベースの名前をターゲットに設定します。		文字列
操作 (common)	このエンドポイントが MongoDB に対して実行される操作を設定します。使用できる値は、MongoDbOperation を参照してください。		MongoDbOperation
outputType (common)	プロデューサーの出力を、DBObjectList DBObject または DBCursor 型に変換します。DBObjectList または DBCursor は findAll および aggregate に適用されます。DBObject は他のすべての操作に適用されます。		MongoDbOutputType

Name	説明	デフォルト	Type
writeConcern (common)	標準の MongoDB を使用して、MongoDB の書き込み操作用に WriteConcern を設定します。link WriteConcernvalueOf(String)メソッドを呼び出すことで、WriteConcern クラスのフィールドから解決されました。	確認済み	WriteConcern
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cursorRegenerationDelay (advanced)	MongoDB の調整可能なカーソルは、新規データが到達するまでブロックされます。新しいデータが挿入されない場合、一定の時間が経過すると、カーソルは自動的に解放され、MongoDB サーバーが閉じられます。クライアントは必要に応じてカーソルを再生成することが予想されます。この値は、新しいカーソルの取得を試みる前に待機する時間を指定します。失敗した場合は、次の試行が実行されるまでの時間を指定します。デフォルト値は 1000ms です。	1000	Long
Dynamicity （詳細）	このエンドポイントが受信エクスチェンジプロパティーからターゲットデータベースおよびコレクションを動的に解決しようとするかどうかを設定します。実行時にデータベースおよび他の静的エンドポイント URI に指定されたコレクションの上書きに使用できます。パフォーマンスを向上させるために、デフォルトでは無効になっています。有効にすると、パフォーマンスが最小限に抑えられます。	false	boolean

Name	説明	デフォルト	Type
readPreference (advanced)	Mongo コネクションで MongoDB ReadPreference を設定します。接続に直接設定された読み取り設定は、この設定によって上書きされます。link ReadPreferencevalueOf(String)ユーティリティーメソッドは、渡されたreadPreference値を解決するために使用されます。使用できる値の例には、最も近い、プライマリー、またはセカンダリーなどがあります。		ReadPreference
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
writeResultAsHeader (advanced)	書き込み操作では、OUT メッセージのボディーとして WriteResult を返すのではなく、IN メッセージを OUT に変換し、WriteResult をヘッダーとしてアタッチします。	false	boolean
persistentId (tail)	1 つのテールトラッキングコレクションでは、調整可能な複数のコンシューマーの多数のトラッカーをホストできます。それらを分離させるには、各トラッカーに固有の persistentId がなければなりません。		文字列
persistentTailTracking (tail)	永続的なテールトラッキングを有効にします。これは、システムを再起動しても最後に使用されたメッセージを追跡するメカニズムです。次回システムが起動すると、エンドポイントは最後にスライピングレコードを停止した時点からカーソルを復元します。	false	boolean
persistRecords (tail)	テールトラッキングデータが MongoDB に永続化されるテールレコードの数を設定します。	-1	int
tailTrackCollection (tail)	テールトラッキング情報が永続化されるコレクション。指定されていない場合、リンク MongoDbTailTrackingConfigDEFAULT_COLLECTION がデフォルトで使用されます。		文字列
tailTrackDb (tail)	テールトラッキングメカニズムが永続化するデータベースを示します。指定されていない場合、現在のデータベースはデフォルトで選択されます。Dynamicity は有効にされても考慮されません。つまり、テールトラッキングデータベースはエンドポイントの初期化とは異なります。		文字列

Name	説明	デフォルト	Type
tailTrackField (tail)	最後に追跡された値を配置するフィールド。指定されていない場合、リンク MongoDbTailTrackingConfigDEFAULT_FIELD がデフォルトで使用されます。		文字列
tailTrackIncreasingField (tail)	性質を引き上げる受信レコードの correlation フィールドは、生成されるたびにカーソルを配置するために使用されます。カーソルは、タイプのクエリーで作成される（TrackIncreasingField lastValue（永続テールトラッキングから復元される可能性がある））になります。Integer、Date、String などを使用できます。注記： 現在の時点でのドット表記のサポートはないため、フィールドはドキュメントのトップレベルに指定する必要があります。		文字列
tailTrackingStrategy (tail)	増加するフィールド値を抽出し、テールカーソルを置くクエリーを作成するために使用されるストラテジーを設定します。	リテラル	MongoDBTailTracking Enum

216.3. SPRING XML でのデータベースの設定

以下の Spring XML は、MongoDB インスタンスへのコネクションを定義する Bean を作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="mongoBean" class="com.mongodb.Mongo">
        <constructor-arg name="host" value="${mongodb.host}" />
        <constructor-arg name="port" value="${mongodb.port}" />
    </bean>
</beans>
```

216.4. サンプルルート

Spring XML で定義された以下のルートは、コレクションで操作 dbStats を実行します。

指定されたコレクションの DB 統計を取得

```
<route>
  <from uri="direct:start" />
```

```
  <!-- using bean 'mongoBean' defined above -->
  <to uri="mongodb:mongoBean?
database=${mongodb.database}&amp;collection=${mongodb.collection}&amp;operation=getDbStats"
/>
  <to uri="direct:result" />
</route>
```

216.5. MONGODB 操作 - プロデューサーエンドポイント

216.5.1. クエリー操作

216.5.1.1. findById

この操作は、_id フィールドが IN メッセージボディーの内容と一致するコレクションから 1 つの要素のみを取得します。受信オブジェクトは、BSON タイプと同等のものをすべて使用できます。http://bsonspec.org//specification[http://bsonspec.org//specification] および http://www.mongodb.org/display/DOCS/Java+Types を参照してください。

```
from("direct:findById")
   .to("mongodb:myDb?database=flights&collection=tickets&operation=findById")
   .to("mock:resultFindById");
```

ヒント

オプションのパラメーター をサポートします。この操作は、フィールドフィルターの指定をサポートします。「オプションのパラメーターの指定」を参照してください。

216.5.1.2. findOneByQuery

この操作を使用して、MongoDB クエリーに一致するコレクションから 1 つの要素のみを取得します。クエリーオブジェクトは IN メッセージボディーから抽出 さ れ ます。JSON String または Hashmap を使用できます。詳細は 、「タイプ変換」を参照してください。

クエリーのない例（コレクションのオブジェクトを返します）。

```
from("direct:findOneByQuery")
   .to("mongodb:myDb?database=flights&collection=tickets&operation=findOneByQuery")
   .to("mock:resultFindOneByQuery");
```

クエリーを使用する例（一致する 1 つの結果を返す）：

```
from("direct:findOneByQuery")
```

```
    .setBody().constant("{ \"name\": \"Raul Kripalani\" }")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findOneByQuery")
    .to("mock:resultFindOneByQuery");
```

ヒント

オプションのパラメーター をサポートします。この操作は、フィールドフィルターや sort 句の指定をサポートします。「 オプションのパラメーターの指定 」を参照してください。

216.5.1.3. findAll

findAll 操作は、クエリーに一致するドキュメントをすべて返します。そうでない場合は、コレクションに含まれるすべてのドキュメントが返されます。クエリーオブジェクトは IN メッセージボディーから抽出 さ れ ます。JSON String または Hashmap を使用できます。詳細は 、「タイプ変換 」を参照してください。

クエリーなしの例（コレクション内のすべてのオブジェクトを返します）。

```
from("direct:findAll")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findAll")
    .to("mock:resultFindAll");
```

クエリーを使用する例（一致するすべての結果を返します）。

```
from("direct:findAll")
    .setBody().constant("{ \"name\": \"Raul Kripalani\" }")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findAll")
    .to("mock:resultFindAll");
```

ページングと効率的な取得は、以下のヘッダーでサポートされます。

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ
CamelMongoDbNumToSkip	**MongoDbConstants.NUM_TO_SKIP**	カーソルの先頭にある特定の数の要素を破棄します。	int/Integer
CamelMongoDbLimit	**MongoDbConstants.LIMIT**	返される要素の数を制限します。	int/Integer
CamelMongoDbBatchSize	**MongoDbConstants.BATCH_SIZE**	1 つのバッチで返される要素の数を制限します。通常、カーソルは結果オブジェクトのバッチを取得し、ローカルに保存します。batchSize が正である場合、取得されるオブジェ	int/Integer

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ
		クトのリストをより効率的に処理するためのパフォーマンスを最適化し、データ転送を制限するように調整することができます。batchSize が負の値の場合、最大バッチサイズ制限（通常は 4MB）内で一致するオブジェクトの数を制限し、カーソルが閉じられます。たとえば、batchSize が -10 の場合、サーバーは最大 10 つのドキュメントを返し、	

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ
		4MB にならない数ンを返し、カルを閉じます。この機能は、ドキュメントが最大サイズ内に収まる必要がある点で limit（）とは異なり、カーソルサーバー側でリクエストを送信する必要がなくなります。バッチサイズは、カーソルが反復された後でも変更できます。この場合、次のバッチ取得に設定が適用されます。	

また、上記のヘッダーを設定するよりも簡単なエンドポイントオプションとして

outputType=DBCursor(Camel 2.16+)を含めることで、サーバーからルートへ返されたドキュメントを「ストリーム」することもできます。これにより、Mongo シェルで findAll（）を実行していたのと同様に、Mongo ドライバーから DBCursor が操作し、ルートが結果を繰り返し処理できるようになります。デフォルトでは、このオプションがないと、このコンポーネントはドライバーのカーソルから List にドキュメントを読み込んで、ルートに戻ります。これにより、多数のインメモリーオブジェクトが発生する可能性があります。DBCursor を使用すると、一致するドキュメントの数が尋ねられない点に注意してください。詳細は、MongoDB のドキュメントサイトを参照してください。

outputType=DBCursor および batch size オプションを使用する例：

```
from("direct:findAll")
   .setHeader(MongoDbConstants.BATCH_SIZE).constant(10)
   .setBody().constant("{ \"name\": \"Raul Kripalani\" }")
   .to("mongodb:myDb?
database=flights&collection=tickets&operation=findAll&outputType=DBCursor")
   .to("mock:resultFindAll");
```

findAll 操作は、ページングを使用している場合に結果ページを繰り返し処理できるように、以下の OUT ヘッダーも返します。

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	データ型
CamelMongoDbResultTotalSize	**MongoDbConstants.RESULT_TOTAL_SIZE**	クエリーに一致するオブジェクトの数。これには制限や省略は行われません。	int/Integer

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	データ型
CamelMongoDbResultPageSize	**MongoDbConstants.RESULT_PAGE_SIZE**	クエリーに一致するオブジェクトの数。これには制限や省略は行われません。	int/Integer

ヒント

オプションのパラメーター をサポートします。この操作は、フィールドフィルターや sort 句の指定をサポートします。「オプションのパラメーターの指定」を参照してください。

216.5.1.4. count

コレクション内のオブジェクトの合計数を返し、Long を OUT メッセージボディーとして返します。
以下の例は、「dynamicCollectionName」コレクション内のレコード数をカウントします。動的性が有効であるか、その結果、操作は「notableScientists」コレクションに対しては実行されませんが、「dynamicCollectionName」コレクションに対しては実行されません。

```
// from("direct:count").to("mongodb:myDb?
database=tickets&collection=flights&operation=count&dynamicity=true");
Long result = template.requestBodyAndHeader("direct:count", "irrelevantBody",
MongoDbConstants.COLLECTION, "dynamicCollectionName");
assertTrue("Result is not of type Long", result instanceof Long);
```

Camel 2.14 以降では、メッセージボディーにクエリーとして com.mongodb.DBObject オブジェクトを指定でき、操作はこの条件に一致するドキュメントの量を返します。

```
DBObject query = ...
Long count = template.requestBodyAndHeader("direct:count", query,
MongoDbConstants.COLLECTION, "dynamicCollectionName");
```

216.5.1.5. フィールドフィルター（プロジェクト）の指定

クエリー操作は、デフォルトで（すべてのフィールドを含む）一致するオブジェクトを完全に返します。ドキュメントが大きい場合に、フィールドのサブセットのみを取得する必要がある場合は、CamelMongoDbFieldsFilter ヘッダーで関連する DBObject （または JSON 文字列、マップなど）を DBObject に設定するだけで、すべてのクエリー操作でフィールドフィルターを指定 できます。

以下は、MongoDB の BasicDBObjectBuilder を使用して DBObject の作成を単純化する例です。これは、_id および boringField を除くすべてのフィールドを取得します。

```
// route: from("direct:findAll").to("mongodb:myDb?
database=flights&collection=tickets&operation=findAll")
DBObject fieldFilter = BasicDBObjectBuilder.start().add("_id", 0).add("boringField", 0).get();
Object result = template.requestBodyAndHeader("direct:findAll", (Object) null,
MongoDbConstants.FIELDS_FILTER, fieldFilter);
```

216.5.1.6. sort 句の指定

特定のフィールドで並び替えに基づき、コレクションから min/max レコードを取得する必要がしばしばあります。Mongo では、以下のような構文を使用して操作が実行されます。

```
db.collection.find().sort({_id: -1}).limit(1)
// or
db.collection.findOne({$query:{},$orderby:{_id:-1}})
```

Camel ルートでは、SORT_BY ヘッダーを findOneByQuery 操作とともに使用して、同じ結果を得ることができます。FIELDS_FILTER ヘッダーも指定された場合、操作は別のコンポーネントに直接渡すことができる単一のフィールド/値のペアを返します（パラメーター化された MyBatis SELECT クエリーなど）。以下の例は、コレクションから一時的な最新ドキュメントを取得し、documentTimestamp フィールドに基づいて結果を 1 つのフィールドに減らす方法を示しています。

```
.from("direct:someTriggeringEvent")
.setHeader(MongoDbConstants.SORT_BY).constant("{\"documentTimestamp\": -1}")
.setHeader(MongoDbConstants.FIELDS_FILTER).constant("{\"documentTimestamp\": 1}")
.setBody().constant("{}")
.to("mongodb:myDb?
```

```
database=local&collection=myDemoCollection&operation=findOneByQuery")
.to("direct:aMyBatisParameterizedSelect")
;
```

216.5.2. 作成/更新操作

216.5.2.1. insert

IN メッセージボディーから取得した MongoDB コレクションに新しいオブジェクトを挿入します。型変換は、DBObject または List に変換を試みます。
1 つの挿入と複数の挿入の 2 つのモードがサポートされます。複数の挿入の場合、エンドポイントは、すべてのタイプのオブジェクトのリスト、配列、またはコレクションを想定します。または、-DBObject に変換できます。すべてのオブジェクトは一度に挿入されます。エンドポイントは、入力に応じて呼び出すバックエンド操作（単一または複数の挿入）をインテリジェントな決定します。

例:

```
from("direct:insert")
   .to("mongodb:myDb?database=flights&collection=tickets&operation=insert");
```

操作は WriteResult を返します。WriteConcern または invokeGetLastError オプションの値によっては、getLastError（） が呼び出されました。書き込み操作の最終的な結果にアクセスするには、WriteResult で getLastError（）または getCachedLastError （） を呼び出して CommandResult を取得する必要があります。その後、CommandResult. ok（）、CommandResult. getErrorMessage（）、および/または CommandResult.getException ( )を呼び出すことで結果を確認できます。

新しいオブジェクトの _id は、コレクションで一意である必要があることに注意してください。値を指定しない場合、MongoDB は自動的に生成します。しかし、指定して一意でない場合は、挿入操作は失敗します（Camel が通知するには、callGetLastError を有効にするか、書き込み結果を待つ WriteConcern を設定する必要があります）。

これはコンポーネントの制限ではありませんが、MongoDB でスループットを向上する方法です。カスタムの _id を使用している場合は、一意のアプリケーションレベル（およびこれが適切なプラクティスである）で確認できることが想定されます。

Camel 2.15 以降、挿入されたレコードの OID は、 CamelMongoOid キー（MongoDbConstants.OID 定数）のメッセージヘッダーに保存されます。保存された値は、単一の挿入の場合は org.bson.types.ObjectId になり ます。複数のレコードが挿入されている場合は java.util.List<org.bson.types.ObjectId > になります。

216.5.2.2. save

save 操作は upsert (UPdate, inSERT)操作と同等で、レコードが更新され、存在しない場合は 1 つの atomic 操作にすべて挿入されます。MongoDB は _id フィールドに基づいて一致を実行します。

更新の場合、オブジェクトは完全に置き換えられ、MongoDB の $modifiers の 使用が許可されないことに注意してください。したがって、オブジェクトがすでに存在する場合は、2 つのオプションがあります。

1. クエリーを実行して、オブジェクト全体とそのすべてのフィールド（効率的ではない）をすべて先に取得し、Camel 内で変更してから保存します。

2. $modifiers で update 操作を使用します。これにより、代わりにサーバー側で更新が実行されます。upsert フラグを有効にすることができます。この場合、挿入が必要な場合、MongoDB は $modifiers をフィルタークエリーオブジェクトに適用し、結果を挿入します。

以下に例を示します。

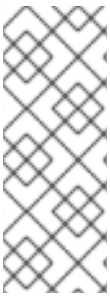
```
from("direct:insert")
   .to("mongodb:myDb?database=flights&collection=tickets&operation=save");
```

216.5.2.3. 更新

コレクションで 1 つまたは複数のレコードを更新します。正確に 2 つの要素が含まれる IN メッセージボディーとして List<DBObject> が必要です。

- 要素 1（インデックス 0）pid フィルタークエリー gitops は、通常のクエリーオブジェクトと同じように、影響を受けるオブジェクトを決定します。

- 要素 2（インデックス 1）pid 更新ルールは、一致したオブジェクトの更新方法。MongoDB の すべての修飾子操作 がサポートされます。



注記

Multiupdates。デフォルトでは、MongoDB は複数のオブジェクトがフィルタークエリーに一致する場合でも 1 つのオブジェクトのみを更新します。MongoDB に対して一致する すべて のレコードを更新するように指示するには、CamelMongoDbMultiUpdate IN メッセージヘッダーを true に設定します。

CamelMongoDbRecordsAffected キーのあるヘッダー（MongoDbConstants.RECORDS_AFFECTED 定数）と更新されたレコード（ WriteResult.getN（）から派生）を返します。

以下の IN メッセージヘッダーをサポートします。

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ
CamelMongoDbMultiUpdate	**MongoDbConstants.MULTIUPDATE**	If the update to apply all objects matching.See http://www.mongodb.org/display/DOCS/Atomic+Operations	boolean/Boolean
CamelMongoDbUpsert	**MongoDbConstants.UPSERT**	要素が存在しない場合にデータベースが要素を作成するべきかどうか	boolean/Boolean

たとえば、以下では "scientist" フィールドの値を "Darwin" に設定して、filterField フィールドが true である すべて のレコードを更新します。

```
// route: from("direct:update").to("mongodb:myDb?
database=science&collection=notableScientists&operation=update");
DBObject filterField = new BasicDBObject("filterField", true);
DBObject updateObj = new BasicDBObject("$set", new BasicDBObject("scientist", "Darwin"));
Object result = template.requestBodyAndHeader("direct:update", new Object[] {filterField,
updateObj}, MongoDbConstants.MULTIUPDATE, true);
```

216.5.3. 削除操作

216.5.3.1. remove

コレクションから一致するレコードを削除します。IN メッセージのボディーは削除フィルタークエリーとして機能し、DBObject のタイプまたは型変換可能である必要があります。
以下の例では、notableScientists コレクションでフィールド 'conditionField' が true のオブジェクトをすべて削除します。

```
// route: from("direct:remove").to("mongodb:myDb?
database=science&collection=notableScientists&operation=remove");
DBObject conditionField = new BasicDBObject("conditionField", true);
Object result = template.requestBody("direct:remove", conditionField);
```

キー CamelMongoDbRecordsAffected のあるヘッダー（MongoDbConstants.RECORDS_AFFECTED 定数）が返され、削除されたレコードの数（WriteResult.getN（）からコピーされます）が含まれます。

216.5.4. 一括書き込み操作

216.5.4.1. bulkWrite

Camel 2.21 で利用可能

実行順序を制御すると共に書き込み操作を一括して実行します。insert、update、および delete 操作のコマンド が含まれる IN メッセージボディーとして List<WriteModel<DBObject >> が必要です。

以下の例では、「scientist」フィールドの値を「Marie Curie」に設定し、ID で新しいサイエンティスト「Pierre Curie」を挿入し、ID が "3" のレコードを削除します。

```
// route: from("direct:bulkWrite").to("mongodb:myDb?
database=science&collection=notableScientists&operation=bulkWrite");
List<WriteModel<DBObject>> bulkOperations = Arrays.asList(
        new InsertOneModel<>(new BasicDBObject("scientist", "Pierre Curie")),
        new UpdateOneModel<>(new BasicDBObject("_id", "5"),
                new BasicDBObject("$set", new BasicDBObject("scientist", "Marie
```

```
Curie"))),
        new DeleteOneModel<>(new BasicDBObject("_id", "3")));

BulkWriteResult result = template.requestBody("direct:bulkWrite", bulkOperations,
BulkWriteResult.class);
```

デフォルトでは、操作は順番に実行され、リスト内で残りの書き込み操作を処理せずに最初の書き込みエラーで中断されます。MongoDB に対してリストの残りの書き込み操作を処理するよう指示するには、CamelMongoDbBulkOrdered IN メッセージヘッダーを false に設定します。順序のない操作は並行して実行され、この動作は保証されません。

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ
CamelMongoDbBulkOrdered	**MongoDbConstants.BULK_ORDERED**	順序付けされた操作または順序付けされていない操作の実行を実行します。デフォルトは true です。	boolean/Boolean

216.5.5. その他の操作

216.5.5.1. aggregate

Camel 2.14 から利用可能

指定されたパイプラインがボディーに含まれる状態で集約を実行します。集約には長い操作があり、大きな操作となる可能性があります。注意して使用してください。

```
// route: from("direct:aggregate").to("mongodb:myDb?
```

```
database=science&collection=notableScientists&operation=aggregate");
from("direct:aggregate")
    .setBody().constant("[{ $match : {$or : [{\"scientist\" : \"Darwin\"},{\"scientist\" :
\"Einstein\"}]}},{ $group: { _id: \"$scientist\", count: { $sum: 1 }} } ]")
    .to("mongodb:myDb?
database=science&collection=notableScientists&operation=aggregate")
    .to("mock:resultAggregate");
```

以下の IN メッセージヘッダーをサポートします。

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ
CamelMongoDbBatchSize	**MongoDbConstants.BATCH_SIZE**	バッチごとに返すドキュメントの数を設定します。	int/Integer
CamelMongoDbAllowDiskUse	**MongoDbConstants.ALLOW_DISK_USE**	集約パイプラインのステージを有効にして、データを一時ファイルに書き込みます。	boolean/Boolean

効率的な取得は、outputType=DBCursor でサポートされています。

また、上記のヘッダーを設定するよりも簡単なエンドポイントオプションとして outputType=DBCursor(Camel 2.21+)を含めることで、サーバーからルートへ返されたドキュメントを「ストリーム」することもできます。これにより、Mongo シェルで aggregate（）を実行しているのと同じく、Mongo ドライバーから DBCursor が操作し、ルートが結果を繰り返し処理できるようになります。デフォルトでは、このオプションがないと、このコンポーネントはドライバーのカーソルから

List にドキュメントを読み込んで、ルートに戻ります。これにより、多数のインメモリーオブジェクトが発生する可能性があります。DBCursor を使用すると、一致するドキュメントの数が尋ねられない点に注意してください。詳細は、MongoDB のドキュメントサイトを参照してください。

outputType=DBCursor および batch サイズのあるオプションの例 :

```
// route: from("direct:aggregate").to("mongodb:myDb?
database=science&collection=notableScientists&operation=aggregate");
from("direct:aggregate")
    .setHeader(MongoDbConstants.BATCH_SIZE).constant(10)
    .setBody().constant("[{ $match : {$or : [{\"scientist\" : \"Darwin\"},{\"scientist\" :
\"Einstein\"}]}},{ $group: { _id: \"$scientist\", count: { $sum: 1 }} } ]")
    .to("mongodb:myDb?
database=science&collection=notableScientists&operation=aggregate&outputType=DBCurso
r")
    .to("mock:resultAggregate");
```

216.5.5.2. getDbStats

MongoDB シェルで db.stats（）コマンドを 実行することと同等で、データベースに関する有用な統計図を表示します。
以下に例を示します。

```
> db.stats();
{
    "db" : "test",
    "collections" : 7,
    "objects" : 719,
    "avgObjSize" : 59.73296244784423,
    "dataSize" : 42948,
    "storageSize" : 1000058880,
    "numExtents" : 9,
    "indexes" : 4,
    "indexSize" : 32704,
    "fileSize" : 1275068416,
    "nsSizeMB" : 16,
    "ok" : 1
}
```

使用例 :

```
// from("direct:getDbStats").to("mongodb:myDb?
database=flights&collection=tickets&operation=getDbStats");
Object result = template.requestBody("direct:getDbStats", "irrelevantBody");
assertTrue("Result is not of type DBObject", result instanceof DBObject);
```

操作は、OUT メッセージボディーの DBObject の形式で、シェルに表示されるデータ構造を返します。

216.5.5.3. getColStats

MongoDB シェルで db.collection.stats（） コマンドを実行することと同等です。これにより、コレクションに関する有用な統計図が表示されます。
以下に例を示します。

```
> db.camelTest.stats();
{
    "ns" : "test.camelTest",
    "count" : 100,
    "size" : 5792,
    "avgObjSize" : 57.92,
    "storageSize" : 20480,
    "numExtents" : 2,
    "nindexes" : 1,
    "lastExtentSize" : 16384,
    "paddingFactor" : 1,
    "flags" : 1,
    "totalIndexSize" : 8176,
    "indexSizes" : {
        "_id_" : 8176
    },
    "ok" : 1
}
```

使用例：

```
// from("direct:getColStats").to("mongodb:myDb?
database=flights&collection=tickets&operation=getColStats");
Object result = template.requestBody("direct:getColStats", "irrelevantBody");
assertTrue("Result is not of type DBObject", result instanceof DBObject);
```

操作は、OUT メッセージボディーの DBObject の形式で、シェルに表示されるデータ構造を返します。

216.5.5.4. command

Camel 2.15 から利用可能

データベースでボディーをコマンドとして実行します。ホスト情報、レプリケーション、またはシャーディングステータスの取得に admin 操作にfull を使用します。

コレクションパラメーターはこの操作には使用されません。

```
// route: from("command").to("mongodb:myDb?database=science&operation=command");
DBObject commandBody = new BasicDBObject("hostInfo", "1");
Object result = template.requestBody("direct:command", commandBody);
```

216.5.6. 動的操作

Exchange は、MongoDbConstants.OPERATION_HEADER 定数で定義された CamelMongoDbOperation ヘッダーを設定することで、エンドポイントの固定操作を上書きできます。
サポートされる値は MongoDbOperation 列挙によって決定され、エンドポイント URI で 操作 パラメーターの許可される値に一致します。

以下に例を示します。

```
// from("direct:insert").to("mongodb:myDb?
database=flights&collection=tickets&operation=insert");
Object result = template.requestBodyAndHeader("direct:insert", "irrelevantBody",
MongoDbConstants.OPERATION_HEADER, "count");
assertTrue("Result is not of type Long", result instanceof Long);
```

216.6. TAILABLE CURSOR CONSUMER

MongoDB は、*nix システムの tail -f コマンドと同様にカーソルを開いたままにすることで、コレクションから継続中のデータを即座に消費するメカニズムを提供します。この仕組みは、クライアントが新しいデータを取得するためにスケジュールされた間隔で ping を行うのではなく、サーバーが利用可能になる時点で新しいデータをクライアントにプッシュするので、スケジュールされたポーリングよりもはるかに効率的です。そうでない場合には、冗長なネットワークトラフィックも軽減します。

テート可能なカーソルを使用する必要があるのは 1 つのみです。コレクションは「キャプチャーコレクション」である必要があります。つまり、N オブジェクトのみを保持する必要があり、制限に達すると、MongoDB は最初に挿入された順序で古いオブジェクトをフラッシュします。詳細は、http://www.mongodb.org/display/DOCS/Tailable+Cursors を参照してください。

Camel MongoDB コンポーネントはテール可能なカーソルコンシューマーを実装し、この機能を Camel ルートで使用できます。新規オブジェクトが挿入されると、MongoDB は調整可能なカーソルコンシューマーに自然な順序で DBObject としてプッシュします。これにより、Exchange に変換され、ルートロジックがトリガーされます。

216.7. 調整可能なカーソルコンシューマーの仕組み

カーソルをテール可能なカーソルに変換するには、最初にカーソルを生成する際に、いくつかの特別なフラグが MongoDB に通知されます。作成後、カーソルは開いたままとなり、新規データが到達するまで DBCursor.next（） メソッドの呼び出し時にブロックされます。ただし、MongoDB サーバーは、不確定な期間後に新規データが表示されない場合は、カーソルを終了する権利を確保します。新しいデータの消費を継続するには、カーソルを再生成する必要があります。この作業を行うには、停止した位置を覚えておく必要があります。そうでないと、最初からやり直す必要があります。

Camel MongoDB の調整可能なカーソルコンシューマーは、これらのすべてのタスクを処理します。性質を増加させるデータ内の一部のフィールドにキーを提供する必要があります。これは、再生成するたびにカーソルを置くマーカーとして機能します。たとえば、タイムスタンプ、連続 ID など。MongoDB でサポートされるデータタイプ。正常に機能するために日付、文字列、および整数が見つかります。このメカニズムをこのコンポーネントのコンテキストで呼び出します。

コンシューマーはこのフィールドの最後の値を記憶し、カーソルを再生成するたびに、reads Field > lastValue のようにフィルターを指定してクエリーを実行し、未読のデータのみが消費される ようにします。

増加フィールドの設定： エンドポイント URI tailTrackingIncreasingField オプションでの増加フィールドのキーを設定します。Camel 2.10 では、このフィールドのネストされたナビゲーションはまだサポートされていないため、データのトップレベルのフィールドでなければなりません。つまり、「timestamp」フィールドは okay ですが、「nested.timestamp」は機能しません。入れ子になった増加フィールドのサポートが必要な場合は、Camel JIRA でチケットを作成してください。

カーソルの再生成遅延： 新規データが初期時に利用できない場合、MongoDB はカーソルを即時に強制終了することです。この場合、サーバーに負荷がかからないため、（デフォルト値 1000ms） cursorRegenerationDelay オプションが導入されました。これは、ニーズに合わせて変更できます。

以下に例を示します。

```
from("mongodb:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime")
    .id("tailableCursorConsumer1")
    .autoStartup(false)
    .to("mock:test");
```

上記のルートは、「flights.cancellations」の上限付きのコレクションから消費されます。「departureTime」を増加フィールドとして使用し、デフォルトの再生成カーソル遅延は 1000ms です。

216.8. 永続的なテールトラッキング

標準的なテールトラッキングは揮発性で、最後の値はメモリーにのみ保持されます。ただし、実際には、Camel コンテナーを再起動する必要があり、その後は最後の値が失われ、調整可能なカーソルコンシューマーは上位から再度消費され、ルートに重複するレコードを送信する可能性が高くなります。

この状況に対応するには、永続的なテールトラッキング機能を有効にして、MongoDB データベース内の特別なコレクションで最後に消費された値を追跡することもできます。コンシューマーの初期設定時に、最後に追跡された値を復元し、何も起こらなかったかのように続行します。

最後の読み取り値は 2 つの状況で永続化されます。カーソルが再生成され、コンシューマーがシャットダウンするタイミングです。需要がある場合に、将来の定期的な間隔（5 秒ごとにフラッシュ）で持続することを検討してください。この機能を要求するには、Camel JIRA でチケットを作成してください。

216.9. 永続的なテールトラッキングの有効化

この機能を有効にするには、少なくともエンドポイント URI に以下のオプションを設定します。

- persistentTailTracking オプションを trueに指定
- このコンシューマーの一意の識別子への persistentId オプション。これにより、同じコレクションを多くのコンシューマーで再利用できます。

さらに、tailTrackDb オプション、tailTrackCollection オプション、および tailTrackField オプションを、ランタイム情報が保存されるカスタム化に設定できます。各オプションの説明は、このページの上部にあるエンドポイントオプションの表を参照してください。

たとえば、以下のルートは「flights.cancellations」の上限付きコレクションから消費します。「departureTime」を増加フィールドとして使用し、デフォルトの再生成カーソル遅延である 1000ms を使用し、永続的なテールトラッキングが有効になっている状態で、「flights.camelTailTracking」の「cancellationsTracker」 ID の下に保持されます。 最後に処理された値を「lastTrackingValue」フィールドに格納（camelTailTracking および lastTrackingValue はデフォルト）。

```
from("mongodb:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime&persistentTailTracking=true" +
    "&persistentId=cancellationsTracker")
  .id("tailableCursorConsumer2")
  .autoStartup(false)
  .to("mock:test");
```

以下は、上記の別の例ですが、「lastProcessedDepartureTime」フィールドの「trackers.camelTrackers」コレクションの下に永続的なテールトラッキングのランタイム情報が保存されます。

```
from("mongodb:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime&persisteNtTailTracking=true" +

"&persistentId=cancellationsTracker&tailTrackDb=trackers&tailTrackCollection=camelTrackers" +
    "&tailTrackField=lastProcessedDepartureTime")
   .id("tailableCursorConsumer3")
   .autoStartup(false)
   .to("mock:test");
```

216.10. OPLOG TAIL TRACKING

oplog コレクション追跡機能により、MongoDB の機能などのトリガーを実装できます。このコレクションを有効にするには、最初にレプリカセットをアクティベートする必要があります。このトピックの詳細については、https://docs.mongodb.com/manual/tutorial/deploy-replica-set/ を参照してください。

以下は、コンポーネントを使用して oplog コレクションを追跡する方法を実証する Java DSL ベースのルートの例になります。この特定のケースでは、データベースの optlog_test でコレクション 顧客 に影響するイベントをフィルタリングします。tailTrackIncreasingField はタイムスタンプフィールド ('ts')であり、これは TIMESTAMP 値で tailTrackingStrategy パラメーターを使用する必要があることを意味します。

```
import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import org.apache.camel.Exchange;
import org.apache.camel.Message;
import org.apache.camel.Processor;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.mongodb.MongoDBTailTrackingEnum;
import org.apache.camel.main.Main;

import java.io.InputStream;

/**
 * For this to work you need to turn on the replica set
 * <p>
 * Commands to create a replica set:
 * <p>
 * rs.initiate( {
 * _id : "rs0",
 * members: [ { _id : 0, host : "localhost:27017" } ]
 * })
 */
```

```
public class MongoDbTracker {

    private final String database;

    private final String collection;

    private final String increasingField;

    private MongoDBTailTrackingEnum trackingStrategy;

    private int persistRecords = -1;

    private boolean persistenTailTracking;

    public MongoDbTracker(String database, String collection, String increasingField) {
        this.database = database;
        this.collection = collection;
        this.increasingField = increasingField;
    }

    public static void main(String[] args) throws Exception {
        final MongoDbTracker mongoDbTracker = new MongoDbTracker("local", "oplog.rs", "ts");
        mongoDbTracker.setTrackingStrategy(MongoDBTailTrackingEnum.TIMESTAMP);
        mongoDbTracker.setPersistRecords(5);
        mongoDbTracker.setPersistenTailTracking(true);
        mongoDbTracker.startRouter();
        // run until you terminate the JVM
        System.out.println("Starting Camel. Use ctrl + c to terminate the JVM.\n");

    }

    public void setTrackingStrategy(MongoDBTailTrackingEnum trackingStrategy) {
        this.trackingStrategy = trackingStrategy;
    }

    public void setPersistRecords(int persistRecords) {
        this.persistRecords = persistRecords;
    }

    public void setPersistenTailTracking(boolean persistenTailTracking) {
        this.persistenTailTracking = persistenTailTracking;
    }

    void startRouter() throws Exception {
        // create a Main instance
        Main main = new Main();
        main.bind(MongoConstants.CONN_NAME, new MongoClient("localhost", 27017));
        main.addRouteBuilder(new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                getContext().getTypeConverterRegistry().addTypeConverter(InputStream.class,
BasicDBObject.class,
                        new MongoToInputStreamConverter());
                from("mongodb://" + MongoConstants.CONN_NAME + "?database=" + database
                        + "&collection=" + collection
                        + "&persistentTailTracking=" + persistenTailTracking
```

```
                + "&persistentId=trackerName" + "&tailTrackDb=local"
                + "&tailTrackCollection=talendTailTracking"
                + "&tailTrackField=lastTrackingValue"
                + "&tailTrackIncreasingField=" + increasingField
                + "&trackingStrategy=" + trackingStrategy.toString()
                + "&persistRecords=" + persistRecords
                + "&cursorRegenerationDelay=1000")
                .filter().jsonpath("$[?(@.ns=='optlog_test.customers')]")
                .id("logger")
                .to("log:logger?level=WARN")
                .process(new Processor() {
                    public void process(Exchange exchange) throws Exception {
                        Message message = exchange.getIn();
                        System.out.println(message.getBody().toString());
                        exchange.getOut().setBody(message.getBody().toString());
                    }
                });
            }
        });
        main.run();
    }
}
```

216.11. 型変換

camel-mongodb コンポーネントに含まれる MongoDbBasicConverters 型コンバーターは、以下の変換を提供します。

Name	タイプから	タイプ	どのように変更を加えればよいですか?
fromMapToDBObject	マップ	**DBObject**	新しい **BasicDBObject (Map m)**コンストラクターを介して新しい **BasicDBObject** を構築します。
fromBasicDBObjectToMap	**BasicDBObject**	マップ	**BasicDBObject** はすでに **マップ**を実装しています。
fromStringToDBObject	文字列	**DBObject**	uses **com.mongodb.util.JSON.parse(String s)**
fromAnyObjectToDBObject	オブジェクト	**DBObject**	Jackson ライブラリー を使用してオブジェクトを **Map** に変換します。これは、新しい **BasicDBObject**の初期化に使用されます。

この型コンバーターは自動検出されるため、手動で設定する必要はありません。

216.12. 関連項目

- *MongoDB の Web サイト*
- *NoSQL Wikipedia の記事*
- *MongoDB Java ドライバー API ドキュメント - 現在のバージョン * 使用例に対するユニットテスト*

第217章 MONGODB GRIDFS COMPONENT

Camel バージョン 2.18 から利用可能

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mongodb-gridfs</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

217.1. URI 形式

```
mongodb-gridfs:connectionBean?database=databaseName&bucket=bucketName[&moreOptions...]
```

217.2. MONGODB GRIDFS オプション

MongoDB GridFS コンポーネントにはオプションがありません。

MongoDB GridFS エンドポイントは、URI 構文を使用して設定します。

```
mongodb-gridfs:connectionBean
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

217.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
connectionBean	使用する **必須** の com.mongodb.Mongo の名前。		文字列

217.2.2. クエリーパラメーター（17 パラメーター）：

Name	説明	デフォルト	Type
バケット （共通）	データベース内の GridFS バケットの名前を設定します。デフォルトは fs です。	fs	文字列
データベース （共通）	**Required**: ターゲットに設定する MongoDB データベースの名前を設定します。		文字列
readPreference (common)	Mongo コネクションで MongoDB ReadPreference を設定します。接続に直接設定された読み取り設定は、この設定によって上書きされます。リンク com.mongodb.ReadPreferencevalueOf(String)ユーティリティーメソッドを使用して、渡された readPreference 値を解決します。使用できる値の例には、最も近い、プライマリー、またはセカンダリーなどがあります。		ReadPreference
writeConcern (common)	標準の MongoDB を使用して、MongoDB の書き込み操作用に WriteConcern を設定します。link WriteConcernvalueOf(String)メソッドを呼び出すことで、WriteConcern クラスのフィールドから解決されました。		WriteConcern
writeConcernRef (common)	MongoDB での書き込み操作用に WriteConcern を設定し、Bean ref でレジストリーに存在するカスタムの WriteConcern に渡します。また、キーを渡すことで標準の WriteConcerns を使用することもできます。link setWriteConcern(String)setWriteConcern メソッドを参照してください。		WriteConcern
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
遅延 （コンシューマー）	コンシューマー内のポーリング間の遅延を設定します。デフォルトは 500ms です。	500	Long
fileAttributeName (consumer)	QueryType が FileAttribute を使用する場合、使用される属性の名前が設定されます。デフォルトは camel-processed です。	camel-processed	文字列
initialDelay （コンシューマー）	コンシューマーがポーリングを開始する前に initialDelay を設定します。デフォルトは 1000ms です。	1000	Long

Name	説明	デフォルト	Type
persistentTSCollection (consumer)	QueryType が永続タイムスタンプを使用する場合、これは DB 内のコレクションの名前を設定し、タイムスタンプを保存します。	camel-timestamps	文字列
persistentTSObject (consumer)	QueryType が永続タイムスタンプを使用する場合、これはコレクションのオブジェクトの ID でタイムスタンプを保存します。	camel-timestamp	文字列
クエリー （コンシューマー）	GridFsConsumer でファイルの検索に使用されるクエリーの設定に使用される追加のクエリーパラメーター（JSON 内）		文字列
queryStrategy (consumer)	新規ファイルのポーリングに使用される QueryStrategy を設定します。デフォルトは Timestamp です。	TimeStamp	QueryStrategy
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
操作 （プロデューサー）	このエンドポイントが GridRS に対して実行する操作を設定します。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

217.3. SPRING XML でのデータベースの設定

以下の Spring XML は、MongoDB インスタンスへのコネクションを定義する Bean を作成します。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="mongoBean" class="com.mongodb.Mongo">
        <constructor-arg name="host" value="${mongodb.host}" />
```

```
        <constructor-arg name="port" value="${mongodb.port}" />
    </bean>
</beans>
```

217.4. サンプルルート

Spring XML で定義された以下のルートは、コレクションで操作 findOne を実行します。

GridFS からファイルを取得します。

```
<route>
  <from uri="direct:start" />
  <!-- using bean 'mongoBean' defined above -->
  <to uri="mongodb-gridfs:mongoBean?database=${mongodb.database}&amp;operation=findOne" />
  <to uri="direct:result" />
</route>
```

217.5. GRIDFS 操作： プロデューサーエンドポイント

217.5.1. count

コレクション内のファイルの合計数を返し、Integer を OUT メッセージのボディーとして返します。

```
// from("direct:count").to("mongodb-gridfs?database=tickets&operation=count");
Integer result = template.requestBodyAndHeader("direct:count", "irrelevantBody");
assertTrue("Result is not of type Long", result instanceof Integer);
```

filename ヘッダーを指定して、ファイル名に一致するファイルの数を指定できます。

```
Map<String, Object> headers = new HashMap<String, Object>();
headers.put(Exchange.FILE_NAME, "filename.txt");
Integer count = template.requestBodyAndHeaders("direct:count", query, headers);
```

217.5.2. listAll

タブ区切りのストリームにすべてのファイル名とその ID を一覧表示する Reader を返します。

```
// from("direct:listAll").to("mongodb-gridfs?database=tickets&operation=listAll");
Reader result = template.requestBodyAndHeader("direct:listAll", "irrelevantBody");

filename1.txt   1252314321
filename2.txt   2897651254
```

217.5.3. findOne

GridFS システムでファイルを見つけ、本文をコンテンツの InputStream に設定します。 メタデータにもヘッダーがあります。 受信ヘッダーから Exchange.FILE_NAME を使用して、検索するファイルを決定します。

```
// from("direct:findOne").to("mongodb-gridfs?database=tickets&operation=findOne");
Map<String, Object> headers = new HashMap<String, Object>();
headers.put(Exchange.FILE_NAME, "filename.txt");
InputStream result = template.requestBodyAndHeaders("direct:findOne", "irrelevantBody",
headers);
```

217.5.4. create

GridFs データベースに新規ファイルを作成します。名前として、受信ヘッダーの Exchange.FILE_NAME と、内容として（InputStream として）ボディーの内容を使用します。

```
// from("direct:create").to("mongodb-gridfs?database=tickets&operation=create");
Map<String, Object> headers = new HashMap<String, Object>();
headers.put(Exchange.FILE_NAME, "filename.txt");
InputStream stream = ... the data for the file ...
template.requestBodyAndHeaders("direct:create", stream, headers);
```

217.5.5. remove

GridFS データベースからファイルを削除します。

```
// from("direct:remove").to("mongodb-gridfs?database=tickets&operation=remove");
Map<String, Object> headers = new HashMap<String, Object>();
headers.put(Exchange.FILE_NAME, "filename.txt");
template.requestBodyAndHeaders("direct:remove", "", headers);
```

217.6. GRIDFS CONSUMER

関連項目

- MongoDB の Web サイト
- NoSQL Wikipedia の記事
- MongoDB Java ドライバー API ドキュメント - 現在のバージョン * 使用例に対するユニットテスト

第218章 MONGODB コンポーネント

Camel バージョン 2.19 から利用可能

注記： Camel MongoDB3 コンポーネントは Java 3.4 の Mongo ドライバーを使用します。プレビューバージョンをお探しの方は、Camel MongoDB コンポーネントを参照してください。

Wikipedia に従って、「NoSQL is a moving promote a loosely defined class of non-rational data store that breaks of an long history ofational databases and ACID guarantees.」 NoSQL ソリューションは過去数年に人気があり、Facebook、LinkedIn、Twitter などの主要サイトやサービスを利用することが知られています。これらのソリューションは、それらを広範囲に使用してスケーラビリティーや生産性を実現していることが知られています。

基本的に、NoSQL ソリューションは従来の faillock(Relational Database Management Systems)とは異なります。これは、SQL をクエリー言語として使用しておらず、一般的に ACID のようなトランザクション動作やリレーショナルデータベースデータを提供していないことです。代わりに、柔軟なデータ構造とスキーマの概念に基づいて設計されており（固定スキーマを持つデータベーステーブルの従来の概念がドロップされると）、商用ハードウェアや高速処理に極端なスケーラビリティーが発生します。

MongoDB は非常に人気のあるNoSQL ソリューションであり、camel-mongodb コンポーネントは Camel と MongoDB を統合するため、プロデューサーとして（コレクションにおける同等の操作）およびコンシューマー（MongoDB コレクションからドキュメントを多用）の両方と対話できます。

MongoDB は、（オフィスドキュメントではなく）ドキュメントの概念に関するもので、JSON/BSON で定義された階層データです。このコンポーネントページは、それらに精通していることを前提としています。そうでない場合は、http://www.mongodb.org/ にアクセスします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mongodb3</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

218.1. URI 形式

```
mongodb3:connectionBean?
database=databaseName&collection=collectionName&operation=operationName[&moreOptio
ns...]
```

218.2. MONGODB オプション

MongoDB コンポーネントにはオプションがありません。

MongoDB エンドポイントは、URI 構文を使用して設定します。

```
mongodb3:connectionBean
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

218.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
connectionBean	使用する **必須** の com.mongodb.Mongo の名前。		文字列

218.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
コレクション （共通）	このエンドポイントにバインドする MongoDB コレクションの名前を設定します。		文字列
collectionIndex (common)	コレクションインデックスを設定します(JSON FORMAT : field1 : order1, field2 : order2)。		文字列
createCollection (common)	初期中にコレクションが存在しない場合は作成します。デフォルトは true です。	true	boolean
データベース （共通）	MongoDB データベースの名前をターゲットに設定します。		文字列
操作 (common)	このエンドポイントが MongoDB に対して実行される操作を設定します。使用できる値は、MongoDbOperation を参照してください。		MongoDbOperation

Name	説明	デフォルト	Type
outputType (common)	プロデューサーの出力を、作成したタイプ DocumentList Document または MongoIterable に変換します。findAll と aggregate には DocumentList または MongoIterable が適用されます。ドキュメントが他のすべての操作に適用されます。		MongoDbOutputType
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
cursorRegenerationDelay (advanced)	MongoDB の調整可能なカーソルは、新規データが到達するまでブロックされます。新しいデータが挿入されない場合、一定の時間が経過すると、カーソルは自動的に解放され、MongoDB サーバーが閉じられます。クライアントは必要に応じてカーソルを再生成することが予想されます。この値は、新しいカーソルの取得を試みる前に待機する時間を指定します。失敗した場合は、次の試行が実行されるまでの時間を指定します。デフォルト値は 1000ms です。	1000	Long
Dynamicity （詳細）	このエンドポイントが受信エクスチェンジプロパティーからターゲットデータベースおよびコレクションを動的に解決しようとするかどうかを設定します。実行時にデータベースおよび他の静的エンドポイント URI に指定されたコレクションの上書きに使用できます。パフォーマンスを向上させるために、デフォルトでは無効になっています。有効にすると、パフォーマンスが最小限に抑えられます。	false	boolean

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
writeResultAsHeader (advanced)	書き込み操作では、OUT メッセージのボディーとして WriteResult を返すのではなく、IN メッセージを OUT に変換し、WriteResult をヘッダーとしてアタッチします。	false	boolean
persistentId (tail)	1 つのテールトラッキングコレクションでは、調整可能な複数のコンシューマーの多数のトラッカーをホストできます。それらを分離させるには、各トラッカーに固有の persistentId がなければなりません。		文字列
persistentTailTracking (tail)	永続的なテールトラッキングを有効にします。これは、システムを再起動しても最後に使用されたメッセージを追跡するメカニズムです。次回システムが起動すると、エンドポイントは最後にスライピングレコードを停止した時点からカーソルを復元します。	false	boolean
tailTrackCollection (tail)	テールトラッキング情報が永続化されるコレクション。指定されていない場合、リンク MongoDbTailTrackingConfigDEFAULT_COLLECTION がデフォルトで使用されます。		文字列
tailTrackDb (tail)	テールトラッキングメカニズムが永続化するデータベースを示します。指定されていない場合、現在のデータベースはデフォルトで選択されます。Dynamicity は有効にされても考慮されません。つまり、テールトラッキングデータベースはエンドポイントの初期化とは異なります。		文字列
tailTrackField (tail)	最後に追跡された値を配置するフィールド。指定されていない場合、リンク MongoDbTailTrackingConfigDEFAULT_FIELD がデフォルトで使用されます。		文字列
tailTrackIncreasingField (tail)	性質を引き上げる受信レコードの correlation フィールドは、生成されるたびにカーソルを配置するために使用されます。カーソルは、タイプのクエリーで作成される（TrackIncreasingField lastValue（永続テールトラッキングから復元される可能性がある））になります。Integer、Date、String などを使用できます。注記： 現在の時点でのドット表記のサポートはないため、フィールドはドキュメントのトップレベルに指定する必要があります。		文字列

MoongoDB コンポーネントのオプションに関する注意

writeConcern Remove in camel 2.19Mongo client options 「Spring XML でのデータベースの設定」 を参照してください。標準の MongoDB を使用して、MongoDB の書き込み操作用に WriteConcern を設定します。link WriteConcernvalueOf(String)メソッドを呼び出すことで、WriteConcern クラスのフィールドから解決されました。

readPreference Remove in camel 2.19（Camel 2.19 の ReadPreference Remove）Mongo client options 「Spring XML でのデータベースの設定」 を参照してください。Mongo コネクションで MongoDB ReadPreference を設定します。接続に直接設定された読み取り設定は、この設定によって上書きされます。リンク com.mongodb.ReadPreferencevalueOf(String)ユーティリティーメソッドを使用して、渡された readPreference 値を解決します。使用できる値の例は、最も近いプライマリーまたはセカンダリーなどです。

218.3. SPRING XML でのデータベースの設定

以下の Spring XML は、MongoDB インスタンスへのコネクションを定義する Bean を作成します。

mongo java ドライバー 3 以降、WriteConcern および readPreference オプションは動的に変更できません。これらは mongoClient オブジェクトで定義されます。

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:mongo="http://www.springframework.org/schema/data/mongo"
xsi:schemaLocation="http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context.xsd
      http://www.springframework.org/schema/data/mongo
      http://www.springframework.org/schema/data/mongo/spring-mongo.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">

  <mongo:mongo-client id="mongoBean" host="${mongo.url}" port="${mongo.port}"
credentials="${mongo.user}:${mongo.pass}@${mongo.dbname}">
    <mongo:client-options write-concern="NORMAL" />
  </mongo:mongo-client>
</beans>
```

218.4. サンプルルート

Spring XML で定義された以下のルートは、コレクションで操作 dbStats を実行します。

指定されたコレクションの DB 統計を取得

```
<route>
  <from uri="direct:start" />
  <!-- using bean 'mongoBean' defined above -->
  <to uri="mongodb3:mongoBean?
database=${mongodb.database}&amp;collection=${mongodb.collection}&amp;operation=getDbStats"
/>
  <to uri="direct:result" />
</route>
```

218.5. MONGODB 操作 - プロデューサーエンドポイント

218.5.1. クエリー操作

218.5.1.1. findById

この操作は、_id フィールドが IN メッセージボディーの内容と一致するコレクションから 1 つの要素のみを取得します。受信オブジェクトは、Bson タイプと同等のものをすべて使用できます。http://bsonspec.org//specification[http://bsonspec.org//specification] および http://www.mongodb.org/display/DOCS/Java+Types を参照してください。

```
from("direct:findById")
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=findById")
    .to("mock:resultFindById");
```

ヒント

オプションのパラメーター をサポートします。この操作は、フィールドフィルターの指定をサポートします。「オプションのパラメーターの指定」を参照してください。

218.5.1.2. findOneByQuery

この操作を使用して、MongoDB クエリーに一致するコレクションから要素（最初）のみを取得します。クエリーオブジェクトは CamelMongoDbCriteria ヘッダーから抽出され ます。CamelMongoDbCriteria ヘッダーが null の場合、クエリーオブジェクトはメッセージボディーを抽出されます。つまり、Bson タイプであるか、Bson に変換する必要があります。JSON String または Hashmap を使用できます。詳細は、「#Type conversions」を参照してください。MongoDB ドライバーの Filters クラスを使用できます。

クエリーのない例（コレクションのオブジェクトを返します）。

```
from("direct:findOneByQuery")
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=findOneByQuery")
    .to("mock:resultFindOneByQuery");
```

クエリーを使用する例（一致する 1 つの結果を返す）：

```
from("direct:findOneByQuery")
    .setHeader(MongoDbConstants.CRITERIA, Filters.eq("name", "Raul Kripalani"))
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=findOneByQuery")
    .to("mock:resultFindOneByQuery");
```

ヒント

オプションのパラメーター をサポートします。この操作は、フィールドの展開や sort 句の指定をサポートします。「オプションのパラメーターの指定」を参照してください。

218.5.1.3. findAll

findAll 操作は、クエリーに一致するドキュメントをすべて返します。そうでない場合は、コレクションに含まれるすべてのドキュメントが返されます。クエリーオブジェクトは CamelMongoDbCriteria ヘッダーから抽出され ます。CamelMongoDbCriteria ヘッダーが null の場合、クエリーオブジェクトはメッセージボディーを抽出されます。つまり、Bson タイプであるか、Bson に変換する必要があります。JSON String または Hashmap を使用できます。詳細は、「タイプ変換」を参照してください。

クエリーなしの例（コレクション内のすべてのオブジェクトを返します）。

```
from("direct:findAll")
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=findAll")
    .to("mock:resultFindAll");
```

クエリーを使用する例（一致するすべての結果を返します）。

```
from("direct:findAll")
    .setHeader(MongoDbConstants.CRITERIA, Filters.eq("name", "Raul Kripalani"))
    .to("mongodb3:myDb?database=flights&collection=tickets&operation=findAll")
    .to("mock:resultFindAll");
```

ページングと効率的な取得は、以下のヘッダーでサポートされます。

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ
CamelMongoDbNumToSkip	**MongoDbConstants.NUM_TO_SKIP**	カーソルの先頭にある特定の数の要素を破棄します。	int/Integer
CamelMongoDbLimit	**MongoDbConstants.LIMIT**	返される要素の数を制限します。	int/Integer
CamelMongoDbBatchSize	**MongoDbConstants.BATCH_SIZE**	1 つのバッチで返される要素の数を制限します。通常、カーソルは結果オブジェクトのバッチを取得し、ローカルに保存します。batchSize が正である場合、取得されるオブジェクトの各バッチのサイズを表します。これは、パフォーマンスを最適化し、データ転送を制限するように調整することができます。batchSize が負の値の場合、最大バッチサイズ制限（通常は 4MB）内で一致するオブジェクトの数を制限し、カーソルが閉じられます。たとえば、batchSize が -10 の場合、サーバーは最大 10 つのドキュメントを返し、4MB に収まらない数と同じものを返し、カーソルを閉じます。この機能は、ドキュメントが最大サイズ内に収まる必要がある点で limit（）とは異なり、カーソルサーバー側でリクエストを送信する必要がなくなります。バッチサイズは、カーソルが反復された後でも変更できます。この場合、次のバッチ取得に設定が適用されます。	int/Integer

outputType=MongoIterable および batch size オプションを使用する例：

```
from("direct:findAll")
   .setHeader(MongoDbConstants.BATCH_SIZE).constant(10)
   .setHeader(MongoDbConstants.CRITERIA, Filters.eq("name", "Raul Kripalani"))
   .to("mongodb3:myDb?
database=flights&collection=tickets&operation=findAll&outputType=MongoIterable")
   .to("mock:resultFindAll");
```

findAll 操作は、ページングを使用している場合に結果ページを繰り返し処理できるように、以下の OUT ヘッダーも返します。

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	データ型
CamelMongoDbResultTotalSize	**MongoDbConstants.RESULT_TOTAL_SIZE**	クエリーに一致するオブジェクトの数。これには制限や省略は行われません。	int/Integer
CamelMongoDbResultPageSize	**MongoDbConstants.RESULT_PAGE_SIZE**	クエリーに一致するオブジェクトの数。これには制限や省略は行われません。	int/Integer

ヒント

オプションのパラメーター をサポートします。この操作は、フィールドの展開や sort 句の指定をサポートします。「オプションのパラメーターの指定 」を参照してください。

218.5.1.4. count

コレクション内のオブジェクトの合計数を返し、Long を OUT メッセージボディーとして返します。
以下の例は、「dynamicCollectionName」コレクション内のレコード数をカウントします。動的性が有効であるか、その結果、操作は「notableScientists」コレクションに対しては実行されませんが、「dynamicCollectionName」コレクションに対しては実行されません。

```
// from("direct:count").to("mongodb3:myDb?
database=tickets&collection=flights&operation=count&dynamicity=true");
Long result = template.requestBodyAndHeader("direct:count", "irrelevantBody",
MongoDbConstants.COLLECTION, "dynamicCollectionName");
assertTrue("Result is not of type Long", result instanceof Long);
```

クエリー オブジェクトは CamelMongoDbCriteria ヘッダーから抽出されるクエリー を指定できます。CamelMongoDbCriteria ヘッダーが null の場合、クエリーオブジェクトは抽出メッセージのボディーを抽出されます。つまり、タイプは Bson か、Bson に変換する必要があります。操作では、この基準に一致するドキュメントの量を返します。

```
Document query = ...
Long count = template.requestBodyAndHeader("direct:count", query,
MongoDbConstants.COLLECTION, "dynamicCollectionName");
```

218.5.1.5. フィールドフィルター（プロジェクト）の指定

クエリー操作は、デフォルトで（すべてのフィールドを含む）一致するオブジェクトを完全に返します。ドキュメントが大きい場合に、フィールドのサブセットのみを取得する必要がある場合は、CamelMongoDbFieldsProjection ヘッダー、MongoDbFieldsProjection ヘッダー、MongoDbFieldsProjection ヘッダー、MongoDbConstants.FIELDS_PROJECTION などの関連する Bson （または型変換）を設定して、すべてのクエリー操作でフィールドフィルターを指定できます。

以下は、MongoDB の Projections を使用して Bson の作成を単純化する例です。これは、_id および boringField を除くすべてのフィールドを取得します。

```
// route: from("direct:findAll").to("mongodb3:myDb?
database=flights&collection=tickets&operation=findAll")
Bson fieldProjection = Projection.exclude("_id", "boringField");
Object result = template.requestBodyAndHeader("direct:findAll", ObjectUtils.NULL,
MongoDbConstants.FIELDS_PROJECTION, fieldProjection);
```

以下は、MongoDB の Projections を使用して Bson の作成を単純化する例です。これは、_id および boringField を除くすべてのフィールドを取得します。

```
// route: from("direct:findAll").to("mongodb3:myDb?
database=flights&collection=tickets&operation=findAll")
Bson fieldProjection = Projection.exclude("_id", "boringField");
Object result = template.requestBodyAndHeader("direct:findAll", ObjectUtils.NULL,
MongoDbConstants.FIELDS_PROJECTION, fieldProjection);
```

218.5.1.6. sort 句の指定

MongoDB の Sort を使用して Bson の作成を単純化する特定のフィールドによるソートに基づいて、コレクションから min/max レコードを取得する要件がよくあります。これは、_id および boringField を除くすべてのフィールドを取得します。

```
// route: from("direct:findAll").to("mongodb3:myDb?
database=flights&collection=tickets&operation=findAll")
Bson sorts = Sorts.descending("_id");
Object result = template.requestBodyAndHeader("direct:findAll", ObjectUtils.NULL,
MongoDbConstants.SORT_BY, sorts);
```

Camel ルートでは、SORT_BY ヘッダーを findOneByQuery 操作とともに使用して、同じ結果を得ることができます。FIELDS_PROJECTION ヘッダーも指定された場合、操作は別のコンポーネント

に直接渡すことができる単一のフィールド/値のペアを返します（パラメーター化された MyBatis SELECT クエリーなど）。以下の例は、コレクションから一時的な最新ドキュメントを取得し、documentTimestamp フィールドに基づいて結果を 1 つのフィールドに減らす方法を示しています。

```
.from("direct:someTriggeringEvent")
.setHeader(MongoDbConstants.SORT_BY).constant(Sorts.descending("documentTimestamp"
))
.setHeader(MongoDbConstants.FIELDS_PROJECTION).constant(Projection.include("documen
tTimestamp"))
.setBody().constant("{}")
.to("mongodb3:myDb?
database=local&collection=myDemoCollection&operation=findOneByQuery")
.to("direct:aMyBatisParameterizedSelect")
;
```

218.5.2. 作成/更新操作

218.5.2.1. insert

IN メッセージボディーから取得した MongoDB コレクションに新しいオブジェクトを挿入します。型変換は、Document または List に変換します。
1 つの挿入と複数の挿入の 2 つのモードがサポートされます。複数の挿入の場合、エンドポイントは、すべてのタイプのオブジェクトの List、Array、または Collections を想定し ます。例:

```
from("direct:insert")
   .to("mongodb3:myDb?database=flights&collection=tickets&operation=insert");
```

操作は WriteResult を返します。WriteConcern または invokeGetLastError オプションの値によっては、getLastError（） が呼び出されました。書き込み操作の最終的な結果にアクセスするには、WriteResult で getLastError（）または getCachedLastError （） を呼び出して CommandResult を取得する必要があります。その後、CommandResult. ok（）、CommandResult. getErrorMessage（）、および/または CommandResult.getException ( )を呼び出すことで結果を確認できます。

新しいオブジェクトの _id は、コレクションで一意である必要があることに注意してください。値を指定しない場合、MongoDB は自動的に生成します。しかし、指定して一意でない場合は、挿入操作は失敗します（Camel が通知するには、callGetLastError を有効にするか、書き込み結果を待つ WriteConcern を設定する必要があります）。

これはコンポーネントの制限ではありませんが、MongoDB でスループットを向上する方法です。カスタムの _id を使用している場合は、一意のアプリケーションレベル（およびこれが適切なプラクティスである）で確認できることが想定されます。

挿入されたレコードの OID は、 CamelMongoOid キー（MongoDbConstants.OID 定数）のメッセージヘッダーに保存されます。保存された値は、単一の挿入の場合は org.bson.types.ObjectId になり ます。複数のレコードが挿入されている場合は java.util.List<org.bson.types.ObjectId > になります。

MongoDB Java Driver 3.x では、insertOne および insertMany 操作が void を返します。Camel 挿入操作は、挿入されたドキュメントの Document または List を返します。必要に応じて、各ドキュメントが新しい OID によって更新されることに注意してください。

218.5.2.2. save

save 操作は upsert (UPdate, inSERT)操作と同等で、レコードが更新され、存在しない場合は 1 つの atomic 操作にすべて挿入されます。MongoDB は _id フィールドに基づいて一致を実行します。

更新の場合、オブジェクトは完全に置き換えられ、MongoDB の $modifiers の 使用が許可されないことに注意してください。したがって、オブジェクトがすでに存在する場合は、2 つのオプションがあります。

1. クエリーを実行して、オブジェクト全体とそのすべてのフィールド（効率的ではない）をすべて先に取得し、Camel 内で変更してから保存します。

2. $modifiers で update 操作を使用します。これにより、代わりにサーバー側で更新が実行されます。upsert フラグを有効にすることができます。この場合、挿入が必要な場合、MongoDB は $modifiers をフィルタークエリーオブジェクトに適用し、結果を挿入します。

保存されるドキュメントに _id 属性が含まれていない場合、操作は挿入され、新しい _id が作成される新しい _id は CamelMongoOid ヘッダーに配置されます。

以下に例を示します。

```
from("direct:insert")
  .to("mongodb3:myDb?database=flights&collection=tickets&operation=save");
```

218.5.2.3. 更新

コレクションで 1 つまたは複数のレコードを更新します。フィルタークエリーと更新ルールが必要です。

MongoDBConstants.CRITERIA ヘッダーを Bson として定義し、更新ルールを Body に Bson として定義できます。

2 つ目の方法は、完全 2 つの要素が含まれる IN メッセージボディーとして List<Bson> を必要とする方法です。

- *要素 1（インデックス 0）pid フィルタークエリー gitops は、通常のクエリーオブジェクトと同じように、影響を受けるオブジェクトを決定します。*
- *要素 2（インデックス 1）pid 更新ルールは、一致したオブジェクトの更新方法。MongoDB の すべての修飾子操作 がサポートされます。*

注記

Multiupdates。デフォルトでは、MongoDB は複数のオブジェクトがフィルタークエリーに一致する場合でも 1 つのオブジェクトのみを更新します。MongoDB に対して一致する すべて のレコードを更新するように指示するには、CamelMongoDbMultiUpdate IN メッセージヘッダーを true に設定します。

CamelMongoDbRecordsAffected キーのあるヘッダー（MongoDbConstants.RECORDS_AFFECTED 定数）と更新されたレコード（ WriteResult.getN（）から派生）を返します。

以下の IN メッセージヘッダーをサポートします。

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ
CamelMongoDbMultiUpdate	**MongoDbConstants.MULTIUPDATE**	If the update to apply all objects matching.See http://www.mongodb.org/display/DOCS/Atomic+Operations	boolean/Boolean
CamelMongoDbUpsert	**MongoDbConstants.UPSERT**	要素が存在しない場合にデータベースが要素を作成するべきかどうか	boolean/Boolean

たとえば、以下では "scientist" フィールドの値を "Darwin" に設定して、filterField フィールドが true である すべて のレコードを更新します。

```
// route: from("direct:update").to("mongodb3:myDb?database=science&collection=notableScientists&operation=update");
Bson filterField = Filters.eq("filterField", true);
String updateObj = Updates.set("scientist", "Darwin");
Object result = template.requestBodyAndHeader("direct:update", new Bson[] {filterField, Document.parse(updateObj)}, MongoDbConstants.MULTIUPDATE, true);
```

```
// route: from("direct:update").to("mongodb3:myDb?database=science&collection=notableScientists&operation=update");
```

```
Maps<String, Object> headers = new HashMap<>(2);
headers.add(MongoDbConstants.MULTIUPDATE, true);
headers.add(MongoDbConstants.FIELDS_FILTER, Filters.eq("filterField", true));
String updateObj = Updates.set("scientist", "Darwin");;
Object result = template.requestBodyAndHeaders("direct:update", updateObj, headers);
```

```
// route: from("direct:update").to("mongodb3:myDb?
database=science&collection=notableScientists&operation=update");
String updateObj = "[{\"filterField\": true}, {\"$set\", {\"scientist\", \"Darwin\"}}]";
Object result = template.requestBodyAndHeader("direct:update", updateObj,
MongoDbConstants.MULTIUPDATE, true);
```

218.5.3. 削除操作

218.5.3.1. remove

コレクションから一致するレコードを削除します。IN メッセージのボディーは削除フィルタークエリーとして機能し、DBObject のタイプまたは型変換可能である必要があります。
以下の例では、notableScientists コレクションでフィールド 'conditionField' が true のオブジェクトをすべて削除します。

```
// route: from("direct:remove").to("mongodb3:myDb?
database=science&collection=notableScientists&operation=remove");
Bson conditionField = Filters.eq("conditionField", true);
Object result = template.requestBody("direct:remove", conditionField);
```

キー CamelMongoDbRecordsAffected のあるヘッダー（MongoDbConstants.RECORDS_AFFECTED 定数）が返され、削除されたレコードの数（WriteResult.getN（）からコピーされます）が含まれます。

218.5.4. 一括書き込み操作

218.5.4.1. bulkWrite

Camel 2.21 で利用可能

実行順序を制御すると共に書き込み操作を一括して実行します。insert、update、および delete 操作のコマンド が含まれる IN メッセージボディーとして List<WriteModel<Document >> が必要です。

以下の例では、「scientist」フィールドの値を「Marie Curie」に設定し、ID で新しいサイエンティスト「Pierre Curie」を挿入し、ID が "3" のレコードを削除します。

```
// route: from("direct:bulkWrite").to("mongodb:myDb?
```

```
database=science&collection=notableScientists&operation=bulkWrite");
List<WriteModel<Document>> bulkOperations = Arrays.asList(
        new InsertOneModel<>(new Document("scientist", "Pierre Curie")),
        new UpdateOneModel<>(new Document("_id", "5"),
                new Document("$set", new Document("scientist", "Marie Curie"))),
        new DeleteOneModel<>(new Document("_id", "3")));

BulkWriteResult result = template.requestBody("direct:bulkWrite", bulkOperations,
BulkWriteResult.class);
```

デフォルトでは、操作は順番に実行され、リスト内で残りの書き込み操作を処理せずに最初の書き込みエラーで中断されます。MongoDB に対してリストの残りの書き込み操作を処理するよう指示するには、CamelMongoDbBulkOrdered IN メッセージヘッダーを false に設定します。順序のない操作は並行して実行され、この動作は保証されません。

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ
CamelMongoDbBulkOrdered	**MongoDbConstants.BULK_ORDERED**	順序付けされた操作または順序付けされていない操作の実行を実行します。デフォルトは true です。	boolean/Boolean

218.5.5. その他の操作

218.5.5.1. aggregate

指定されたパイプラインがボディーに含まれる状態で集約を実行します。集約には長い操作があり、大きな操作となる可能性があります。注意して使用してください。

```
// route: from("direct:aggregate").to("mongodb3:myDb?
```

```
database=science&collection=notableScientists&operation=aggregate");
List<Bson> aggregate = Arrays.asList(match(or(eq("scientist", "Darwin"), eq("scientist",
        group("$scientist", sum("count", 1))));
from("direct:aggregate")
    .setBody().constant(aggregate)
    .to("mongodb3:myDb?
database=science&collection=notableScientists&operation=aggregate")
    .to("mock:resultAggregate");
```

以下の IN メッセージヘッダーをサポートします。

ヘッダーキー	クイック定数	説明（MongoDB API ドキュメントから抽出）	想定されるタイプ
CamelMongoDbBatchSize	**MongoDbConstants.BATCH_SIZE**	バッチごとに返すドキュメントの数を設定します。	int/Integer
CamelMongoDbAllowDiskUse	**MongoDbConstants.ALLOW_DISK_USE**	集約パイプラインのステージを有効にして、データを一時ファイルに書き込みます。	boolean/Boolean

効率的な取得は outputType=MongoIterable でサポートされています。

また、上記のヘッダーを設定するよりも簡単なエンドポイントオプションとして outputType=DBCursor(Camel 2.21+)を含めることで、サーバーからルートへ返されたドキュメントを「ストリーム」することもできます。これにより、Mongo シェルで aggregate（）を実行しているのと同じく、Mongo ドライバーから DBCursor が操作し、ルートが結果を繰り返し処理できるようにな

ります。デフォルトでは、このオプションがないと、このコンポーネントはドライバーのカーソルから List にドキュメントを読み込んで、ルートに戻ります。これにより、多数のインメモリーオブジェクトが発生する可能性があります。DBCursor を使用すると、一致するドキュメントの数が尋ねられない点に注意してください。詳細は、MongoDB のドキュメントサイトを参照してください。

outputType=MongoIterable および batch size オプションの例：

```
// route: from("direct:aggregate").to("mongodb3:myDb?
database=science&collection=notableScientists&operation=aggregate&outputType=MongoIte
rable");
List<Bson> aggregate = Arrays.asList(match(or(eq("scientist", "Darwin"), eq("scientist",
      group("$scientist", sum("count", 1)));
from("direct:aggregate")
   .setHeader(MongoDbConstants.BATCH_SIZE).constant(10)
   .setBody().constant(aggregate)
   .to("mongodb3:myDb?
database=science&collection=notableScientists&operation=aggregate&outputType=MongoIte
rable")
   .to("mock:resultAggregate");
```

218.5.5.2. getDbStats

MongoDB シェルで db.stats（）コマンドを 実行することと同等で、データベースに関する有用な統計図を表示します。
以下に例を示します。

```
> db.stats();
{
   "db" : "test",
   "collections" : 7,
   "objects" : 719,
   "avgObjSize" : 59.73296244784423,
   "dataSize" : 42948,
   "storageSize" : 1000058880,
   "numExtents" : 9,
   "indexes" : 4,
   "indexSize" : 32704,
   "fileSize" : 1275068416,
   "nsSizeMB" : 16,
   "ok" : 1
}
```

使用例：

```
// from("direct:getDbStats").to("mongodb3:myDb?
database=flights&collection=tickets&operation=getDbStats");
Object result = template.requestBody("direct:getDbStats", "irrelevantBody");
```

```
assertTrue("Result is not of type Document", result instanceof Document);
```

操作は、OUT メッセージボディーの Document の形式で、シェルに表示されるデータ構造を返します。

218.5.5.3. getColStats

MongoDB シェルで db.collection.stats（） コマンドを実行することと同等です。これにより、コレクションに関する有用な統計図が表示されます。
以下に例を示します。

```
> db.camelTest.stats();
{
    "ns" : "test.camelTest",
    "count" : 100,
    "size" : 5792,
    "avgObjSize" : 57.92,
    "storageSize" : 20480,
    "numExtents" : 2,
    "nindexes" : 1,
    "lastExtentSize" : 16384,
    "paddingFactor" : 1,
    "flags" : 1,
    "totalIndexSize" : 8176,
    "indexSizes" : {
        "_id_" : 8176
    },
    "ok" : 1
}
```

使用例：

```
// from("direct:getColStats").to("mongodb3:myDb?
database=flights&collection=tickets&operation=getColStats");
Object result = template.requestBody("direct:getColStats", "irrelevantBody");
assertTrue("Result is not of type Document", result instanceof Document);
```

操作は、OUT メッセージボディーの Document の形式で、シェルに表示されるデータ構造を返します。

218.5.5.4. command

データベースでボディーをコマンドとして実行します。ホスト情報、レプリケーション、またはシャーディングステータスの取得に admin 操作にfull を使用します。

コレクションパラメーターはこの操作には使用されません。

```
// route: from("command").to("mongodb3:myDb?database=science&operation=command");
DBObject commandBody = new BasicDBObject("hostInfo", "1");
Object result = template.requestBody("direct:command", commandBody);
```

218.5.6. 動的操作

Exchange は、MongoDbConstants.OPERATION_HEADER 定数で定義された CamelMongoDbOperation ヘッダーを設定することで、エンドポイントの固定操作を上書きできます。
サポートされる値は MongoDbOperation 列挙によって決定され、エンドポイント URI で 操作 パラメーターの許可される値に一致します。

以下に例を示します。

```
// from("direct:insert").to("mongodb3:myDb?
database=flights&collection=tickets&operation=insert");
Object result = template.requestBodyAndHeader("direct:insert", "irrelevantBody",
MongoDbConstants.OPERATION_HEADER, "count");
assertTrue("Result is not of type Long", result instanceof Long);
```

218.6. TAILABLE CURSOR CONSUMER

MongoDB は、*nix システムの tail -f コマンドと同様にカーソルを開いたままにすることで、コレクションから継続中のデータを即座に消費するメカニズムを提供します。この仕組みは、クライアントが新しいデータを取得するためにスケジュールされた間隔で ping を行うのではなく、サーバーが利用可能になる時点で新しいデータをクライアントにプッシュするので、スケジュールされたポーリングよりもはるかに効率的です。そうでない場合には、冗長なネットワークトラフィックも軽減します。

テート可能なカーソルを使用する必要があるのは 1 つのみです。コレクションは「キャプチャーコレクション」である必要があります。つまり、N オブジェクトのみを保持する必要があり、制限に達すると、MongoDB は最初に挿入された順序で古いオブジェクトをフラッシュします。詳細は、http://www.mongodb.org/display/DOCS/Tailable+Cursors を参照してください。

Camel MongoDB コンポーネントはテール可能なカーソルコンシューマーを実装し、この機能を Camel ルートで使用できます。新規オブジェクトが挿入されると、MongoDB は調整可能なカーソルコンシューマーに Document としてプッシュします。これにより、Exchange に変換され、ルートロジックがトリガーされます。

218.7. 調整可能なカーソルコンシューマーの仕組み

カーソルをテール可能なカーソルに変換するには、最初にカーソルを生成する際に、いくつかの特別なフラグが MongoDB に通知されます。作成後、カーソルは開いたままとなり、新規データが到達するまで MongoCursor.next（） メソッドの呼び出し時にブロックされます。ただし、MongoDB サーバーは、不確定な期間後に新規データが表示されない場合は、カーソルを終了する権利を確保します。新しいデータの消費を継続するには、カーソルを再生成する必要があります。この作業を行うには、停止した位置を覚えておく必要があります。そうでないと、最初からやり直す必要があります。

Camel MongoDB の調整可能なカーソルコンシューマーは、これらのすべてのタスクを処理します。性質を増加させるデータ内の一部のフィールドにキーを提供する必要があります。これは、再生成するたびにカーソルを置くマーカーとして機能します。たとえば、タイムスタンプ、連続 ID など。MongoDB でサポートされるデータタイプ。正常に機能するために日付、文字列、および整数が見つかります。このメカニズムをこのコンポーネントのコンテキストで呼び出します。

コンシューマーはこのフィールドの最後の値を記憶し、カーソルを再生成するたびに、reads Field > lastValue のようにフィルターを指定してクエリーを実行し、未読のデータのみが消費される ようにします。

増加フィールドの設定： エンドポイント URI tailTrackingIncreasingField オプションでの増加フィールドのキーを設定します。Camel 2.10 では、このフィールドのネストされたナビゲーションはまだサポートされていないため、データのトップレベルのフィールドでなければなりません。つまり、「timestamp」フィールドは okay ですが、「nested.timestamp」は機能しません。入れ子になった増加フィールドのサポートが必要な場合は、Camel JIRA でチケットを作成してください。

カーソルの再生成遅延： 新規データが初期時に利用できない場合、MongoDB はカーソルを即時に強制終了することです。この場合、サーバーに負荷がかからないため、（デフォルト値 1000ms） cursorRegenerationDelay オプションが導入されました。これは、ニーズに合わせて変更できます。

以下に例を示します。

```
from("mongodb3:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime")
    .id("tailableCursorConsumer1")
    .autoStartup(false)
    .to("mock:test");
```

上記のルートは、「flights.cancellations」の上限付きのコレクションから消費されます。「departureTime」を増加フィールドとして使用し、デフォルトの再生成カーソル遅延は 1000ms です。

218.8. 永続的なテールトラッキング

標準的なテールトラッキングは揮発性で、最後の値はメモリーにのみ保持されます。ただし、実際には、Camel コンテナーを再起動する必要があり、その後は最後の値が失われ、調整可能なカーソルコンシューマーは上位から再度消費され、ルートに重複するレコードを送信する可能性が高くなります。

この状況に対応するには、永続的なテールトラッキング機能を有効にして、MongoDB データベース内の特別なコレクションで最後に消費された値を追跡することもできます。コンシューマーの初期設定時に、最後に追跡された値を復元し、何も起こらなかったかのように続行します。

最後の読み取り値は 2 つの状況で永続化されます。カーソルが再生成され、コンシューマーがシャットダウンするタイミングです。需要がある場合に、将来の定期的な間隔（5 秒ごとにフラッシュ）で持続することを検討してください。この機能を要求するには、Camel JIRA でチケットを作成してください。

218.9. 永続的なテールトラッキングの有効化

この機能を有効にするには、少なくともエンドポイント URI に以下のオプションを設定します。

- persistentTailTracking オプションを trueに指定
- このコンシューマーの一意の識別子への persistentId オプション。これにより、同じコレクションを多くのコンシューマーで再利用できます。

さらに、tailTrackDb オプション、tailTrackCollection オプション、および tailTrackField オプションを、ランタイム情報が保存されるカスタム化に設定できます。各オプションの説明は、このページの上部にあるエンドポイントオプションの表を参照してください。

たとえば、以下のルートは「flights.cancellations」の上限付きコレクションから消費します。「departureTime」を増加フィールドとして使用し、デフォルトの再生成カーソル遅延である 1000ms を使用し、永続的なテールトラッキングが有効になっている状態で、「flights.camelTailTracking」の「cancellationsTracker」 ID の下に保持されます。 最後に処理された値を「lastTrackingValue」フィールドに格納（camelTailTracking および lastTrackingValue はデフォルト）。

```
from("mongodb3:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime&persistentTailTracking=true" +
    "&persistentId=cancellationsTracker")
  .id("tailableCursorConsumer2")
  .autoStartup(false)
  .to("mock:test");
```

以下は、上記の別の例ですが、「lastProcessedDepartureTime」フィールドの「trackers.camelTrackers」コレクションの下に永続的なテールトラッキングのランタイム情報が保存されます。

```
from("mongodb3:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime&persistentTailTracking=true" +

"&persistentId=cancellationsTracker&tailTrackDb=trackers&tailTrackCollection=camelTrackers" +
    "&tailTrackField=lastProcessedDepartureTime")
  .id("tailableCursorConsumer3")
  .autoStartup(false)
  .to("mock:test");
```

218.10. 型変換

camel-mongodb コンポーネントに含まれる MongoDbBasicConverters 型コンバーターは、以下の変換を提供します。

Name	タイプから	タイプ	どのように変更を加えればよいですか?
fromMapToDocument	**マップ**	**Document**	新しい **Document (Map m)**コンストラクターで新しいドキュメントを構築します。
fromDocumentToMap	**Document**	**マップ**	**ドキュメントは** すでにマップを実装して**い** ます。
fromStringToDocument	**文字列**	**Document**	uses **com.mongodb.Document.parse(String s)**.
fromAnyObjectToDocument	**オブジェクト**	**Document**	Jackson ライブラリー を使用してオブジェクトを **Map** に変換し、新しい**ドキュメントを初期化するために使用されます。**
fromStringToList	**文字列**	**List<Bson>**	**org.bson.codecs.configuration.CodecRegistries** を使用して BsonArray および List<Bson> に変換します。

この型コンバーターは自動検出されるため、手動で設定する必要はありません。

218.11. 関連項目

- *MongoDB の Web サイト*
- *NoSQL Wikipedia の記事*
- *MongoDB Java ドライバー API ドキュメント - 現在のバージョン * 使用例に対するユニットテスト*

第219章 MQTT コンポーネント

Camel バージョン 2.10 で利用可能

mqtt: コンポーネントは、Apache ActiveMQ や Mosquitto などの MQTT 準拠のメッセージブローカーとの通信に使用されます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mqtt</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

219.1. URI 形式

```
mqtt://name[?options]
```

name は、コンポーネントを割り当てる名前に置き換えます。

219.2. オプション

MQTT コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
ホスト（共通）	接続する MQTT ブローカーの URI - このコンポーネントは SSL もサポートします（例：ssl://127.0.0.1:8883）。		文字列
userName (security)	MQTT ブローカーに対する認証に使用するユーザー名		文字列
パスワード（セキュリティー）	MQTT ブローカーに対する認証に使用するパスワード		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

MQTT エンドポイントは、URI 構文を使用して設定します。

```
mqtt:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

219.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	トピック名ではない論理名が **必要** です。		文字列

219.2.2. クエリーパラメーター（39 パラメーター）：

Name	説明	デフォルト	Type
blockingExecutor (common)	SSL 接続は、setBlockingExecutor メソッドを呼び出して代わりに使用するエグゼキューターを設定する場合を除き、内部スレッドプールに対してブロッキング操作を実行します。		エグゼキューター (Executor)
byDefaultRetain (common)	MQTT ブローカーに送信されたメッセージで使用されるデフォルトの保持ポリシー	false	boolean
cleanSession (common)	MQTT サーバーがトピックサブスクリプションを永続化させ、クライアントセッション全体でのack の位置を保持する場合は false に設定します。デフォルトは true です。	false	boolean

Name	説明	デフォルト	Type
clientId (common)	セッションのクライアント ID を設定するのに使用します。これは、setCleanSession(false)が使用されているセッションを識別するために MQTT サーバーが使用するものです。id は 23 文字以下である必要があります。デフォルトは、自動生成される ID です（ソケットアドレス、ポート、およびタイムスタンプに基づきます）。		文字列
connectAttemptsMax (common)	クライアントがサーバーに接続するために最初の試行時にクライアントにエラーが報告するまでの最大再接続試行回数。無制限の試行を使用するには、-1 に設定します。デフォルトは -1 です。	-1	Long
connectWaitInSeconds (common)	コンポーネントが MQTT ブローカーへの接続を確立するまで待機する遅延（秒単位）	10	int
disconnectWaitInSeconds (common)	MQTT ブローカーから stop（）の有効な切断についてコンポーネントが待機する秒数	5	int
dispatchQueue (common)	HawtDispatch ディスパッチキューは、接続へのアクセスを同期するために使用されます。明示的なキューが setDispatchQueue メソッドで設定されていない場合、接続に新しいキューが作成されます。同期のために複数の接続が同じキューを共有する場合は、明示的なキューを設定すると便利です。		DispatchQueue
ホスト （共通）	接続する MQTT ブローカーの URI - このコンポーネントは SSL もサポートします（例：ssl://127.0.0.1:8883）。	tcp://127.0.0.1:1883	URI
keepAlive (common)	Keep Alive タイマーを秒単位で設定します。クライアントから受信されるメッセージの間隔を定義します。長い TCP/IP タイムアウトを待つことなく、サーバーへのネットワーク接続がドロップしたことをサーバーが検出できるようになります。		short
localAddress (common)	使用するローカルの InetAddress およびポート		URI
maxReadRate (common)	このトランスポートがデータを受信する 1 秒あたりの最大バイトを設定します。この設定では、レートを超えないように、読み取りをスロットルします。デフォルトは 0 で、スロットリングが無効になります。		int

Name	説明	デフォルト	Type
maxWriteRate (common)	このトランスポートがデータを送信する最大バイトを設定します。この設定では、レートが超過しないように書き込みをスロットルします。デフォルトは 0 で、スロットリングが無効になります。		int
mqttQosPropertyName (common)	個別の公開されたメッセージの Exchange を検索するプロパティー名。これが設定されている場合（AtMostOnce、AtLeastOnce、または ExactlyOnce のいずれか）、その QoS は MQTT メッセージブローカーに送信されたメッセージに設定されます。	MQTT Qos	文字列
mqttRetainPropertyName (common)	個別の公開されたメッセージの Exchange を検索するプロパティー名。これが設定されている場合（ブール値が必要）- retain プロパティーは MQTT メッセージブローカーに送信されたメッセージに設定されます。	MQTT Retain	文字列
mqttTopicPropertyName (common)	パブリッシュするエクスチェンジで検索されるこれらのプロパティー	MQTT TopicProperty Name	文字列
publishTopicName (common)	メッセージのパブリッシュに使用するデフォルトのトピック	camel/mqtt/test	文字列
qualityOfService (common)	トピックに使用するサービスレベル	AtLeastOnce	文字列
receiveBufferSize (common)	内部ソケット受信バッファーのサイズを設定します。デフォルトは 65536(64k)です。	65536	int
reconnectAttemptsMax (common)	サーバー接続が以前に確立された後にエラーがクライアントに報告するまでの最大再接続試行回数。無制限の試行を使用するには、-1 に設定します。デフォルトは -1 です。	-1	Long
reconnectBackOffMultiplier (common)	再接続試行間で使用されるExponential バックオフ。指数バックオフを無効にするには、1 に設定します。デフォルトは 2 に設定されます。	2.0	double
reconnectDelay (common)	最初の再接続試行までの待機時間（ミリ秒単位）。デフォルトは10です。	10	Long
reconnectDelayMax (common)	再接続試行まで待機する最大時間（ミリ秒単位）。デフォルトは 30,000 です。	30000	Long

Name	説明	デフォルト	Type
sendBufferSize (common)	内部ソケット送信バッファーのサイズを設定します。デフォルトは 65536(64k)です。	65536	int
sendWaitInSeconds (common)	例外が発生する前に、コンポーネントが MQTT ブローカーからの受信を待機する最大時間。	5	int
sslContext (common)	SSLContext 設定を使用してセキュリティーを設定するには、以下を行います。		SSLContext
subscribeTopicName (common)	**非推奨** 化。これらは、MQTT から継承されたプロパティーとともに、エンドポイントで設定されます。		文字列
subscribeTopicNames (common)	メッセージをサブスクライブするトピックのカンマ区切りの一覧。この一覧の各項目には、階層内の特定のパターンに一致するトピックをサブスクライブするために MQTT ワイルドカード（または）を含めることができます。たとえば、階層内のレベルにあるすべてのトピックのワイルドカードであるため、ブローカーにトピック/トピックおよびトピック/2 がある場合は、topics/ を使用して両方をサブスクライブすることができます。ここで考慮すべき注意点として、ブローカーがトピック/3を追加した場合、ルートはそのトピックからメッセージ受信も開始することです。		文字列
trafficClass (common)	トランスポートから送信されるパケットの IP ヘッダーに、トラフィッククラスまたは type-of-service オクテットを設定します。デフォルトは 8 で、トラフィックがスループットに対して最適化される必要があることを意味します。	8	int
バージョン (common)	MQTT バージョン 3.1.1 を使用するには 3.1.1 に設定します。それ以外の場合は、デフォルトは 3.1 プロトコルバージョンです。	3.1	文字列
willMessage (common)	送信する Will メッセージ。デフォルトはゼロ長メッセージです。		文字列
willQos （共通）	Will メッセージに使用する QoS を設定します。デフォルトは AT_MOST_ONCE です。	AtMostOnce	QoS
willRetain (common)	Will を retain オプションを指定して公開するには、true に設定します。		QoS
willTopic (common)	クライアントの設定により、クライアントに予期しない接続が解除されている場合には、サーバーの Will メッセージを指定されたトピックに公開します。		文字列

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
lazySessionCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合に、例外を回避するためにセッションを遅延的に作成できます。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

219.3. サンプル

メッセージの送信：

```
from("direct:foo").to("mqtt:cheese?publishTopicName=test.mqtt.topic");
```

メッセージの消費：

```
from("mqtt:bar?
subscribeTopicName=test.mqtt.topic").transform(body().convertToString()).to("mock:result")
```

219.4. エンドポイント

Camel は Endpoint インターフェースを使用した Message Endpoint パターンをサポートします。

通常、エンドポイントは Component および Endpoints によって作成されます。これは通常、URI 経由で DSL で参照されます。

エンドポイントから以下のメソッドを使用できます。

- createProducer() は、メッセージエクスチェンジをエンドポイントに送信する プロデューサー を作成します。
- createConsumer() は、コンシューマーの作成時に、プロセッサー を介してエンドポイントからメッセージエクスチェンジを消費する Event Driven Consumerパターンを実装します。
- createPollingConsumer() は、PollingConsumer経由でエンドポイントからメッセージエクスチェンジを消費するための Polling Consumer パターンを実装します。

219.5. 関連項目

- Configuring Camel (Camel の設定)
- Message Endpoint パターン
- URI
- コンポーネントの作成

第220章 MSV コンポーネント

Camel バージョン 1.1 で利用可能

MSV コンポーネントは、MSV ライブラリー と、サポートされる XML スキーマ言語（XML Schema または RelaxNG XML Syntax など）を使用してメッセージボディーの XML 検証を実行します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-msv</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Jing コンポーネントは RelaxNG Compact 構文もサポートしていることに注意してください。

220.1. URI 形式

```
msv:someLocalOrRemoteResource[?options]
```

someLocalOrRemoteResource は、クラスパス上のローカルリソースへの URL の一部、またはファイルシステムのリモートリソースまたはリソースに対する完全な URL です。たとえば、以下のようになります。

```
msv:org/foo/bar.rng
msv:file:../foo/bar.rng
msv:http://acme.com/cheese.rng
```

URI にクエリーオプションを追加するには、?option=value&option=value&...

220.2. オプション

MSV コンポーネントは以下の 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
schemaFactory（詳細）	javax.xml.validation.SchemaFactory を使用します。		SchemaFactory
resourceResolverFactory (advanced)	動的エンドポイントリソース URI に依存するカスタムの LSResourceResolver を使用する場合		ValidatorResourceResolverFactory
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

MSV エンドポイントは URI 構文を使用します。

```
msv:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

220.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceUri	クラスパス上のローカルリソースへの URL、またはレジストリーで Bean を検索する参照、または検証する XSD が含まれるファイルシステムのリモートリソースまたはリソースへの完全 **な** URL。		文字列

220.2.2. クエリーパラメーター（11 パラメーター）：

Name	説明	デフォルト	Type
failOnNullBody (producer)	ボディーが存在しない場合に失敗するかどうか。	true	boolean
failOnNullHeader (producer)	ヘッダーに対して検証時にヘッダーが存在しない場合に失敗するかどうか。	true	boolean
headerName (producer)	メッセージボディーではなくヘッダーに対して検証するには、以下を行います。		文字列

Name	説明	デフォルト	Type
errorHandler (advanced)	カスタムの org.apache.camel.processor.validation.ValidatorErrorHandler を使用します。デフォルトのエラーハンドラーはエラーをキャプチャーし、例外をスローします。		ValidatorErrorHandler
resourceResolver (advanced)	カスタムの LSResourceResolver を使用するには、以下を行います。resourceResolverFactory と併用しないでください。		LSResourceResolver
resourceResolverFactory (advanced)	動的エンドポイントリソース URI に依存するカスタムの LSResourceResolver を使用するには、以下を行います。デフォルトのリソースリゾルバーファクトリーは、クラスパスとファイルシステムからファイルを読み取ることができるリソースリゾルバーです。resourceResolver と併用しないでください。		ValidatorResourceResolverFactory
schemaFactory （詳細）	カスタムの javax.xml.validation.SchemaFactory を使用するには、以下を実行します。		SchemaFactory
schemaLanguage (advanced)	W3C XML Schema Namespace URI を設定します。	http://www.w3.org/2001/XMLSchema	文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
useDom (advanced)	DOMSource/DOMResult または SaxSource/SaxResult がバリデーターによって使用されるかどうか。	false	boolean
useSharedSchema (advanced)	Schema インスタンスを共有すべきかどうか。このオプションは、JDK 1.6.x のバグを回避するために導入されています。Xerces にこの問題はありません。	true	boolean

220.3. 例

以下の 例 は、エンドポイント direct:start からのルートを設定する方法を示しています。次に、XML が指定の RelaxNG XML スキーマ に一致するかどうかに基づいて、2 つのエンドポイントのいずれかに移動します（クラスパスで提供されます）。

220.4. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第221章 MUSTACHE コンポーネント

Camel バージョン 2.12 から利用可能

mustache: コンポーネントは Mustache テンプレートを使用したメッセージの処理を許可します。これは、Templating を使用してリクエストの応答を生成する場合に便利です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
<groupId>org.apache.camel</groupId>
<artifactId>camel-mustache</artifactId>
<version>x.x.x</version> <!-- use the same version as your Camel core version -->
</dependency>
```

221.1. URI 形式

```
mustache:templateName[?options]
```

templateName は、呼び出すテンプレートのクラスパスローカル URI、またはリモートテンプレートの完全な URL（例： file:///folder/myfile.mustache）です。

URI にクエリーオプションを追加するには、?option=value&option=value&...

221.2. オプション

Mustache コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
mustacheFactory (advanced)	カスタム MustacheFactory を使用するには、以下を実行します。		MustacheFactory
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Mustache エンドポイントは URI 構文を使用して設定されます。

```
mustache:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

221.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceUri	リソースへの **必須** パス。プレフィックス：classpath、file、http、ref、または bean. classpath、file、http は、これらのプロトコルを使用してリソースをロードします（classpath はデフォルト）。ref はレジストリーのリソースを検索します。Bean はリソースとして使用される Bean でメソッドを呼び出します。Bean には、ドットの後にメソッド名を指定できます（例：bean:myBean.myMethod）。		文字列

221.2.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
encoding （プロデューサー）	リソースコンテンツの文字エンコーディング。		文字列
endDelimiter (producer)	テンプレートコードの終端をマークするために使用される文字。	}}	文字列
startDelimiter (producer)	テンプレートコードの開始をマークするために使用される文字。	{{	文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

221.3. MUSTACHE コンテキスト

Camel は Mustache コンテキスト（マップのみ）で交換情報を提供し ます。エクスチェンジは以下の ように転送されます。

key	value
exchange	エクスチェンジ **自体**。
exchange.properties	エクスチェンジ **プロパティー**。
ヘッダー	In メッセージのヘッダー。
camelContext	Camel コンテキスト。
request	In メッセージ。
ボディー	In メッセージのボディー。
response	Out メッセージ（InOut メッセージ交換パターンのみ）。

221.4. 動的テンプレート

Camel は 2 つのヘッダーを提供し、テンプレートまたはテンプレートコンテンツ自体に異なるリソースの場所を定義できます。これらのヘッダーのいずれかが設定されている場合、Camel はこれを設定されたエンドポイントで使用します。これにより、ランタイム時に動的テンプレートを提供できます。

ヘッダー	Type	説明	バージョンのサポート

ヘッダー	Type	説明	バージョンのサポート
MustacheConstants.MUSTACHE_RESOURCE_URI	文字列	設定されたエンドポイントの代わりに使用するテンプレートリソースのURI。	
MustacheConstants.MUSTACHE_TEMPLATE	文字列	設定されたエンドポイントの代わりに使用するテンプレート。	

221.5. サンプル

たとえば、以下のようなものを使用できます。

```
from("activemq:My.Queue").
to("mustache:com/acme/MyResponse.mustache");
```

Mustache テンプレートを使用して InOut メッセージエクスチェンジのメッセージの応答（JMSReplyTo ヘッダーがある）を作成するには、以下を実行します。

InOnly を使用してメッセージを消費し、別の宛先に送信する場合は、以下を使用できます。

```
from("activemq:My.Queue").
to("mustache:com/acme/MyResponse.mustache").
to("activemq:Another.Queue");
```

コンポーネントがヘッダーで動的に使用する必要のあるテンプレートを指定できます。以下に例を示します。

```
from("direct:in").
setHeader(MustacheConstants.MUSTACHE_RESOURCE_URI).constant("path/to/my/template.
mustache").
to("mustache:dummy");
```

221.6. メールサンプル

この例では、注文確認メールに Mustache テンプレートを使用します。メールテンプレートは、以下のように Mustache でレイアウトされます。

```
Dear {{headers.lastName}}}, {{headers.firstName}}

Thanks for the order of {{headers.item}}.

Regards Camel Riders Bookstore
{{body}}
```

221.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第222章 MVEL コンポーネント

Camel バージョン 2.12 から利用可能

mvel: コンポーネントを使用すると、MVEL テンプレートを使用してメッセージを処理できます。これは、Templating を使用してリクエストの応答を生成する場合に便利です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mvel</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

222.1. URI 形式

```
mvel:templateName[?options]
```

templateName は、呼び出すテンプレートのクラスパスローカル URI、またはリモートテンプレートの完全な URL（例： file://folder/myfile.mvel）です。

URI にクエリーオプションを追加するには、?option=value&option=value&...

222.2. オプション

MVEL コンポーネントにはオプションがありません。

MVEL エンドポイントは、URI 構文を使用して設定します。

```
mvel:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

222.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceUri	リソースへの **必須** パス。プレフィックス：classpath、file、http、ref、または bean. classpath、file、http は、これらのプロトコルを使用してリソースをロードします（classpath はデフォルト）。ref はレジストリーのリソースを検索します。Bean はリソースとして使用される Bean でメソッドを呼び出します。Bean には、ドットの後にメソッド名を指定できます（例：bean:myBean.myMethod）。		文字列

222.2.2. クエリーパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
encoding （プロデューサー）	リソースコンテンツの文字エンコーディング。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

222.3. メッセージヘッダー

mvel コンポーネントはメッセージに 2 つのヘッダーを設定します。

ヘッダー	説明
CamelMvelResourceUri	templateName を **String** オブジェクトとする。

222.4. MVEL コンテキスト

Camel は MVEL コンテキストで交換情報を提供します（マップのみ ）。エクスチェンジは 以下 のように転送されます。

key	value
exchange	エクスチェンジ **自体**。
exchange.properties	エクスチェンジ **プロパティー**。
ヘッダー	In メッセージのヘッダー。
camelContext	Camel コンテキストの説明。
request	In メッセージ。
in	In メッセージ。
ボディー	In メッセージのボディー。
out	Out メッセージ（InOut メッセージ交換パターンのみ）。
response	Out メッセージ（InOut メッセージ交換パターンのみ）。

222.5. ホットリロード

mvel テンプレートリソースは、デフォルトではファイルとクラスパスリソース（展開 jar）の両方でホットリロードが可能です。contentCache=true を設定すると、Camel はリソースを一度だけロードするため、ホットリロードはできません。リソースが変更しない場合には、このシナリオを実稼働環境で使用することができます。

222.6. 動的テンプレート

Camel は 2 つのヘッダーを提供し、テンプレートまたはテンプレートコンテンツ自体に異なるリソースの場所を定義できます。これらのヘッダーのいずれかが設定されている場合、Camel はこれを設定されたエンドポイントで使用します。これにより、ランタイム時に動的テンプレートを提供できます。

ヘッダー	Type	説明
Camel MvelResourceUri	文字列	設定されたエンドポイントの代わりに使用するテンプレートリソースの URI。
Camel MvelTemplate	文字列	設定されたエンドポイントの代わりに使用するテンプレート。

222.7. サンプル

たとえば、以下のようなものを使用できます。

```
from("activemq:My.Queue").
  to("mvel:com/acme/MyResponse.mvel");
```

MVEL テンプレートを使用して InOut メッセージエクスチェンジのメッセージへの応答（JMSReplyTo ヘッダーがある）

コンポーネントがヘッダー経由で動的に使用する必要のあるテンプレートを指定するには、以下を実行します。

```
from("direct:in").
  setHeader("CamelMvelResourceUri").constant("path/to/my/template.mvel").
  to("mvel:dummy");
```

テンプレートをヘッダーとして直接指定するには、以下のように、コンポーネントはヘッダー経由で動的に使用する必要があります。

```
from("direct:in").
  setHeader("CamelMvelTemplate").constant("@{\"The result is \" + request.body * 3}\" }").
  to("velocity:dummy");
```

222.8. 関連項目

- *Configuring Camel (Camel の設定)*

- *コンポーネント*
- *エンドポイント*
- *はじめに*

第223章 MVEL 言語

Camel バージョン 2.0 で利用可能

Camel では、Mvel を Expression または Xml 設定として使用できます。

Mvel を使用して、メッセージフィルターまたは Recipient List の式として述語を作成できます。

Mvel ドット表記を使用して操作を呼び出すことができます。たとえば、getFamiliyName メソッドを持つ POJO が含まれるボディーがある場合は、以下のように構文を作成できます。

```
"request.body.familyName"
  // or
"getRequest().getBody().getFamilyName()"
```

223.1. MVEL オプション

MVEL 言語は、以下に示す 1 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
trim	**true**	**ブール値**	値をトリミングして先頭および末尾の空白と改行を削除するかどうか。

223.2. 変数

変数	型	説明
this	エクスチェンジ	Exchange はルートオブジェクトです。
exchange	エクスチェンジ	Exchange オブジェクト
exception	Throwable	エクスチェンジの例外 (ある場合)

変数	型	説明
exchangeId	文字列	エクスチェンジ ID
fault	メッセージ	Fault メッセージ（ある場合）
request	メッセージ	exchange.in メッセージ
response	メッセージ	exchange.out メッセージ（ある場合）
properties	マップ	エクスチェンジプロパティー
property(name)	オブジェクト	指定の名前のプロパティー
property(name, type)	Type	指定の名前のプロパティー（指定のタイプ）

223.3. サンプル

たとえば、XML の メッセージフィルター 内で Mvel を使用できます。

```
<route>
  <from uri="seda:foo"/>
  <filter>
    <mvel>request.headers.foo == 'bar'</mvel>
    <to uri="seda:bar"/>
  </filter>
</route>
```

Java DSL を使用した例を以下に示します。

```
from("seda:foo").filter().mvel("request.headers.foo == 'bar'").to("seda:bar");
```

223.4. 外部リソースからのスクリプトの読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、「classpath:」、"file:"、または "http: " などのリソースから Camel に読み込むことができます。
これは、「 resource:scheme:location」 構文を使用して行われます。たとえば、実行可能なクラスパスのファイルを参照します。

```
.setHeader("myHeader").mvel("resource:classpath:script.mvel")
```

223.5. 依存関係

camel ルートで Mvel を使用するには、Mvel 言語を実装する camel-mvel の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mvel</artifactId>
  <version>x.x.x</version>
</dependency>
```

第224章 MYBATIS コンポーネント

Camel バージョン 2.7 で利用可能

mybatis: コンポーネントを使用すると、MyBatis を使用してリレーショナルデータベース内のデータのクエリー、ポーリング、挿入、更新、および削除を実行できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mybatis</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

224.1. URI 形式

```
mybatis:statementName[?options]
```

statementName は、評価するクエリー、挿入、更新、または削除操作にマップする MyBatis XML マッピングファイルのステートメント名です。

URI にクエリーオプションを追加するには、?option=value&option=value&…

このコンポーネントはデフォルトで、SqlMapConfig.xml の名前でクラスパスのルートから MyBatis SqlMapConfig ファイルを読み込みます。
ファイルが別の場所にある場合は、MyBatisComponent コンポーネントで configurationUri オプションを設定する必要があります。

224.2. オプション

MyBatis コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
sqlSessionFactory (advanced)	SqlSessionFactory を使用するには、以下を実行します。		SqlSessionFactory
configurationUri (common)	MyBatis xml 設定ファイルの場所。デフォルト値は、クラスパスからロードされた SqlMapConfig.xml です。	SqlMapConfig.xml	文字列
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

MyBatis エンドポイントは URI 構文を使用して設定します。

```
mybatis:statement
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

224.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
statement	**必須**: クエリー、挿入、更新、または評価する操作にマッピングされる MyBatis XML マッピングファイルのステートメント名。		文字列

224.2.2. クエリーパラメーター（29 パラメーター）：

Name	説明	デフォルト	Type
outputHeader (common)	クエリー結果をメッセージのボディーの代わりにヘッダーに保存します。デフォルトでは、outputHeader == null とクエリー結果はメッセージボディーに保存され、メッセージボディー内の既存のコンテンツは破棄されます。outputHeader が設定されている場合、値はクエリー結果を保存するヘッダーの名前として使用され、元のメッセージボディーは保持されます。また、outputHeader を設定すると、デフォルトの CamelMyBatisResult ヘッダーの設定が省略されます。		文字列

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
inputHeader (consumer)	ユーザー は、メッセージのボディーではなく入力パラメーターのヘッダー値です。デフォルトでは、inputHeader == null と入力パラメーターはメッセージボディーから取得されます。outputHeader が設定されている場合、値が使用され、クエリーパラメーターがボディーではなくヘッダーから取得されます。		文字列
maxMessagesPerPoll (consumer)	このオプションは、データベースプールによって返される結果をバッチに分割し、複数のエクスチェンジに配信することを目的としています。この整数は単一のエクスチェンジで配信する最大メッセージを定義します。デフォルトでは、最大値は設定されません。1000 などの制限を設定して、数千のファイルがあるサーバーを起動すると回避できます。無効にするには、値を 0 または負の値に設定します。	0	int
onConsume (consumer)	ルート内でデータが処理された後に実行するステートメント		文字列
routeEmptyResultSet (consumer)	空の結果セットを次のホップにルーティングするかどうか。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
トランザクション (コンシューマー)	トランザクションを有効または無効にします。有効にすると、エクスチェンジの処理に失敗した場合、コンシューマーは追加のエクスチェンジの処理を中断してロールバック Eager をトリガーします。	false	boolean
useIterator (consumer)	プロセス結果セットを個別に、またはリストとして設定	true	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processingStrategy (consumer)	カスタムの MyBatisProcessingStrategy の使用		MyBatisProcessing Strategy
executorType (producer)	ステートメントの実行中に使用されるエクゼキュータータイプ。simple - executor は特別なものではありません。- executor は準備済みステートメントを再利用します。batch - エグゼキューターはステートメントを再利用し、更新をバッチ処理します。	SIMPLE	ExecutorType
statementType (producer)	プロデューサーが呼び出す操作の種類を制御するために必須です。		StatementType
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int

Name	説明	デフォルト	Type
遅延（スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

224.3. メッセージヘッダー

Camel は結果メッセージ（IN または OUT のいずれか）に、使用するステートメントが含まれるヘッ

ダーを入力します。

ヘッダー	Type	説明
CamelMyBatisStatementName	**文字列**	使用される **statementName** （例： insertAccount）。
CamelMyBatisResult	**オブジェクト**	いずれかの操作で MtBatis から返される **応答**。たとえば、**INSERT** は自動生成されたキーまたは行数を返す可能性があります。

224.4. メッセージボディー

MyBatis からの応答は、SELECT ステートメントの場合、ボディーとしてのみ設定されます。たとえば、INSERT ステートメントの場合、Camel はボディーを置き換えません。これにより、ルーティングを継続し、元のボディーを維持できます。MyBatis からの応答は常に CamelMyBatisResult キーでヘッダーに保存されます。

224.5. サンプル

たとえば、JMS キューから Bean を消費してデータベースに挿入する場合は、以下を行うことができます。

```
from("activemq:queue:newAccount").
  to("mybatis:insertAccount?statementType=Insert");
```

呼び出す操作の種類を Camel に指示する必要があるため、statementType を指定する必要があります。

insertAccount は、SQL マッピングファイルの MyBatis ID に置き換えます。

```
<!-- Insert example, using the Account parameter class -->
<insert id="insertAccount" parameterType="Account">
  insert into ACCOUNT (
    ACC_ID,
    ACC_FIRST_NAME,
    ACC_LAST_NAME,
    ACC_EMAIL
```

```
    )
    values (
      #{id}, #{firstName}, #{lastName}, #{emailAddress}
    )
  </insert>
```

224.6. STATEMENTTYPE を使用した MYBATIS の制御の強化

MyBatis エンドポイントへのルーティング時により詳細な制御を行い、SQL ステートメントが SELECT、UPDATE、DELETE、または INSERT であるかを制御できます。たとえば、IN ボディーに SELECT ステートメントのパラメーターが含まれる MyBatis エンドポイントにルーティングする場合は、以下を行うことができます。

上記のコードで MyBatis ステートメント selectAccountById を呼び出すことができ、IN ボディーには Integer タイプなどの取得するアカウント ID が含まれる必要があります。

SelectList などの他の操作で同じ操作を行うことができます。

Account オブジェクトを IN 本文として MyBatis に送信できる UPDATE も同様です。

224.6.1. InsertList StatementType の使用

Camel 2.10 で利用可能

mybatis を使用すると、for-each batch ドライバーを使用して複数の行を挿入できます。これを使用するには、マッパー XML ファイルで <foreach> を使用する必要があります。以下に例を示します。

その後、以下のように InsertList ステートメントタイプを使用する mybatis エンドポイントに Camel メッセージを送信することで、複数の行を挿入できます。

224.6.2. UpdateList StatementType の使用

Camel 2.11 から利用可能

mybatis を使用すると、各バッチドライバーを使用して複数の行を更新できます。これを使用するには、マッパー XML ファイルで <foreach> を使用する必要があります。以下に例を示します。

```
<update id="batchUpdateAccount" parameterType="java.util.Map">
    update ACCOUNT set
    ACC_EMAIL = #{emailAddress}
    where
    ACC_ID in
    <foreach item="Account" collection="list" open="(" close=")" separator=",">
        #{Account.id}
    </foreach>
</update>
```

その後、以下のように UpdateList ステートメントタイプを使用する mybatis エンドポイントに Camel メッセージを送信することで、複数の行を更新できます。

```
from("direct:start")
    .to("mybatis:batchUpdateAccount?statementType=UpdateList")
    .to("mock:result");
```

224.6.3. DeleteList StatementType の使用

Camel 2.11 から利用可能

mybatis を使用すると、各バッチドライバーを使用して複数の行を削除できます。これを使用するには、マッパー XML ファイルで <foreach> を使用する必要があります。以下に例を示します。

```
<delete id="batchDeleteAccountById" parameterType="java.util.List">
    delete from ACCOUNT
    where
    ACC_ID in
    <foreach item="AccountID" collection="list" open="(" close=")" separator=",">
        #{AccountID}
    </foreach>
</delete>
```

次に、以下のように Camel メッセージを DeleteList ステートメントタイプを使用する mybatis エンドポイントに送信することで、複数の行を削除できます。

```
from("direct:start")
    .to("mybatis:batchDeleteAccount?statementType=DeleteList")
    .to("mock:result");
```

224.6.4. InsertList、UpdateList、および DeleteList StatementTypes に注意

任意のタイプのパラメーター（List、Map など）を mybatis に渡すことができます。エンドユーザーは、mybatis 動的クエリー 機能を利用して、必要に応じてこれを処理します。

224.6.5. スケジュールされたポーリングの例

このコンポーネントはスケジュールされたポーリングをサポートするため、ポーリングコンシューマーとして使用できます。たとえば、1 分ごとにデータベースをポーリングするには、以下のコマンドを実行します。

```
from("mybatis:selectAllAccounts?delay=60000").to("activemq:queue:allAccounts");
```

その他のオプションは、「Polling Consumer Options」の「ScheduledPollConsumer Options」を参照してください。

または、Timer や Quartz コンポーネントなどのスケジュールされたポーリングをトリガーする別のメカニズムを使用することもできます。 以下の例では、Timer コンポーネントを使用して 30 秒ごとにデータベースをポーリングし、データを JMS キューに送信します。

```
from("timer://pollTheDatabase?
delay=30000").to("mybatis:selectAllAccounts").to("activemq:queue:allAccounts");
```

使用される MyBatis SQL マッピングファイル：

```
<!-- Select with no parameters using the result map for Account class. -->
<select id="selectAllAccounts" resultMap="AccountResult">
  select * from ACCOUNT
</select>
```

224.6.6. 消費時の使用

このコンポーネントは、Camel によるデータの消費および処理 後 のステートメントの実行をサポートします。これにより、データベースで更新を投稿できます。すべてのステートメントが UPDATE ステートメントである必要があることに注意してください。Camel は、名前がコンマで区切られた複数のステートメントの実行をサポートします。

以下のルートは、use Account ステートメントデータが処理される方法を示しています。これにより、データベースの行のステータスを処理できるように変更できるため、2 回以上の使用を回避することができます。

また、sqlmap ファイルのステートメントは次のとおりです。

224.6.7. トランザクションへの参加

camel-mybatis でトランザクションマネージャーを設定することは、標準の MyBatis SqlMapConfig.xml ファイル外でデータベース設定を外部化する必要があるため、ほとんど少し複雑です。

最初の部分では DataSource の設定が必要です。通常、これは Spring プロキシーでラップする必要があるプール（DBCP または c3p0）です。このプロキシーにより、Spring 以外のデータソースの使用が Spring トランザクションに参加できるようになります（MyBatis SqlSessionFactory はこれのみを行います）。

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.TransactionAwareDataSourceProxy">
    <constructor-arg>
        <bean class="com.mchange.v2.c3p0.ComboPooledDataSource">
            <property name="driverClass" value="org.postgresql.Driver"/>
            <property name="jdbcUrl" value="jdbc:postgresql://localhost:5432/myDatabase"/>
            <property name="user" value="myUser"/>
            <property name="password" value="myPassword"/>
        </bean>
    </constructor-arg>
</bean>
```

これは、プロパティープレースホルダーを使用してデータベース設定を外部化できるようにするもう 1 つの利点があります。

その後、トランザクションマネージャーは、外側の DataSource を管理するように設定されます。

```
<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

mybatis-spring SqlSessionFactoryBean は、同じ DataSource をラップします。

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <!-- standard mybatis config file -->
<property name="configLocation" value="/META-INF/SqlMapConfig.xml"/>
    <!-- externalised mappers -->
<property name="mapperLocations" value="classpath*:META-INF/mappers/**/*.xml"/>
</bean>
```

次に、camel-mybatis コンポーネントはそのファクトリーで設定されます。

```
<bean id="mybatis" class="org.apache.camel.component.mybatis.MyBatisComponent">
  <property name="sqlSessionFactory" ref="sqlSessionFactory"/>
</bean>
```

最後に、トランザクションポリシーはトランザクションマネージャーの上部に定義されます。これは通常通り使用できます。

```
<bean id="PROPAGATION_REQUIRED"
class="org.apache.camel.spring.spi.SpringTransactionPolicy">
  <property name="transactionManager" ref="txManager"/>
  <property name="propagationBehaviorName" value="PROPAGATION_REQUIRED"/>
</bean>

<camelContext id="my-model-context" xmlns="http://camel.apache.org/schema/spring">
  <route id="insertModel">
    <from uri="direct:insert"/>
    <transacted ref="PROPAGATION_REQUIRED"/>
    <to uri="mybatis:myModel.insert?statementType=Insert"/>
  </route>
</camelContext>
```

224.7. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第225章 NAGIOS コンポーネント

Camel バージョン 2.3 の時点で利用可能

Nagios コンポーネントを使用すると、パッシブチェックを Nagios に送信できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-nagios</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

225.1. URI 形式

```
nagios://host[:port][?Options]
```

Camel は、Nagios コンポーネントに 2 つの機能を提供しています。メッセージをエンドポイントに送信することで、パッシブチェックメッセージを送信できます。
Camel は EventNotifer も提供します。これにより、通知を Nagios に送信できます。

225.2. オプション

Nagios コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有 NagiosConfiguration の使用		NagiosConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Nagios エンドポイントは、URI 構文を使用して設定します。

```
nagios:host:port
```

*以下の **path** パラメーターおよびクエリーパラメーターを使用します。*

225.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
host	**これ** は、チェックが送信される Nagios ホストのアドレスです。		文字列
port	**必須**: ホストのポート番号。		int

225.2.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
connectionTimeout (producer)	接続のタイムアウト（ミリ秒単位）。	5000	int
sendSync (producer)	パッシブチェックの送信時に同期を使用するかどうか。false に設定すると、Camel はメッセージのルーティングを継続し、パッシブチェックメッセージは非同期で送信されます。	true	boolean
タイムアウト （プロデューサー）	タイムアウトの送信（ミリ秒単位）。	5000	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
暗号化 （セキュリティー）	暗号化方法を指定します。		暗号化
encryptionMethod (security)	暗号化方法を指定するため **非推奨**。		NagiosEncryptionMethod
パスワード （セキュリティー）	Nagios へのチェックの送信時に認証されるパスワード。		文字列

225.3. メッセージの送信例

メッセージペイロードにメッセージが含まれる Nagios にメッセージを送信できます。デフォルトでは、OK レベルになり、CamelContext 名をサービス名として使用します。これらの値は、上記のようにヘッダーを使用して上書きすることができます。

たとえば、以下のように Hello Nagios メッセージを Nagios に送信します。

```
template.sendBody("direct:start", "Hello Nagios");

from("direct:start").to("nagios:127.0.0.1:5667?password=secret").to("mock:result");
```

CRITICAL メッセージを送信するには、以下のようなヘッダーを送信します。

```
Map headers = new HashMap();
headers.put(NagiosConstants.LEVEL, "CRITICAL");
headers.put(NagiosConstants.HOST_NAME, "myHost");
headers.put(NagiosConstants.SERVICE_NAME, "myService");
template.sendBodyAndHeaders("direct:start", "Hello Nagios", headers);
```

225.4. USING NAGIOSEVENTNOTIFER

Nagios コンポーネントは、イベントを Nagios に送信するのに使用できる EventNotifer も提供します。たとえば、以下のように Java からこれを有効にします。

```
NagiosEventNotifier notifier = new NagiosEventNotifier();
notifier.getConfiguration().setHost("localhost");
notifier.getConfiguration().setPort(5667);
notifier.getConfiguration().setPassword("password");

CamelContext context = ...
context.getManagementStrategy().addEventNotifier(notifier);
return context;
```

Spring XML では、EventNotifier タイプで Spring Bean を定義することや、Camel はこれを記載どおりに選択します。Spring XML では、Spring を使用した CamelContext の高度な設定 です。

225.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- ***エンドポイント***

- ***はじめに***

第226章 NAT コンポーネント

Camel バージョン 2.17 から利用可能

NATS は、高速で信頼性の高いメッセージングプラットフォームです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-nats</artifactId>
  <!-- use the same version as your Camel core version -->
  <version>x.y.z</version>
</dependency>
```

226.1. URI 形式

```
nats:servers[?options]
```

ここで、サーバー は NATS サーバーのリストを表します。

226.2. オプション

revokets コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

エンドポイントは、URI 構文を使用して設定します。

nats:servers

以下の path パラメーターおよびクエリーパラメーターを使用します。

226.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
サーバー	1 つ以上の NAT サーバーに **必要な** URL。複数のサーバーを指定する場合は、コンマを使用して URL を区切ります。		文字列

226.2.2. クエリーパラメーター（22 パラメーター）：

Name	説明	デフォルト	Type
flushConnection (common)	接続をフラッシュするかどうかを定義します。	false	boolean
flushTimeout （共通）	フラッシュタイムアウトの設定	1000	int
maxReconnectAttempts (common)	最大再接続試行数	3	int
noRandomizeServers (common)	接続試行のサーバーの順序をランダム化するかどうか	false	boolean
pedantic (common)	pedantic モードで実行されているかどうか（これは performace に影響します）	false	boolean
pingInterval (common)	接続が有効であるかどうかを認識する ping 間隔（ミリ秒単位）	4000	int
reconnect (common)	再接続機能を使用するかどうか	true	boolean
reconnectTimeWait (common)	再接続を試行するまでの待機時間（ミリ秒単位）	2000	int
トピック (common)	**必要な** トピックの名前		文字列
詳細 （共通）	詳細モードで実行するかどうか	false	boolean

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
maxMessages（コンシューマー）	maxMessages の後にサブスクライブしているトピックからメッセージの受信を停止する		文字列
poolSize (consumer)	コンシューマープールサイズ	10	int
queueName (consumer)	キュー設定に nats を使用している場合はキュー名。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
replySubject (producer)	応答を送信するサブスクライバーの件名		文字列
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
セキュア（セキュリティー）	TLS が必要であることを示す secure オプションの設定	false	boolean
ssl（セキュリティー）	SSL を使用するかどうか	false	boolean
sslContextParameters (security)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters
tlsDebug (security)	TLS Debug。コンソール出力を追加します。	false	boolean

226.3. HEADERS

Name	タイプ	説明
CamelNatsMessageTimestamp	Long	消費されるメッセージのタイムスタンプ。
CamelNatsSubscriptionId	整数	コンシューマーのサブスクリプション ID。

プロデューサーの例：

```
from("direct:send").to("nats://localhost:4222?topic=test");
```

コンシューマーの例：

```
from("nats://localhost:4222?
topic=test&maxMessages=5&queueName=test").to("mock:result");
```

第227章 NETTY コンポーネント（非推奨）

Camel バージョン 2.3 の時点で利用可能

警告

このコンポーネントは非推奨になりました。Netty4 を使用する必要があります。

Camel の netty コンポーネントは Netty プロジェクトをベースとしたソケット通信コンポーネントです。

Netty は NIO クライアントサーバーフレームワークで、プロトコルサーバーやクライアントなどのネットワークアプリケーションを迅速かつ簡単に開発できます。
Netty は、TCP、UDP ソケットサーバーなどのネットワークプログラミングを大幅に単純化し、合理化します。

この Camel コンポーネントは、プロデューサーとコンシューマーエンドポイントの両方をサポートします。

Netty コンポーネントには複数のオプションがあり、数多くの TCP/UDP 通信パラメーター（バッファーサイズ、keepAlives、tcpNoDelay など）を細かく制御でき、Camel ルート上の In-Only 通信と In-Out 通信の両方を容易にします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-netty</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

227.1. URI 形式

netty コンポーネントの URI スキームは以下のようになります。

```
netty:tcp://localhost:99999[?options]
netty:udp://remotehost:99999/[?options]
```

このコンポーネントは、TCP と UDP の両方のプロデューサーおよびコンシューマーエンドポイントをサポートします。

URI にクエリーオプションを追加するには、?option=value&option=value&...

227.2. オプション

Netty コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	エンドポイントの作成時に NettyConfiguration を設定として使用します。		NettyConfiguration
maximumPoolSize (advanced)	順序付けされたスレッドプールのコアプールサイズ（使用されている場合）。デフォルト値は 16 です。	16	int
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Netty エンドポイントは、URI 構文を使用して設定します。

```
netty:protocol:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

227.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
protocol	tcp または udp が使用できるプロトコルが **必要です**。		文字列
host	**必須** のホスト名。コンシューマーの場合、ホスト名は localhost または 0.0.0.0 です。プロデューサーの場合、ホスト名はリモートホストで接続するリモートホストになります。		文字列
port	**必須**: ホストのポート番号		int

227.2.2. クエリーパラメーター（67 パラメーター）：

Name	説明	デフォルト	Type
切断 （共通）	Netty チャンネルの使用直後に切断する（クローズ）かどうか。コンシューマーとプロデューサーの両方に使用できます。	false	boolean
keepAlive (common)	非アクティブのためソケットが閉じられないようにする設定	true	boolean
reuseAddress (common)	ソケットの多重化を容易にするための設定	true	boolean
同期 （共通）	エンドポイントを一方向または request-response として設定する設定	true	boolean
tcpNoDelay (common)	TCP プロトコルのパフォーマンスを改善するための設定	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
ブロードキャスト （コンシューマー）	UDP でマルチキャストを選択する設定	false	boolean

Name	説明	デフォルト	Type
clientMode (consumer)	clientMode が true の場合、netty コンシューマーはアドレスを TCP クライアントとして接続します。	false	boolean
Back log(consumer)	netty コンシューマー（サーバー）のバックログを設定できます。バックログは、OS に応じてベストエフォートである点に注意してください。このオプションを 200、500、1000 などの値に設定し、TCP スタックに accept キューが適しているかどうかを指示し、このオプションが設定されていない場合、バックログは OS の設定により異なります。		int
bossCount (consumer)	Netty が nio モードで動作している場合、Netty からのデフォルトの bossCount パラメーターを使用します。これは 1 です。ユーザーはこの操作を使用して Netty からデフォルトの bossCount をオーバーライドできます。	1	int
bossPool (consumer)	明示的な org.jboss.netty.channel.socket.nio.BossPool を boss スレッドプールとして使用する。たとえば、複数のコンシューマーでスレッドプールを共有する場合などです。デフォルトでは、各コンシューマーには 1 つのコアスレッドを持つ独自の boss プールがあります。		BossPool
channelGroup (consumer)	明示的な ChannelGroup を使用します。		ChannelGroup
disconnectOnNoReply (consumer)	sync が有効になっている場合、送信先の応答がない場合に NettyConsumer を指定します。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
maxChannelMemorySize (consumer)	orderedThreadPoolExecutor を使用する場合、チャネルごとにキューに置かれたイベントの最大サイズ。無効にするには 0 を指定します。	1048576	Long
maxTotalMemorySize (consumer)	orderedThreadPoolExecutor を使用する場合に、このプールのキューに置かれたイベントの最大サイズ。無効にするには 0 を指定します。	209715200	Long

Name	説明	デフォルト	Type
nettyServerBootstrapFactory (consumer)	カスタム NettyServerBootstrapFactory を使用する場合		NettyServerBootstrap Factory
networkInterface (consumer)	UDP を使用する場合、このオプションを使用して、マルチキャストグループに参加する eth0 などの名前でネットワークインターフェースを指定できます。		文字列
noReplyLogLevel (consumer)	同期が有効な場合、ロギングに使用するログレベルを NettyConsumer に指示し、返信する応答はありません。	WARN	LoggingLevel
orderedThreadPoolExecutor (consumer)	順序付けされたスレッドプールを使用して、同じチャネルでイベントが順序付け処理されるかどうか。詳細は、org.jboss.netty.handler.execution.OrderedMemoryAwareThreadPoolExecutor の netty javadoc を参照してください。	true	boolean
serverClosedChannelExceptionCaughtLogLevel (consumer)	サーバー(NettyConsumer)が java.nio.channels.ClosedChannelException を取得する場合、このログレベルを使用してログに記録されます。これは、クライアントが突然切断され、Nettyサーバーで閉じられた例外が一杯になる可能性があるため、閉じられたチャンネル例外のロギングを避けるために使用されます。	DEBUG	LoggingLevel
serverExceptionCaughtLog Level (consumer)	サーバー(NettyConsumer)が例外をキャッチすると、このログレベルを使用してログに記録されます。	WARN	LoggingLevel
serverPipelineFactory (consumer)	カスタムの ServerPipelineFactory を使用するには、以下を行います。		ServerPipelineFactory
workerCount (consumer)	Netty が nio モードで機能する場合、Netty からのデフォルトの workerCount パラメーターを使用します。これは cpu_core_threads2 です。ユーザーはこの操作を使用して Netty からデフォルトの workerCount をオーバーライドできます。		int
workerPool (consumer)	明示的な org.jboss.netty.channel.socket.nio.WorkerPool をワーカースレッドプールとして使用する。たとえば、複数のコンシューマーでスレッドプールを共有する場合などです。デフォルトでは、各コンシューマーには 2 x cpu 数コアスレッドを持つ独自のワーカープールがあります。		WorkerPool

Name	説明	デフォルト	Type
connectTimeout (producer)	ソケット接続が利用可能になるまで待機する時間。値はミリ秒単位です。	10000	Long
requestTimeout (producer)	リモートサーバーを呼び出すときに Netty プロデューサーにタイムアウトを使用できます。デフォルトではタイムアウトは使用されていません。値はミリ秒単位であるため、たとえば 30000 は 30 秒です。requestTimeout は Netty の ReadTimeoutHandler を使用してタイムアウトを発生させます。		Long
clientPipelineFactory (producer)	カスタムの ClientPipelineFactory を使用するには、以下を実行します。		ClientPipelineFactory
lazyChannelCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合に、例外を回避するためにチャネルをレイジーに作成できます。	true	boolean
producerPoolEnabled (producer)	プロデューサープールが有効かどうか。重要： 同時実行で信頼できるリクエスト/応答の処理にプールが必要なため、これをオフにしないでください。	true	boolean
producerPoolMaxActive (producer)	特定の時間にプールによって割り当て可能なオブジェクト数の上限を設定します（クライアントにチェックするか、待機しているオブジェクト数）。制限なしに負の値を使用してください。	-1	int
producerPoolMaxIdle (producer)	プール内のアイドルインスタンスの数の上限を設定します。	100	int
producerPoolMinEvictable Idle (producer)	アイドルオブジェクトによりエビクションの対象となる前にオブジェクトがプール内にアイドル状態である可能性がある最小時間（ミリ秒単位）を設定します。	300000	Long
producerPoolMinIdle (producer)	エビクトスレッド（アクティブな場合）が新規オブジェクトを生成する前に、プロデューサープールで許可されるインスタンスの最小数を設定します。		int
udpConnectionlessSending (producer)	このオプションは、実際の火災報知能である接続が少ない udp 送信をサポートします。接続された udp は、受信側ポートをリッスンしていない場合は PortUnreachableException を受け取ります。	false	boolean
useChannelBuffer (producer)	useChannelBuffer が true の場合、netty プロデューサーはメッセージボディーを ChannelBuffer に変換してから送信します。	false	boolean

Name	説明	デフォルト	Type
bootstrapConfiguration (advanced)	このエンドポイントを設定するには、カスタムに設定された NettyServerBootstrapConfiguration を使用します。		NettyServerBootstrap Configuration
オプション （詳細）	オプション. を接頭辞として使用して追加の netty オプションを設定できます。たとえば、options.child.keepAlive=false を指定して netty オプションの child.keepAlive=false を設定します。使用できるオプションについては、Netty のドキュメントを参照してください。		マップ
receiveBufferSize (advanced)	受信接続中に使用される TCP/UDP バッファーサイズ。サイズはバイト単位です。	65536	Long
receiveBufferSizePredictor (advanced)	バッファーサイズの予測を設定します。詳細は、Jetty のドキュメント およびこのメールスレッド を参照してください。		int
sendBufferSize (advanced)	アウトバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイト単位です。	65536	Long
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
transferExchange (advanced)	TCP にのみ使用されます。ボディーだけでなく、ネットワーク上でエクスチェンジを転送できます。次のフィールドが転送されます。本文、Out ボディー、フォールトボディー、ヘッダー、送信ヘッダー、エクスチェンジプロパティー、エクスチェンジ例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。	false	boolean
allowDefaultCodec (codec)	netty コンポーネントは、エンコーダー/デックが null で、テキストラインが false の場合、デフォルトのコーデックをインストールします。allowDefaultCodec を false に設定すると、netty コンポーネントがフィルターチェーンの最初の要素としてデフォルトのコーデックをインストールできなくなります。	true	boolean
autoAppendDelimiter (codec)	テキストラインコーデックを使用して送信する際に、欠落している終了区切り文字を自動追加するかどうか。	true	boolean

Name	説明	デフォルト	Type
decoder (codec)	インバウンドペイロードの特別なマーシャリングを実行するために使用できるカスタム ChannelHandler クラス。**org.jboss.netty.channel.ChannelUpStreamHandler を上書きする必要があります。**		ChannelHandler
decoderMaxLineLength (codec)	テキストラインコーデックに使用する最大行長。	1024	int
decoders (codec)	使用されるデコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値をプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。		文字列
delimiter (codec)	テキストラインコーデックに使用する区切り文字。使用できる値は LINE と NULL です。	LINE	TextLineDelimiter
encoder (codec)	アウトバウンドペイロードの特別なマーシャリングを実行するために使用できるカスタム ChannelHandler クラス。**Must override org.jboss.netty.channel.ChannelDownStreamHandler.**		ChannelHandler
encoders (codec)	使用されるエンコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値をプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。		文字列
encoding (codec)	テキストラインコーデックに使用するエンコーディング（文字セット名）。指定されていない場合、Camel は JVM デフォルトの Charset を使用します。		文字列
テキストライン（コード）	TCP にのみ使用されます。codec が指定されていない場合、このフラグを使用してテキスト行ベースのコーデックを指定できます。指定されていない場合や、値が false の場合は、Object Serialization は TCP を介して想定されます。	false	boolean
enabledProtocols (security)	SSL を使用する際に有効にするプロトコル	TLSv1, TLSv1.1, TLSv1.2	文字列
keyStoreFile (security)	暗号化に使用されるクライアント側の証明書キーストア		ファイル

Name	説明	デフォルト	Type
keyStoreFormat (security)	ペイロードの暗号化に使用するキーストア形式。設定されていない場合、デフォルトで JKS に設定されます。	JKS	文字列
keyStoreResource (security)	暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		文字列
needClientAuth (security)	SSL の使用時にサーバーがクライアント認証を必要とするかどうかを設定します。	false	boolean
passphrase （セキュリティー）	SSH を使用して送信されたペイロードの暗号化/復号化に使用するパスワード設定		文字列
securityProvider (security)	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合、デフォルトは SunX509 に設定されます。	SunX509	文字列
ssl （セキュリティー）	SSL 暗号化がこのエンドポイントに適用されるかどうかを指定する設定	false	boolean
sslClientCertHeaders (security)	有効にすると、Netty コンシューマーはサブジェクト名、発行者名、シリアル番号、有効な日付範囲などのクライアント証明書に関する情報を持つヘッダーで Camel メッセージを強化します。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters
sslHandler (security)	SSL ハンドラーを返すために使用できるクラスへの参照		SslHandler
trustStoreFile (security)	暗号化に使用されるサーバー側の証明書キーストア		ファイル
trustStoreResource (security)	暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		文字列

227.3. レジストリーベースのオプション

Codec ハンドラーおよび SSL キーストアは、Spring XML ファイルなど、レジストリーに登録でき

ます。渡すことができる値は次のとおりです。

Name	説明
passphrase	SSH を使用して送信されたペイロードの暗号化/復号化に使用するパスワード設定
keyStoreFormat	ペイロードの暗号化に使用するキーストア形式。設定されていない場合は、デフォルトで「JKS」に設定されます。
securityProvider	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合は、デフォルトで "SunX509" に設定されます。
keyStoreFile	**非推奨：** 暗号化に使用されるクライアント側の証明書キーストア
trustStoreFile	**非推奨：** 暗号化に使用されるサーバー側の証明書キーストア
keyStoreResource	Camel 2.11.1: 暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、**「classpath:」、"file:"** 、または **"http:"** のプレフィックスを指定して、異なるシステムからリソースをロードすることもできます。
trustStoreResource	Camel 2.11.1: 暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、**「classpath:」、"file:"** 、または **"http:"** のプレフィックスを指定して、異なるシステムからリソースをロードすることもできます。
sslHandler	SSL ハンドラーを返すために使用できるクラスへの参照
encoder	アウトバウンドペイロードの特別なマーシャリングの実行に使用できるカスタム **ChannelHandler** クラス。Must override **org.jboss.netty.channel.ChannelDownStreamHandler**.
encorders	使用されるエンコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値を # のプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。
decoder	インバウンドペイロードの特別なマーシャリングの実行に使用できるカスタム **ChannelHandler** クラス。**org.jboss.netty.channel.ChannelUpStreamHandler** を上書きする必要があります。

Name	説明
decoders	使用されるデコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値を # のプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。

重要： 共有不可能なエンコーダー/デコーダーの使用については、以下を参照してください。

227.3.1. 共有不可能なエンコーダーまたはデコーダーの使用

エンコーダーまたはデコーダーが共有できない場合（@Shareable クラスアノテーションなど）、エンコーダー/デコーダーは org.apache.camel.component.netty.ChannelHandlerFactory インターフェースを実装し、newChannelHandler メソッドで新規インスタンスを返す必要があります。これは、エンコーダー/デコーダーを安全に使用できるようにするためです。そうでない場合、Netty コンポーネントはエンドポイントの作成時に WARN をログに記録します。

Netty コンポーネントは、一般的に使用されるメソッドが多数含まれる org.apache.camel.component.netty.ChannelHandlerFactories ファクトリークラスを提供します。

227.4. NETTY エンドポイントとの/からのメッセージの送信

227.4.1. Netty プロデューサー

*Producer モードでは、コンポーネントは TCP プロトコルまたは UDP プロトコルのいずれかを使用してペイロードをソケットエンドポイント
に送信する機能を提供します（任意の SSL サポートを使用）。*

プロデューサーモードは、一方向およびリクエスト応答ベースの操作の両方をサポートします。

227.4.2. Netty コンシューマー

Consumer モードでは、コンポーネントは以下を行う機能を提供します。

- *TCP プロトコルまたは UDP プロトコル（任意の SSL サポートあり）を使用して、指定のソケットをリッスンします。*

- *text/xml、バイナリー、およびシリアライズされたオブジェクトベースのペイロードを使用してソケットでリクエストを受信する。*

- *メッセージエクスチェンジとしてルート経由で送信します。*

コンシューマーモードは、一方向およびリクエスト応答ベースの操作の両方をサポートします。

227.5. HEADERS

以下のヘッダーは、Netty コンシューマーによって作成されたエクスチェンジに対して入力されます。

ヘッダーキー	クラス	説明
NettyConstants.NETTY_CHANNEL_HANDLER_CONTEXT / CamelNettyChannelHandlerContext	**org.jboss.netty.channel.ChannelHandlerContext**	'ChannelHandlerContext 'instance associated with the connection received by netty.
NettyConstants.NETTY_MESSAGE_EVENT / CamelNettyMessageEvent	**org.jboss.netty.channel.MessageEvent**	'MessageEvent 'instance associated with the connection received by netty.

ヘッダーキー	クラス	説明
NettyConstants.NETTY_REMOTE_ADDRESS / CamelNettyRemoteAddress	java.net.SocketAddress	受信ソケット接続のリモートアドレス。
NettyConstants.NETTY_LOCAL_ADDRESS / CamelNettyLocalAddress	java.net.SocketAddress	受信ソケット接続のローカルアドレス。

227.6. 使用状況サンプル

227.6.1. リクエスト応答およびシリアライズされたオブジェクトペイロードを使用した UDP Netty エンドポイント

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
    from("netty:udp://localhost:5155?sync=true")
      .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          Poetry poetry = (Poetry) exchange.getIn().getBody();
          poetry.setPoet("Dr. Sarojini Naidu");
          exchange.getOut().setBody(poetry);
        }
      }
    }
};
```

227.6.2. 一方向通信を使用した TCP ベースの Netty コンシューマーエンドポイント

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
      from("netty:tcp://localhost:5150")
          .to("mock:result");
  }
};
```

227.6.3. リクエスト応答通信を使用した SSL/TCP ベースの Netty コンシューマーエンドポイント

JSSE 設定ユーティリティーの使用

Camel 2.9 の時点で、Netty コンポーネントは Camel JSSE 設定ユーティリティーを介して SSL/TLS 設定をサポートします。 このユーティリティーは、エンドポイントおよびコンポーネントレベルで記述し、設定する必要のあるコンポーネント固有のコードの量を大幅に削減します。 以下の例は、Netty コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

NettyComponent nettyComponent = getContext().getComponent("netty",
NettyComponent.class);
nettyComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
      id="sslContextParameters">
    <camel:keyManagers
        keyPassword="keyPassword">
      <camel:keyStore
          resource="/users/home/server/keystore.jks"
          password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
```

```
  <to uri="netty:tcp://localhost:5150?
sync=true&ssl=true&sslContextParameters=#sslContextParameters"/>
...
```

Jetty コンポーネントでの基本的な SSL/TLS 設定の使用

```
JndiRegistry registry = new JndiRegistry(createJndiContext());
registry.bind("password", "changeit");
registry.bind("ksf", new File("src/test/resources/keystore.jks"));
registry.bind("tsf", new File("src/test/resources/keystore.jks"));

context.createRegistry(registry);
context.addRoutes(new RouteBuilder() {
  public void configure() {
     String netty_ssl_endpoint =
       "netty:tcp://localhost:5150?sync=true&ssl=true&passphrase=#password"
       + "&keyStoreFile=#ksf&trustStoreFile=#tsf";
     String return_string =
       "When You Go Home, Tell Them Of Us And Say,"
       + "For Your Tomorrow, We Gave Our Today.";

     from(netty_ssl_endpoint)
      .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          exchange.getOut().setBody(return_string);
        }
      }
  }
});
```

SSLSession およびクライアント証明書へのアクセス

Camel 2.12 から利用可能

クライアント証明書に関する詳細を取得する必要がある場合は、javax.net.ssl.SSLSession にアクセスできます。ssl=true の場合、Netty コンポーネントは以下のように SSLSession を Camel メッセージにヘッダーとして保存します。

```
SSLSession session = exchange.getIn().getHeader(NettyConstants.NETTY_SSL_SESSION,
SSLSession.class);
// get the first certificate which is client certificate
javax.security.cert.X509Certificate cert = session.getPeerCertificateChain()[0];
Principal principal = cert.getSubjectDN();
```

クライアントを認証するために needClientAuth=true を設定するのを忘れないようにしてください。それ以外の場合、SSLSession はクライアント証明書に関する情報にアクセスできないため、例外

javax.net.ssl.SSLPeerUnverifiedException: peer not authenticated が発生することがあります。また、クライアント証明書の有効期限が切れるか、有効でない場合はこの例外を取得することもできます。

ヒント

sslClientCertHeaders オプションを true に設定し、クライアント証明書に関する詳細が含まれるヘッダーを使用して Camel メッセージを強化できます。たとえば、サブジェクト名は CamelNettySSLClientCertSubjectName ヘッダーで読み取り可能です。

227.6.4. 複数のコーデックの使用

場合によっては、エンコーダーとデコーダーのチェーンを netty パイプラインに追加する必要がある場合があります。multipile codecs を camel netty エンドポイントに追加するには、'encoders' および 'decoders' uri パラメーターを使用する必要があります。'encoder' および 'decoder' パラメーターと同様に、パイプラインに追加する必要のある参照（ChannelUpstreamHandlers および ChannelDownstreamHandler のリスト）を指定するために使用されます。エンコーダーが指定されている場合は、デコーダーやデコーダーパラメーターと同様にエンコーダーパラメーターが無視されることに注意してください。

INFO: 非共有可能なエンコーダー/デコーダーの使用について追加でお読みください。

エンドポイントの作成時に解決できるようにコーデックのリストを Camel のレジストリーに追加する必要があります。

Spring のネイティブコレクションサポートを使用して、アプリケーションコンテキストでコーデックリストを指定することができます。

その後、bean 名は netty エンドポイント定義で、コンマ区切りのリストとして、または List（例 : List）を含めることができます。

または spring 経由。

227.7. 完了後にチャンネルを閉じる

サーバーとして動作する場合は、クライアント変換の完了時にチャネルを閉じる必要がある場合があります。
これは、endpoint オプション disconnect=true を設定するだけで実行できます。

ただし、以下のようにメッセージごとに Camel に指示することもできます。
Camel にチャネルを閉じるよう指示するには、CamelNettyCloseChannelWhenComplete キーでヘッダーをブール値 true に追加する必要があります。
たとえば、以下の例では bye メッセージをクライアントに書き直した後にチャンネルを閉じます。

```
    from("netty:tcp://localhost:8080").process(new Processor() {
        public void process(Exchange exchange) throws Exception {
            String body = exchange.getIn().getBody(String.class);
            exchange.getOut().setBody("Bye " + body);
            // some condition which determines if we should close
            if (close) {

exchange.getOut().setHeader(NettyConstants.NETTY_CLOSE_CHANNEL_WHEN_COMPLETE,
true);
            }
        }
    });
```

227.8. 作成されたパイプラインを完全に制御するためにカスタムチャネルパイプラインファクトリーを追加

Camel 2.5 で利用可能

カスタムチャンネルパイプラインは、カスタムハンドラー、エンコーダー(s)、デコーダーを挿入することで、ハンドラー/インターセプターチェーン上で完全な制御を提供します。

カスタムパイプラインを追加するには、コンテキストレジストリー（JNDIRegistry、または camel-spring ApplicationContextRegistry など）を使用してコンテキストで、カスタムチャネルパイプラインファクトリーを作成して登録する必要があります。

カスタムパイプラインファクトリーは、以下のように構築する必要があります。

- プロデューサーにリンクされたチャネルパイプラインファクトリーは、抽象クラス ClientPipelineFactory を拡張する必要があります。
- コンシューマーにリンクされたチャネルパイプラインファクトリーは、抽象クラス ServerPipelineFactory を拡張する必要があります。
- クラスは、カスタムハンドラー、エンコーダー、およびデコーダーを挿入するために、getPipeline（）メソッドをオーバーライドする必要があります。getPipeline（）メソッドを

上書きすると、パイプラインに接続しているハンドラー、エンコーダー、またはデコーダーのないパイプラインが作成されます。

以下の例は、ServerChannel Pipeline ファクトリーを作成する方法を示しています。

カスタムパイプラインファクトリーの使用

```
public class SampleServerChannelPipelineFactory extends ServerPipelineFactory {
    private int maxLineSize = 1024;

    public ChannelPipeline getPipeline() throws Exception {
        ChannelPipeline channelPipeline = Channels.pipeline();

        channelPipeline.addLast("encoder-SD", new StringEncoder(CharsetUtil.UTF_8));
        channelPipeline.addLast("decoder-DELIM", new
DelimiterBasedFrameDecoder(maxLineSize, true, Delimiters.lineDelimiter()));
        channelPipeline.addLast("decoder-SD", new StringDecoder(CharsetUtil.UTF_8));
        // here we add the default Camel ServerChannelHandler for the consumer, to allow Camel
to route the message etc.
        channelPipeline.addLast("handler", new ServerChannelHandler(consumer));

        return channelPipeline;
    }
}
```

その後、カスタムチャネルパイプラインファクトリーをレジストリーに追加し、以下の方法で camel ルートでインスタンス化/使用率を使用できるようにします。

```
Registry registry = camelContext.getRegistry();
serverPipelineFactory = new TestServerChannelPipelineFactory();
registry.bind("spf", serverPipelineFactory);
context.addRoutes(new RouteBuilder() {
  public void configure() {
     String netty_ssl_endpoint =
       "netty:tcp://localhost:5150?serverPipelineFactory=#spf"
     String return_string =
       "When You Go Home, Tell Them Of Us And Say,"
       + "For Your Tomorrow, We Gave Our Today.";

     from(netty_ssl_endpoint)
      .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          exchange.getOut().setBody(return_string);
        }
      }
  }
});
```

227.9. NETTY ボットおよびワーカースレッドプールの再利用

Camel 2.12 から利用可能

Netty には、boss と worker の 2 種類のスレッドプールがあります。デフォルトでは、各 Netty コンシューマーおよびプロデューサーにはプライベートスレッドプールがあります。複数のコンシューマーまたはプロデューサー間でこれらのスレッドプールを再利用する場合は、スレッドプールを作成し、レジストリーに登録する必要があります。

たとえば Spring XML を使用する場合、以下のように 2 つのワーカースレッドを持つ NettyWorkerPoolBuilder を使用して共有ワーカースレッドプールを作成できます。

```
<!-- use the worker pool builder to help create the shared thread pool -->
<bean id="poolBuilder" class="org.apache.camel.component.netty.NettyWorkerPoolBuilder">
  <property name="workerCount" value="2"/>
</bean>

<!-- the shared worker thread pool -->
<bean id="sharedPool" class="org.jboss.netty.channel.socket.nio.WorkerPool"
      factory-bean="poolBuilder" factory-method="build" destroy-method="shutdown">
</bean>
```

ヒント

boss スレッドプールには、Netty コンシューマー向けの org.apache.camel.component.netty.NettyServerBossPoolBuilder ビルダーと、Netty プロデューサー用の org.apache.camel.component.netty.NettyClientBossPoolBuilder があります。

その後、以下のように URI に workerPool オプションを設定して、このワーカープールを参照できます。

```
<route>
  <from uri="netty:tcp://localhost:5021?
textline=true&amp;sync=true&amp;workerPool=#sharedPool&amp;orderedThreadPoolExecutor=false"
/>
  <to uri="log:result"/>
  ...
</route>
```

別のルートがある場合は、共有ワーカープールを参照できます。

```
<route>
```

```
    <from uri="netty:tcp://localhost:5022?
textline=true&amp;sync=true&amp;workerPool=#sharedPool&amp;orderedThreadPoolExecutor=false"
/>
    <to uri="log:result"/>
    ...
  </route>
```

i. などになります。

227.10. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- Netty HTTP
- MINA

第228章 NETTY HTTP コンポーネント（非推奨）

Camel バージョン 2.12 から利用可能

netty-http コンポーネントは Netty で HTTP トランスポートを調整するための Netty コンポーネントのエクステンションです。

この Camel コンポーネントは、プロデューサーとコンシューマーエンドポイントの両方をサポートします。

警告

このコンポーネントは非推奨になりました。Netty4 HTTP を使用する必要があります。

INFO: Stream.Netty はストリームベースであるため、受信する入力はストリームとして Camel に送信されます。つまり、は 1 回 のみストリームのコンテンツを読み取ることができます。メッセージボディーが空であるか、データに複数回アクセスする必要がある場合（マルチキャストの実行や再配信エラー処理など）、ストリームキャッシュを使用するか、メッセージボディーを複数回再読み取りできる String に変換する必要があります。Netty4 HTTP は、io.netty.handler.codec.http.HttpObjectAggregator を使用してストリーム全体をメモリーに読み取り、完全な http メッセージ全体を構築することに注意してください。ただし、生成されるメッセージは、1 度に判読できるストリームベースのメッセージになります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-netty-http</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

228.1. URI 形式

netty コンポーネントの URI スキームは以下のようになります。

```
netty-http:http://localhost:8080[?options]
```

URI にクエリーオプションを追加するには、 ?option=value&option=value&...

INFO: Query parameters vs endpoint options.Camel が URI クエリーパラメーターおよびエンドポイントオプションを認識する方法について理解している場合があります。たとえば、エンドポイント URI を - netty-http:http//example.com?myParam=myValue&compression=true のように作成できます。この例では、myParam は HTTP パラメーターですが、圧縮 は Camel エンドポイントオプションです。このような状況で Camel によって使用されるストラテジーは、利用可能なエンドポイントオプションを解決し、URI から削除することです。この例では、Netty HTTP プロデューサーによってエンドポイントに送信される HTTP リクエストは、 http//example.com?myParam=myValue のように表示されます。圧縮 エンドポイントオプションはターゲット URL から解決され、削除されるためです。また、動的ヘッダー（ CamelHttpQueryなど）を使用してエンドポイントオプションを指定できない点に留意してください。エンドポイントオプションは、エンドポイント URI 定義レベルでのみ指定できます（DSL 要素 に対する ような）。

228.2. HTTP オプション

INFO: 多くのオプション。重要： このコンポーネントは Netty からすべてのオプションを継承します。そのため、Netty のドキュメント も確認してください。
UDP トランスポートに関連するオプションなど、 Netty HTTP コンポーネントを使用する場合、Netty からの一部のオプションは適用されないことに注意してください。

Netty HTTP コンポーネントは、以下に示す 7 つのオプションをサポートします。

Name	説明	デフォルト	Type
nettyHttpBinding (advanced)	Netty および Camel Message API への/からのバインディングにカスタムの org.apache.camel.component.netty.http.NettyHttpBinding を使用するには、以下を行います。		NettyHttpBinding
設定 （共通）	エンドポイントの作成時に NettyConfiguration を設定として使用します。		NettyHttpConfiguration
headerFilterStrategy (advanced)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用してヘッダーをフィルターします。		HeaderFilterStrategy

Name	説明	デフォルト	Type
securityConfiguration （セキュリティー）	セキュアな Web リソースを設定するための org.apache.camel.component.netty.http.NettyHttpSecurityConfiguration を参照します。		NettyHttpSecurityConfiguration
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
maximumPoolSize (advanced)	順序付けされたスレッドプールのコアプールサイズ（使用されている場合）。デフォルト値は 16 です。	16	int
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Netty HTTP エンドポイントは、URI 構文を使用して設定します。

```
netty-http:protocol:host:port/path
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

228.2.1. パスパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
protocol	**必須**: http または https のいずれかを使用するプロトコル。		文字列
host	localhost などのローカルホスト名、またはコンシューマーの場合は 0.0.0.0 が **必要** になります。プロデューサーの使用時にリモート HTTP サーバーのホスト名。		文字列
port	ホストのポート番号		int
path	リソースパス		文字列

228.2.2. クエリーパラメーター（78 パラメーター）：

Name	説明	デフォルト	Type
bridgeEndpoint (common)	オプションが true の場合、プロデューサーは Exchange.HTTP_URI ヘッダーを無視し、リクエストにエンドポイントの URI を使用します。また、throwExceptionOnFailure を false に設定して、プロデューサーが障害応答をすべて返信するようにすることもできます。ブリッジモードで動作しているコンシューマーは、gzip 圧縮および WWW URL フォームエンコーディングを省略します（Exchange.SKIP_GZIP_ENCODING および Exchange.SKIP_WWW_FORM_URLENCODED ヘッダーを消費したエクスチェンジに追加します）。	false	boolean
切断 （共通）	Netty チャンネルの使用直後に切断する（クローズ）かどうか。コンシューマーとプロデューサーの両方に使用できます。	false	boolean
keepAlive (common)	非アクティブのためソケットが閉じられないようにする設定	true	boolean
reuseAddress (common)	ソケットの多重化を容易にするための設定	true	boolean
同期 （共通）	エンドポイントを一方向または request-response として設定する設定	true	boolean
tcpNoDelay (common)	TCP プロトコルのパフォーマンスを改善するための設定	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
matchOnUriPrefix (consumer)	完全一致がない場合、Camel が URI プレフィックスと一致するターゲットコンシューマーの検索を試行するかどうか。	false	boolean
send503whenSuspended (consumer)	コンシューマーが停止した場合に HTTP ステータスコード 503 を送信するかどうか。オプションが false の場合、コンシューマーが中断された場合に Netty Acceptor がバインド解除されるため、クライアントは接続できなくなります。	true	boolean

Name	説明	デフォルト	Type
Back log(consumer)	netty コンシューマー（サーバー）のバックログを設定できます。バックログは、OS に応じてベストエフォートである点に注意してください。このオプションを 200、500、1000 などの値に設定し、TCP スタックに accept キューが適しているかどうかを指示し、このオプションが設定されていない場合、バックログは OS の設定により異なります。		int
bossCount (consumer)	Netty が nio モードで動作している場合、Netty からのデフォルトの bossCount パラメーターを使用します。これは 1 です。ユーザーはこの操作を使用して Netty からデフォルトの bossCount をオーバーライドできます。	1	int
bossPool (consumer)	明示的な org.jboss.netty.channel.socket.nio.BossPool を boss スレッドプールとして使用する。たとえば、複数のコンシューマーでスレッドプールを共有する場合などです。デフォルトでは、各コンシューマーには 1 つのコアスレッドを持つ独自の boss プールがあります。		BossPool
channelGroup (consumer)	明示的な ChannelGroup を使用します。		ChannelGroup
chunkedMaxContentLength (consumer)	Netty HTTP サーバーで受信されるチャンクされたフレームごとの最大コンテンツの長さ（バイト単位）。	1048576	int
圧縮（コンシューマー）	クライアントが HTTP ヘッダーからサポートしている場合は、Netty HTTP サーバーで圧縮に gzip/deflate の使用を許可します。	false	boolean

Name	説明	デフォルト	Type
disableStreamCache (consumer)	Netty HttpRequestgetContent（）からの raw 入力ストリームがキャッシュされているかどうかを決定します（Camel はストリームを軽量メモリーベースのストリームキャッシュに読み込みます）。デフォルトでは、Camel は Netty 入力ストリームをキャッシュして、複数回ロードし、Camel がストリームからすべてのデータを取得できるようにします。ただし、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。たとえば、ファイルまたは他の永続ストアに直接ストリーミングする場合などに、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。このオプションを有効にすると、Netty ストリームが追加設定なしで複数回読み取ることができないため、Netty raw ストリームでリーダーインデックスを手動でリセットする必要がある点に留意してください。	false	boolean
disconnectOnNoReply (consumer)	sync が有効になっている場合、送信先の応答がない場合に NettyConsumer を指定します。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
httpMethodRestrict (consumer)	Netty HTTP コンシューマーで HTTP メソッドを無効にします。カンマで区切って複数指定できます。		文字列
mapHeaders (consumer)	このオプションを有効にすると、Netty から Camel メッセージへのバインディング時にヘッダーもマッピングされます（ヘッダーとして Camel メッセージに追加されます）。このオプションを無効にして無効にすることができます。ヘッダーには、Netty HTTP リクエスト org.jboss.netty.handler.codec.http.HttpRequest インスタンスを返す getHttpRequest（）メソッドを使用して、org.apache.camel.component.netty.http.NettyHttpMessage メッセージからアクセスできます。	true	boolean

Name	説明	デフォルト	Type
maxChannelMemorySize (consumer)	orderedThreadPoolExecutor を使用する場合、チャネルごとにキューに置かれたイベントの最大サイズ。無効にするには 0 を指定します。	10485760	Long
maxHeaderSize (consumer)	すべてのヘッダーの最大長。各ヘッダーの長さの合計がこの値を超えると、TooLongFrameException が発生します。	8192	int
maxTotalMemorySize (consumer)	orderedThreadPoolExecutor を使用する場合に、このプールのキューに置かれたイベントの最大サイズ。無効にするには 0 を指定します。	209715200	Long
nettyServerBootstrapFactory (consumer)	カスタム NettyServerBootstrapFactory を使用する場合		NettyServerBootstrap Factory
nettySharedHttpServer (consumer)	共有された Netty HTTP サーバーを使用します。詳細は、「Netty HTTP Server の例」を参照してください。		NettySharedHttpServer
noReplyLogLevel (consumer)	同期が有効な場合、ロギングに使用するログレベルを NettyConsumer に指示し、返信する応答はありません。	WARN	LoggingLevel
orderedThreadPoolExecutor (consumer)	順序付けされたスレッドプールを使用して、同じチャネルでイベントが順序付け処理されるかどうか。詳細は、org.jboss.netty.handler.execution.OrderedMemoryAwareThreadPoolExecutor の netty javadoc を参照してください。	true	boolean
serverClosedChannelExceptionCaughtLogLevel (consumer)	サーバー(NettyConsumer)が java.nio.channels.ClosedChannelException を取得する場合、このログレベルを使用してログに記録されます。これは、クライアントが突然切断され、Nettyサーバーで閉じられた例外が一杯になる可能性があるため、閉じられたチャンネル例外のロギングを避けるために使用されます。	DEBUG	LoggingLevel
serverExceptionCaughtLog Level (consumer)	サーバー(NettyConsumer)が例外をキャッチすると、このログレベルを使用してログに記録されます。	WARN	LoggingLevel
serverPipelineFactory (consumer)	カスタムの ServerPipelineFactory を使用するには、以下を行います。		ServerPipelineFactory

Name	説明	デフォルト	Type
traceEnabled (consumer)	この Netty HTTP コンシューマーの HTTP TRACE を有効にするかどうかを指定します。デフォルトでは、TRACE はオフになっています。	false	boolean
urlDecodeHeaders (consumer)	このオプションが有効な場合には、Netty から Camel メッセージへのバインディング時にヘッダー値がデコードされます（例： %20 はスペース文字です）。このオプションは、デフォルトの org.apache.camel.component.netty.http.NettyHttpBinding によって使用されるため、カスタムの org.apache.camel.component.netty.http.NettyHttpBinding を実装する場合は、このオプションに応じてヘッダーをデコードする必要があります。	false	boolean
workerCount (consumer)	Netty が nio モードで機能する場合、Netty からのデフォルトの workerCount パラメーターを使用します。これは cpu_core_threads2 です。ユーザーはこの操作を使用して Netty からデフォルトの workerCount をオーバーライドできます。		int
workerPool (consumer)	明示的な org.jboss.netty.channel.socket.nio.WorkerPool をワーカースレッドプールとして使用する。たとえば、複数のコンシューマーでスレッドプールを共有する場合などです。デフォルトでは、各コンシューマーには 2 x cpu 数コアスレッドを持つ独自のワーカープールがあります。		WorkerPool
connectTimeout (producer)	ソケット接続が利用可能になるまで待機する時間。値はミリ秒単位です。	10000	Long
requestTimeout (producer)	リモートサーバーを呼び出すときに Netty プロデューサーにタイムアウトを使用できます。デフォルトではタイムアウトは使用されていません。値はミリ秒単位であるため、たとえば 30000 は 30 秒です。requestTimeout は Netty の ReadTimeoutHandler を使用してタイムアウトを発生させます。		Long
throwExceptionOnFailure (producer)	リモートサーバーからの応答に失敗した場合に HttpOperationFailedException のスローを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
clientPipelineFactory (producer)	カスタムの ClientPipelineFactory を使用するには、以下を実行します。		ClientPipelineFactory

Name	説明	デフォルト	Type
lazyChannelCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合に、例外を回避するためにチャネルをレイジーに作成できます。	true	boolean
okStatusCodeRange (producer)	正常な応答とみなされるステータスコード。値は含まれます。コンマで区切られた複数の範囲を定義できます（例： 200-204,209,301-304）。各範囲は、1 つの数字またはダッシュを含む from から でなければなりません。デフォルトの範囲は 200-299 です。	200-299	文字列
producerPoolEnabled (producer)	プロデューサープールが有効かどうか。重要： 同時実行で信頼できるリクエスト/応答の処理にプールが必要なため、これをオフにしないでください。	true	boolean
producerPoolMaxActive (producer)	特定の時間にプールによって割り当て可能なオブジェクト数の上限を設定します（クライアントにチェックするか、待機しているオブジェクト数）。制限なしに負の値を使用してください。	-1	int
producerPoolMaxIdle (producer)	プール内のアイドルインスタンスの数の上限を設定します。	100	int
producerPoolMinEvictable Idle (producer)	アイドルオブジェクトによりエビクションの対象となる前にオブジェクトがプール内にアイドル状態である可能性がある最小時間（ミリ秒単位）を設定します。	300000	Long
producerPoolMinIdle (producer)	エビクトスレッド（アクティブな場合）が新規オブジェクトを生成する前に、プロデューサープールで許可されるインスタンスの最小数を設定します。		int
useChannelBuffer (producer)	useChannelBuffer が true の場合、netty プロデューサーはメッセージボディーを ChannelBuffer に変換してから送信します。	false	boolean
useRelativePath (producer)	HTTP リクエストで相対パスを使用するかどうかを設定します。IBM Datapower などの一部のサードパーティーバックエンドシステムは、HTTP POST の絶対 URI をサポートしないため、このオプションを true に設定するとこの問題を回避できます。	false	boolean
bootstrapConfiguration (advanced)	このエンドポイントを設定するには、カスタムに設定された NettyServerBootstrapConfiguration を使用します。		NettyServerBootstrap Configuration

Name	説明	デフォルト	Type
Configuration (advanced)	このエンドポイントを設定するには、カスタムに設定された NettyHttpConfiguration を使用します。		NettyHttpConfiguration
headerFilterStrategy (advanced)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用してヘッダーをフィルターします。		HeaderFilterStrategy
nettyHttpBinding (advanced)	Netty および Camel Message API への/からのバインディングにカスタムの org.apache.camel.component.netty.http.NettyHttpBinding を使用するには、以下を行います。		NettyHttpBinding
オプション （詳細）	オプション. を接頭辞として使用して追加の netty オプションを設定できます。たとえば、options.child.keepAlive=false を指定して netty オプションの child.keepAlive=false を設定します。使用できるオプションについては、Netty のドキュメントを参照してください。		マップ
receiveBufferSize (advanced)	受信接続中に使用される TCP/UDP バッファーサイズ。サイズはバイト単位です。	65536	Long
receiveBufferSizePredictor (advanced)	バッファーサイズの予測を設定します。詳細は、Jetty のドキュメント およびこのメールスレッド を参照してください。		int
sendBufferSize (advanced)	アウトバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイト単位です。	65536	Long
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
transferException (advanced)	有効で Exchange がコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object のコンテンツタイプとして応答でシリアライズされたかどうか。プロデューサー側では、例外は HttpOperationFailedException ではなくデシリアライズされ、そのままスローされます。原因となる例外はシリアライズされている必要があります。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean

Name	説明	デフォルト	Type
transferExchange (advanced)	TCP にのみ使用されます。ボディーだけでなく、ネットワーク上でエクスチェンジを転送できます。次のフィールドが転送されます。本文、Out ボディー、フォールトボディー、ヘッダー、送信ヘッダー、エクスチェンジプロパティー、エクスチェンジ例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。	false	boolean
decoder (codec)	単一のデコーダーを使用するには **非推奨** になりました。このオプションは非推奨となっています。代わりにエンコーダーを使用してください。		ChannelHandler
decoders (codec)	使用されるデコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値をプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。		文字列
encoder (codec)	単一の **エンコーダーを使用するには非推奨** になりました。このオプションは非推奨となっています。代わりにエンコーダーを使用してください。		ChannelHandler
encoders (codec)	使用されるエンコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値をプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。		文字列
enabledProtocols (security)	SSL を使用する際に有効にするプロトコル	TLSv1, TLSv1.1, TLSv1.2	文字列
keyStoreFile (security)	暗号化に使用されるクライアント側の証明書キーストア		ファイル
keyStoreFormat (security)	ペイロードの暗号化に使用するキーストア形式。設定されていない場合、デフォルトで JKS に設定されます。	JKS	文字列
keyStoreResource (security)	暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		文字列

Name	説明	デフォルト	Type
needClientAuth (security)	SSL の使用時にサーバーがクライアント認証を必要とするかどうかを設定します。	false	boolean
passphrase （セキュリティー）	SSH を使用して送信されたペイロードの暗号化/復号化に使用するパスワード設定		文字列
securityConfiguration （セキュリティー）	セキュアな Web リソースを設定するための org.apache.camel.component.netty.http.NettyHttpSecurityConfiguration を参照します。		NettyHttpSecurityConfiguration
securityOptions (security)	マップからのキー/値のペアを使用した NettyHttpSecurityConfiguration の設定		マップ
securityProvider (security)	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合、デフォルトは SunX509 に設定されます。	SunX509	文字列
ssl （セキュリティー）	SSL 暗号化がこのエンドポイントに適用されるかどうかを指定する設定	false	boolean
sslClientCertHeaders (security)	有効にすると、Netty コンシューマーはサブジェクト名、発行者名、シリアル番号、有効な日付範囲などのクライアント証明書に関する情報を持つヘッダーで Camel メッセージを強化します。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters
sslHandler (security)	SSL ハンドラーを返すために使用できるクラスへの参照		SslHandler
trustStoreFile (security)	暗号化に使用されるサーバー側の証明書キーストア		ファイル
trustStoreResource (security)	暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		文字列

228.3. メッセージヘッダー

以下のヘッダーをプロデューサーで使用することで、HTTP リクエストを制御できます。

Name	タイプ	説明
CamelHttpMethod	**文字列**	GET、POST、TRACE などの HTTP メソッドの使用を制御できます。タイプは **org.jboss.netty.handler.codec.http.HttpMethod** インスタンスにもできます。
CamelHttpQuery	**文字列**	エンドポイント設定を上書きする URI クエリーパラメーターを **String** 値として指定できます。& 記号を使用して、複数のパラメーターを分離します。例：**foo=bar&beer=yes**
CamelHttpPath	**文字列**	Camel 2.13.1/2.12.4: エンドポイント設定を上書きする **String** 値として URI context-path および query パラメーターを提供できます。これにより、同じリモート http サーバーを呼び出すために同じプロデューサーを再利用できますが、動的なコンテキストパスとクエリーパラメーターを使用することができます。
Content-Type	**文字列**	HTTP ボディーの content-type を設定します。例： **text/plain; charset="UTF-8"**
CamelHttpResponseCode	**int**	HTTP ステータスコードの使用を許可します。デフォルトでは、成功には 200 を使用し、失敗した場合は 500 を使用します。

ルートが Netty HTTP エンドポイントから起動すると、以下のヘッダーが meta-data として提供されます。

表の説明は、 from("netty-http:http:0.0.0.0:8080/myapp")...

Name	タイプ	説明
CamelHttpMethod	**文字列**	GET、POST、TRACE などの使用される HTTP メソッド。
CamelHttpUrl	**文字列**	プロトコル、ホスト、ポートなどを含む URL
CamelHttpUri	**文字列**	プロトコル、ホストおよびポート等のない URI

Name	タイプ	説明
CamelHttpQuery	**文字列**	**foo=bar&beer=yes**などのクエリーパラメーター
CamelHttpRawQuery	**文字列**	Camel 2.13.0: **foo=bar&beer=yes** などのクエリーパラメーター。コンシューマーに到達すると（URL のデコード前）、raw フォームに保存されます。
CamelHttpPath	**文字列**	追加のコンテキストパス。この値は、context-path /**myapp** というクライアントの場合は空になります。クライアントが /**myapp**/**mystuff** を呼び出す場合、このヘッダーの値は /**mystuff** になります。つまり、ルートエンドポイントに設定された context-path の後の値です。
CamelHttpCharacterEncoding	**文字列**	content-type ヘッダーの文字セット。
CamelHttpAuthentication	**文字列**	ユーザーが HTTP **Basic** を使用して認証されている場合は、Basic の値でこのヘッダーが追加されます。
Content-Type	**文字列**	コンテンツタイプ（指定されている場合）。例： **text/plain; charset="UTF-8"**

228.4. NETTY タイプへのアクセス

このコンポーネントは、Exchange でのメッセージ実装として org.apache.camel.component.netty.http.NettyHttpMessage を使用します。これにより、以下のように、エンドユーザーは、必要に応じて元の Netty リクエスト/レスポンスインスタンスにアクセスできるようになります。元の応答はいつでもアクセスできない可能性があることに注意してください。

```
org.jboss.netty.handler.codec.http.HttpRequest request =
exchange.getIn(NettyHttpMessage.class).getHttpRequest();
```

228.5. 例

以下のルートでは、ハードコーディングされた「Bye World」メッセージを返す Netty HTTP を HTTP サーバーとして使用します。

-

```
    from("netty-http:http://0.0.0.0:8080/foo")
      .transform().constant("Bye World");
```

そして、Camel を使用してこの HTTP サーバーを以下のように ProducerTemplate を使用して呼び出すことができます。

```
    String out = template.requestBody("netty-http:http://localhost:8080/foo", "Hello World",
String.class);
    System.out.println(out);
```

そして、「Bye World」を出力として戻ります。

228.6. NETTY のワイルドカードを一致させる方法

デフォルトでは、Netty HTTP は URI に完全に一致します。ただし、Netty にプレフィックスに一致するよう指示することができます。たとえば、以下のようになります。

```
from("netty-http:http://0.0.0.0:8123/foo").to("mock:foo");
```

上記のルートでは、Netty HTTP は URI が完全に一致している場合にのみ一致するため、http://0.0.0.0:8123/foo と入力しても http://0.0.0.0:8123/foo/bar を行う場合に一致します。

そのため、ワイルドカードの一致を有効にする場合は、以下を実行します。

```
from("netty-http:http://0.0.0.0:8123/foo?matchOnUriPrefix=true").to("mock:foo");
```

そのため、Netty は foo で始まるエンドポイントに一致します。

任意のエンドポイントに一致させる には、以下を行います。

```
from("netty-http:http://0.0.0.0:8123?matchOnUriPrefix=true").to("mock:foo");
```

228.7. 同じポートで複数のルートを使用する

同じ CamelContext では、同じポート（ org.jboss.netty.bootstrap.ServerBootstrap インスタンスなど）を共有する Netty HTTP の複数のルートを持つことができます。ルートが同じ org.jboss.netty.bootstrap.ServerBootstrap インスタンスを共有するため、ルートに多くのブートス

トラップオプションが同じである必要があります。インスタンスは、最初に作成されたルートからのオプションを使用して設定します。

ルートが同一で設定されるオプションは、org.apache.camel.component.netty.NettyServerBootstrapConfiguration 設定クラスで定義されたすべてのオプションです。異なるオプションで別のルートを設定している場合、Camel は起動時に例外を発生させ、オプションが同一ではないことを示します。これを軽減するには、すべてのオプションが同一になるようにします。

以下は、同じポートを共有する 2 つのルートを使用した例です。

2 つのルートが同じポートを共有する

```
from("netty-http:http://0.0.0.0:{{port}}/foo")
  .to("mock:foo")
  .transform().constant("Bye World");

from("netty-http:http://0.0.0.0:{{port}}/bar")
  .to("mock:bar")
  .transform().constant("Bye Camel");
```

以下は、org.apache.camel.component.netty.NettyServerBootstrapConfiguration オプションが 1st ルートと同じではない 2 番目のルートの例になります。これにより、起動時に Camel が失敗します。

同じポートを共有する 2 つのルートがありますが、2 番目のルートの設定が正しく設定され、起動時に失敗します。

```
from("netty-http:http://0.0.0.0:{{port}}/foo")
  .to("mock:foo")
  .transform().constant("Bye World");

// we cannot have a 2nd route on same port with SSL enabled, when the 1st route is NOT
from("netty-http:http://0.0.0.0:{{port}}/bar?ssl=true")
  .to("mock:bar")
  .transform().constant("Bye Camel");
```

228.7.1. 複数のルートでの同じサーバーブートストラップ設定の再使用

org.apache.camel.component.netty.NettyServerBootstrapConfiguration タイプの単一のインスタンスで共通のサーバーブートストラップオプションを設定すると、Netty HTTP コンシューマーで

bootstrapConfiguration オプションを使用して、すべてのコンシューマーで同じオプションを参照および再利用できます。

```
<bean id="nettyHttpBootstrapOptions"
class="org.apache.camel.component.netty.NettyServerBootstrapConfiguration">
  <property name="backlog" value="200"/>
  <property name="connectTimeout" value="20000"/>
  <property name="workerCount" value="16"/>
</bean>
```

また、以下のようにこのオプションを参照するルートで

```
<route>
  <from uri="netty-http:http://0.0.0.0:{{port}}/foo?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>

<route>
  <from uri="netty-http:http://0.0.0.0:{{port}}/bar?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>

<route>
  <from uri="netty-http:http://0.0.0.0:{{port}}/beer?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>
```

228.7.2. OSGi コンテナーで複数のバンドルにまたがる複数のルートを持つ同じサーバーブートストラップ設定を再利用

詳細は、「Netty HTTP Server の例」を参照してください。

228.8. HTTP BASIC 認証の使用

Netty HTTP コンシューマーは、使用するセキュリティーレルム名を指定して HTTP Basic 認証をサポートします。以下に例を示します。

```
<route>
  <from uri="netty-http:http://0.0.0.0:{{port}}/foo?securityConfiguration.realm=karaf"/>
  ...
</route>
```

Basic 認証を有効にするには、レルム名が必須です。デフォルトでは、JAAS ベースのオーセンティケーターが使用されます。これは指定されたレルム名（上記の例のkaraf）を使用し、認証に JAAS レルムと JAAS \{{LoginModule}}s を使用します。

Apache Karaf / ServiceMix のエンドユーザーには、そのまま使える karaf レルムがあります。そのため、上記のサンプルがこれらのコンテナーで追加設定なしで機能しなくなるからです。

228.8.1. Web リソースでの ACL の指定

org.apache.camel.component.netty.http.SecurityConstraint では、Web リソースで制約を定義できます。また、org.apache.camel.component.netty.http.SecurityConstraintMapping は追加設定なしで提供され、ロールの追加や除外を簡単に定義できます。

たとえば、XML DSL で以下のように制約 Bean を定義します。

```
<bean id="constraint" class="org.apache.camel.component.netty.http.SecurityConstraintMapping">
  <!-- inclusions defines url -> roles restrictions -->
  <!-- a * should be used for any role accepted (or even no roles) -->
  <property name="inclusions">
    <map>
      <entry key="/*" value="*"/>
      <entry key="/admin/*" value="admin"/>
      <entry key="/guest/*" value="admin,guest"/>
    </map>
  </property>
  <!-- exclusions is used to define public urls, which requires no authentication -->
  <property name="exclusions">
    <set>
      <value>/public/*</value>
    </set>
  </property>
</bean>
```

上記の制約は、以下のように定義されます。

- /* へのアクセスは制限され、すべてのロールが許可されます（また、ユーザーにロールがない場合も）
- /admin/* へのアクセスには admin ロールが必要です。

- */guest/* へのアクセスには admin ロールまたは guest ロールが必要です。

- */public/* へのアクセスは、認証を必要としない除外されるため、ログインせずにパブリックになります。

この制約を使用するには、以下のように Bean ID を参照する必要があります。

```
<route>
  <from uri="netty-http:http://0.0.0.0:{{port}}/foo?
matchOnUriPrefix=true&amp;securityConfiguration.realm=karaf&amp;securityConfiguration.securityCon
straint=#constraint"/>
  ...
</route>
```

228.9. 関連項目

- Configuring Camel (Camel の設定)

- コンポーネント

- エンドポイント

- はじめに

- Netty

- Netty HTTP サーバーの例

- Jetty

第229章 NETTY4 コンポーネント

Camel バージョン 2.14 から利用可能

Camel の netty4 コンポーネントは、Netty プロジェクトバージョン 4 をベースとしたソケット通信コンポーネントです。
Netty は NIO クライアントサーバーフレームワークで、プロトコルサーバーやクライアントなどの netwServerInitializerFactoryork アプリケーションを迅速かつ簡単に開発できます。
Netty は、TCP、UDP ソケットサーバーなどのネットワークプログラミングを大幅に単純化し、合理化します。

この Camel コンポーネントは、プロデューサーとコンシューマーエンドポイントの両方をサポートします。

Netty コンポーネントには複数のオプションがあり、数多くの TCP/UDP 通信パラメーター（バッファーサイズ、keepAlives、tcpNoDelay など）を細かく制御でき、Camel ルート上の In-Only 通信と In-Out 通信の両方を容易にします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-netty4</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

229.1. URI 形式

netty コンポーネントの URI スキームは以下のようになります。

```
netty4:tcp://localhost:99999[?options]
netty4:udp://remotehost:99999/[?options]
```

このコンポーネントは、TCP と UDP の両方のプロデューサーおよびコンシューマーエンドポイントをサポートします。

URI にクエリーオプションを追加するには、?option=value&option=value&…

229.2. オプション

Netty4 コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
maximumPoolSize (advanced)	使用されている場合は EventExecutorGroup のスレッドプールサイズ。デフォルト値は 16 です。	16	int
Configuration (advanced)	エンドポイントの作成時に NettyConfiguration を設定として使用します。		NettyConfiguration
executorService (advanced)	指定の EventExecutorGroup を使用する場合		EventExecutorGroup
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Netty4 エンドポイントは、URI 構文を使用して設定します。

```
netty4:protocol:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

229.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
protocol	tcp または udp が使用できるプロトコルが **必要です**。		文字列
host	**必須** のホスト名。コンシューマーの場合、ホスト名は localhost または 0.0.0.0 です。プロデューサーの場合、ホスト名はリモートホストで接続するリモートホストになります。		文字列

Name	説明	デフォルト	Type
port	**必須**: ホストのポート番号		int

229.2.2. クエリーパラメーター（72 パラメーター）：

Name	説明	デフォルト	Type
切断 （共通）	Netty チャンネルの使用直後に切断する（クローズ）かどうか。コンシューマーとプロデューサーの両方に使用できます。	false	boolean
keepAlive (common)	非アクティブのためソケットが閉じられないようにする設定	true	boolean
reuseAddress (common)	ソケットの多重化を容易にするための設定	true	boolean
reuseChannel (common)	このオプションを使用すると、プロデューサーとコンシューマー（クライアントモード）は、Exchange の処理ライフサイクルのために同じ Netty チャンネルを再利用できます。これは、Camel ルートでサーバーを複数回呼び出し、同じネットワーク接続を使用する必要がある場合に便利です。このチャネルを使用すると、エクスチェンジが完了するまでチャネルが接続プールには返されません。または disconnect オプションが true に設定されている場合に切断されます。再利用されるチャンネルは、キーリンク NettyConstantsNETTY_CHANNEL を持つエクスチェンジプロパティーとして Exchange に保存され、ルーティング中にチャネルを取得して使用することができます。	false	boolean
同期 （共通）	エンドポイントを一方向または request-response として設定する設定	true	boolean
tcpNoDelay (common)	TCP プロトコルのパフォーマンスを改善するための設定	true	boolean

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
ブロードキャスト（コンシューマー）	UDP でマルチキャストを選択する設定	false	boolean
clientMode (consumer)	clientMode が true の場合、netty コンシューマーはアドレスを TCP クライアントとして接続します。	false	boolean
Reconnect (consumer)	コンシューマーの clientMode でのみ使用されます。これが有効な場合、コンシューマーは接続解除時に再接続を試みます。	true	boolean
reconnectInterval (consumer)	reconnect および clientMode が有効になっている場合に使用されます。再接続を試行する間隔（ミリ秒単位）	10000	int
Back log(consumer)	netty コンシューマー（サーバー）のバックログを設定できます。バックログは、OS に応じてベストエフォートである点に注意してください。このオプションを 200、500、1000 などの値に設定し、TCP スタックに accept キューが適しているかどうかを指示し、このオプションが設定されていない場合、バックログは OS の設定により異なります。		int
bossCount (consumer)	Netty が nio モードで動作している場合、Netty からのデフォルトの bossCount パラメーターを使用します。これは 1 です。ユーザーはこの操作を使用して Netty からデフォルトの bossCount をオーバーライドできます。	1	int
bossGroup (consumer)	NettyEndpoint 全体でサーバー側の新しい接続を処理するために使用できる BossGroup を設定します。		EventLoopGroup
disconnectOnNoReply (consumer)	sync が有効になっている場合、送信先の応答がない場合に NettyConsumer を指定します。	true	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
nettyServerBootstrapFactory (consumer)	カスタム NettyServerBootstrapFactory を使用する場合		NettyServerBootstrap Factory
networkInterface (consumer)	UDP を使用する場合、このオプションを使用して、マルチキャストグループに参加する eth0 などの名前でネットワークインターフェースを指定できます。		文字列
noReplyLogLevel (consumer)	同期が有効な場合、ロギングに使用するログレベルを NettyConsumer に指示し、返信する応答はありません。	WARN	LoggingLevel
serverClosedChannelExceptionCaughtLogLevel (consumer)	サーバー(NettyConsumer)が java.nio.channels.ClosedChannelException を取得する場合、このログレベルを使用してログに記録されます。これは、クライアントが突然切断され、Nettyサーバーで閉じられた例外が一杯になる可能性があるため、閉じられたチャンネル例外のロギングを避けるために使用されます。	DEBUG	LoggingLevel
serverExceptionCaughtLog Level (consumer)	サーバー(NettyConsumer)が例外をキャッチすると、このログレベルを使用してログに記録されます。	WARN	LoggingLevel
serverInitializerFactory (consumer)	カスタムの ServerInitializerFactory を使用する場合		ServerInitializer Factory
usingExecutorService (consumer)	順序付けされたスレッドプールを使用して、同じチャネルでイベントが順序付け処理されるかどうか。	true	boolean
connectTimeout (producer)	ソケット接続が利用可能になるまで待機する時間。値はミリ秒単位です。	10000	int

Name	説明	デフォルト	Type
requestTimeout (producer)	リモートサーバーを呼び出すときに Netty プロデューサーにタイムアウトを使用できます。デフォルトではタイムアウトは使用されていません。値はミリ秒単位であるため、たとえば 30000 は 30 秒です。requestTimeout は Netty の ReadTimeoutHandler を使用してタイムアウトを発生させます。		Long
clientInitializerFactory (producer)	カスタムの ClientInitializerFactory を使用する場合		ClientInitializerFactory
correlationManager (producer)	netty プロデューサーで request/reply を使用すると、カスタム相関マネージャーを使用してリクエストおよびリプライメッセージがマッピングされる方法を管理します。リクエストとリプライメッセージの両方に相関 ID がある場合など、リクエストをリプライとマッピングする方法のみ使用してください。これは、netty で同じチャネル（別名接続）で同時メッセージを複数使用する場合に使用できます。これを実行する場合は、リクエストと応答のメッセージを関連付けて、ルーティングを継続する前に、インフライト Camel Exchange に正しいリプライを保存できます。カスタム相関マネージャーを構築するときに TimeoutCorrelationManagerSupport を拡張することを推奨します。これにより、タイムアウトと、他の複雑な機能も実装する必要があります。詳細は、producerPoolEnabled オプションも参照してください。		NettyCamelStateCorrelationManager
lazyChannelCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合に、例外を回避するためにチャネルをレイジーに作成できます。	true	boolean
producerPoolEnabled (producer)	プロデューサープールが有効かどうか。重要： これをオフにした場合には、プロデューサーに単一の共有接続が使用されます。また、リクエスト/リプライを行う場合でも使用します。つまり、応答が順不同でなければ、インターリーブレスポンスで問題が発生する可能性があることを意味します。そのため、リクエストとリプライメッセージの両方に相関 ID が必要になります。これにより、Camel でメッセージの処理を継続する Camel コールバックに返信を適切に関連付けることができます。そのためには、NettyCamelStateCorrelationManager を相関マネージャーとして実装し、correlationManager オプションを使用して設定する必要があります。詳細は、correlationManager オプションも参照してください。	true	boolean

Name	説明	デフォルト	Type
producerPoolMaxActive (producer)	特定の時間にプールによって割り当て可能なオブジェクト数の上限を設定します（クライアントにチェックするか、待機しているオブジェクト数）。制限なしに負の値を使用してください。	-1	int
producerPoolMaxIdle (producer)	プール内のアイドルインスタンスの数の上限を設定します。	100	int
producerPoolMinEvictable Idle (producer)	アイドルオブジェクトによりエビクションの対象となる前にオブジェクトがプール内にアイドル状態である可能性がある最小時間（ミリ秒単位）を設定します。	300000	Long
producerPoolMinIdle (producer)	エビクトスレッド（アクティブな場合）が新規オブジェクトを生成する前に、プロデューサープールで許可されるインスタンスの最小数を設定します。		int
udpConnectionlessSending (producer)	このオプションは、実際の火災報知能である接続が少ない udp 送信をサポートします。接続された udp は、受信側ポートをリッスンしていない場合は PortUnreachableException を受け取ります。	false	boolean
useByteBuf (producer)	useByteBuf が true の場合、netty プロデューサーはメッセージボディーを ByteBuf に変換してから送信します。	false	boolean
allowSerializedHeaders (advanced)	transferExchange が true の場合にのみ TCP に使用されます。true に設定すると、ヘッダーおよびプロパティーでシリアライズ可能なオブジェクトがエクスチェンジに追加されます。それ以外の場合は、Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。	false	boolean
bootstrapConfiguration (advanced)	このエンドポイントを設定するには、カスタムに設定された NettyServerBootstrapConfiguration を使用します。		NettyServerBootstrap Configuration
channelGroup (advanced)	明示的な ChannelGroup を使用します。		ChannelGroup

Name	説明	デフォルト	Type
nativeTransport (advanced)	NIO の代わりにネイティブトランスポートを使用するかどうか。ネイティブトランスポートはホストのオペレーティングシステムを活用し、一部のプラットフォームでのみサポートされます。使用しているホストオペレーティングシステムの netty JAR を追加する必要があります。詳細は、http://netty.io/wiki/native-transports.htmlを参照してください。	false	boolean
オプション （詳細）	オプション. を接頭辞として使用して追加の netty オプションを設定できます。たとえば、options.child.keepAlive=false を指定して netty オプションの child.keepAlive=false を設定します。使用できるオプションについては、Netty のドキュメントを参照してください。		マップ
receiveBufferSize (advanced)	受信接続中に使用される TCP/UDP バッファーサイズ。サイズはバイト単位です。	65536	int
receiveBufferSizePredictor (advanced)	バッファーサイズの予測を設定します。詳細は、Jetty のドキュメント およびこのメールスレッド を参照してください。		int
sendBufferSize (advanced)	アウトバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイト単位です。	65536	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
transferExchange (advanced)	TCP にのみ使用されます。ボディーだけでなく、ネットワーク上でエクスチェンジを転送できます。次のフィールドが転送されます。本文、Out ボディー、フォールトボディー、ヘッダー、送信ヘッダー、エクスチェンジプロパティー、エクスチェンジ例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。	false	boolean
udpByteArrayCodec (advanced)	UDP のみの場合Java シリアライゼーションプロトコルの代わりにバイトアレイコーデックの使用が有効になっている場合。	false	boolean
workerCount (advanced)	Netty が nio モードで機能する場合、Netty からのデフォルトの workerCount パラメーターを使用します。これは cpu_core_threads2 です。ユーザーはこの操作を使用して Netty からデフォルトの workerCount をオーバーライドできます。		int

Name	説明	デフォルト	Type
workerGroup (advanced)	明示的な EventLoopGroup を boss スレッドプールとして使用する。たとえば、複数のコンシューマーまたはプロデューサーとスレッドプールを共有する場合などです。デフォルトでは、各コンシューマーまたはプロデューサーには 2 x cpu の数コアスレッドを持つ独自のワーカープールがあります。		EventLoopGroup
allowDefaultCodec (codec)	netty コンポーネントは、エンコーダー/デックが null で、テキストラインが false の場合、デフォルトのコーデックをインストールします。allowDefaultCodec を false に設定すると、netty コンポーネントがフィルターチェーンの最初の要素としてデフォルトのコーデックをインストールできなくなります。	true	boolean
autoAppendDelimiter (codec)	テキストラインコーデックを使用して送信する際に、欠落している終了区切り文字を自動追加するかどうか。	true	boolean
decoder (codec)	インバウンドペイロードの特別なマーシャリングを実行するために使用できるカスタム ChannelHandler クラス。		ChannelHandler
decoderMaxLineLength (codec)	テキストラインコーデックに使用する最大行長。	1024	int
decoders (codec)	使用されるデコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値をプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。		文字列
delimiter (codec)	テキストラインコーデックに使用する区切り文字。使用できる値は LINE と NULL です。	LINE	TextLineDelimiter
encoder (codec)	アウトバウンドペイロードの特別なマーシャリングを実行するために使用できるカスタム ChannelHandler クラス。		ChannelHandler
encoders (codec)	使用されるエンコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値をプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。		文字列

Name	説明	デフォルト	Type
encoding (codec)	テキストラインコーデックに使用するエンコーディング（文字セット名）。指定されていない場合、Camel は JVM デフォルトの Charset を使用します。		文字列
テキストライン（コード）	TCP にのみ使用されます。codec が指定されていない場合、このフラグを使用してテキスト行ベースのコーデックを指定できます。指定されていない場合や、値が false の場合は、Object Serialization は TCP を介して想定されます。	false	boolean
enabledProtocols (security)	SSL を使用する際に有効にするプロトコル	TLSv1, TLSv1.1, TLSv1.2	文字列
keyStoreFile (security)	暗号化に使用されるクライアント側の証明書キーストア		ファイル
keyStoreFormat (security)	ペイロードの暗号化に使用するキーストア形式。設定されていない場合、デフォルトで JKS に設定されます。		文字列
keyStoreResource (security)	暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		文字列
needClientAuth (security)	SSL の使用時にサーバーがクライアント認証を必要とするかどうかを設定します。	false	boolean
passphrase （セキュリティー）	SSH を使用して送信されたペイロードの暗号化/復号化に使用するパスワード設定		文字列
securityProvider (security)	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合、デフォルトは SunX509 に設定されます。		文字列
ssl （セキュリティー）	SSL 暗号化がこのエンドポイントに適用されるかどうかを指定する設定	false	boolean
sslClientCertHeaders (security)	有効にすると、Netty コンシューマーはサブジェクト名、発行者名、シリアル番号、有効な日付範囲などのクライアント証明書に関する情報を持つヘッダーで Camel メッセージを強化します。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters

Name	説明	デフォルト	Type
sslHandler (security)	SSL ハンドラーを返すために使用できるクラスへの参照		SslHandler
trustStoreFile (security)	暗号化に使用されるサーバー側の証明書キーストア		ファイル
trustStoreResource (security)	暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		文字列

229.3. レジストリーベースのオプション

Codec ハンドラーおよび SSL キーストアは、Spring XML ファイルなど、レジストリーに登録できます。渡すことができる値は次のとおりです。

Name	説明
passphrase	SSH を使用して送信されたペイロードの暗号化/復号化に使用するパスワード設定
keyStoreFormat	ペイロードの暗号化に使用するキーストア形式。設定されていない場合は、デフォルトで「JKS」に設定されます。
securityProvider	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合は、デフォルトで "SunX509" に設定されます。
keyStoreFile	**非推奨：** 暗号化に使用されるクライアント側の証明書キーストア
trustStoreFile	**非推奨：** 暗号化に使用されるサーバー側の証明書キーストア
keyStoreResource	Camel 2.11.1: 暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、**「classpath:」、"file:"** 、または **"http:"** のプレフィックスを指定して、異なるシステムからリソースをロードすることもできます。

Name	説明
trustStoreResource	Camel 2.11.1: 暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、「**classpath:**」、**"file:"** 、または **"http:"** のプレフィックスを指定して、異なるシステムからリソースをロードすることもできます。
sslHandler	SSL ハンドラーを返すために使用できるクラスへの参照
encoder	アウトバウンドペイロードの特別なマーシャリングの実行に使用できるカスタム **ChannelHandler** クラス。Must override io.netty.channel.ChannelInboundHandlerAdapter.
encoders	使用されるエンコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値を # のプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。
decoder	インバウンドペイロードの特別なマーシャリングの実行に使用できるカスタム **ChannelHandler** クラス。Must override io.netty.channel.ChannelOutboundHandlerAdapter.
decoders	使用されるデコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値を # のプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。

注記

共有不可能なエンコーダー/デコーダーの使用については、以下を参照してください。

229.3.1. 共有不可能なエンコーダーまたはデコーダーの使用

エンコーダーまたはデコーダーが共有できない場合（@Shareable クラスアノテーションなど）、エンコーダー/デコーダーは org.apache.camel.component.netty.ChannelHandlerFactory インターフェースを実装し、newChannelHandler メソッドで新規インスタンスを返す必要があります。これは、エンコーダー/デコーダーを安全に使用できるようにするためです。そうでない場合、Netty コンポーネントはエンドポイントの作成時に WARN をログに記録します。

Netty コンポーネントは、一般的に使用されるメソッドが多数含まれる org.apache.camel.component.netty.ChannelHandlerFactories ファクトリークラスを提供します。

229.4. NETTY エンドポイントとの/からのメッセージの送信

229.4.1. Netty プロデューサー

Producer モードでは、コンポーネントは TCP プロトコルまたは UDP プロトコル（任意の SSL サポート）を使用して、ペイロードをソケットエンドポイントに送信する機能を提供します。

プロデューサーモードは、一方向およびリクエスト応答ベースの操作の両方をサポートします。

229.4.2. Netty コンシューマー

Consumer モードでは、コンポーネントは以下を行う機能を提供します。

- TCP プロトコルまたは UDP プロトコル（任意の SSL サポートあり）を使用して、指定のソケットをリッスンします。
- text/xml、バイナリー、およびシリアライズされたオブジェクトベースのペイロードを使用してソケットでリクエストを受信する。
- メッセージエクスチェンジとしてルート経由で送信します。

コンシューマーモードは、一方向およびリクエスト応答ベースの操作の両方をサポートします。

229.5. 例

229.5.1. リクエスト応答およびシリアライズされたオブジェクトペイロードを使用した UDP Netty エンドポイント

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
    from("netty4:udp://localhost:5155?sync=true")
      .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          Poetry poetry = (Poetry) exchange.getIn().getBody();
          poetry.setPoet("Dr. Sarojini Naidu");
          exchange.getOut().setBody(poetry);
        }
      }
    }
};
```

229.5.2. 一方向通信を使用した TCP ベースの Netty コンシューマーエンドポイント

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
      from("netty4:tcp://localhost:5150")
          .to("mock:result");
  }
};
```

229.5.3. リクエスト応答通信を使用した SSL/TCP ベースの Netty コンシューマーエンドポイント

JSSE 設定ユーティリティーの使用

Camel 2.9 の時点で、Netty コンポーネントは Camel JSSE 設定ユーティリティーを介して SSL/TLS 設定をサポートします。 このユーティリティーは、エンドポイントおよびコンポーネントレベルで記述し、設定する必要のあるコンポーネント固有のコードの量を大幅に削減します。 以下の例は、Netty コンポーネントでユーティリティーを使用する方法を示しています。

コンポーネントのプログラムによる設定

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

NettyComponent nettyComponent = getContext().getComponent("netty4",
NettyComponent.class);
nettyComponent.setSslContextParameters(scp);
```

エンドポイントの Spring DSL ベースの設定

```
...
  <camel:sslContextParameters
      id="sslContextParameters">
    <camel:keyManagers
        keyPassword="keyPassword">
      <camel:keyStore
          resource="/users/home/server/keystore.jks"
          password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
```

```
  <to uri="netty4:tcp://localhost:5150?
sync=true&ssl=true&sslContextParameters=#sslContextParameters"/>
...
```

[[Netty4-UsingBasicSSL/TLSconfigurationontheJettyComponent]] Jetty コンポーネントの基本的な SSL/TLS 設定の使用

```
JndiRegistry registry = new JndiRegistry(createJndiContext());
registry.bind("password", "changeit");
registry.bind("ksf", new File("src/test/resources/keystore.jks"));
registry.bind("tsf", new File("src/test/resources/keystore.jks"));

context.createRegistry(registry);
context.addRoutes(new RouteBuilder() {
 public void configure() {
    String netty_ssl_endpoint =
      "netty4:tcp://localhost:5150?sync=true&ssl=true&passphrase=#password"
      + "&keyStoreFile=#ksf&trustStoreFile=#tsf";
    String return_string =
      "When You Go Home, Tell Them Of Us And Say,"
      + "For Your Tomorrow, We Gave Our Today.";

    from(netty_ssl_endpoint)
     .process(new Processor() {
       public void process(Exchange exchange) throws Exception {
         exchange.getOut().setBody(return_string);
       }
     }
 }
});
```

SSLSession およびクライアント証明書へのアクセス

クライアント証明書に関する詳細を取得する必要がある場合は、javax.net.ssl.SSLSession にアクセスできます。ssl=true の場合、Netty4 コンポーネントは以下のように SSLSession を Camel メッセージにヘッダーとして保存します。

```
SSLSession session = exchange.getIn().getHeader(NettyConstants.NETTY_SSL_SESSION,
SSLSession.class);
// get the first certificate which is client certificate
javax.security.cert.X509Certificate cert = session.getPeerCertificateChain()[0];
Principal principal = cert.getSubjectDN();
```

クライアントを認証するために needClientAuth=true を設定するのを忘れないようにしてください。それ以外の場合、SSLSession はクライアント証明書に関する情報にアクセスできないため、例外 javax.net.ssl.SSLPeerUnverifiedException: peer not authenticated が発生することがあります。ま

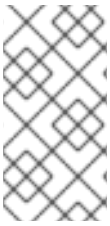
た、クライアント証明書の有効期限が切れるか、有効でない場合はこの例外を取得することもできます。

ヒント

sslClientCertHeaders オプションを true に設定し、クライアント証明書に関する詳細が含まれるヘッダーを使用して Camel メッセージを強化できます。たとえば、サブジェクト名は CamelNettySSLClientCertSubjectName ヘッダーで読み取り可能です。

229.5.4. 複数のコーデックの使用

場合によっては、エンコーダーとデコーダーのチェーンを netty パイプラインに追加する必要がある場合があります。multpile codecs を camel netty エンドポイントに追加するには、'encoders' および 'decoders' uri パラメーターを使用する必要があります。'encoder' および 'decoder' パラメーターと同様に、パイプラインに追加する必要のある参照（ChannelUpstreamHandlers および ChannelDownstreamHandler のリスト）を指定するために使用されます。エンコーダーが指定されている場合は、デコーダーやデコーダーパラメーターと同様にエンコーダーパラメーターが無視されることに注意してください。



注記

共有不可能なエンコーダー/デコーダーの使用についてさらに詳しくお読みください。

エンドポイントの作成時に解決できるようにコーデックのリストを Camel のレジストリーに追加する必要があります。

```
ChannelHandlerFactory lengthDecoder =
ChannelHandlerFactories.newLengthFieldBasedFrameDecoder(1048576, 0, 4, 0, 4);

StringDecoder stringDecoder = new StringDecoder();
registry.bind("length-decoder", lengthDecoder);
registry.bind("string-decoder", stringDecoder);

LengthFieldPrepender lengthEncoder = new LengthFieldPrepender(4);
StringEncoder stringEncoder = new StringEncoder();
registry.bind("length-encoder", lengthEncoder);
registry.bind("string-encoder", stringEncoder);

List<ChannelHandler> decoders = new ArrayList<ChannelHandler>();
decoders.add(lengthDecoder);
decoders.add(stringDecoder);

List<ChannelHandler> encoders = new ArrayList<ChannelHandler>();
encoders.add(lengthEncoder);
```

```
encoders.add(stringEncoder);

registry.bind("encoders", encoders);
registry.bind("decoders", decoders);
```

Spring のネイティブコレクションサポートを使用して、アプリケーションコンテキストでコーデックリストを指定することができます。

```
<util:list id="decoders" list-class="java.util.LinkedList">
        <bean class="org.apache.camel.component.netty4.ChannelHandlerFactories" factory-method="newLengthFieldBasedFrameDecoder">
            <constructor-arg value="1048576"/>
            <constructor-arg value="0"/>
            <constructor-arg value="4"/>
            <constructor-arg value="0"/>
            <constructor-arg value="4"/>
        </bean>
        <bean class="io.netty.handler.codec.string.StringDecoder"/>
    </util:list>

    <util:list id="encoders" list-class="java.util.LinkedList">
        <bean class="io.netty.handler.codec.LengthFieldPrepender">
            <constructor-arg value="4"/>
        </bean>
        <bean class="io.netty.handler.codec.string.StringEncoder"/>
    </util:list>

    <bean id="length-encoder" class="io.netty.handler.codec.LengthFieldPrepender">
        <constructor-arg value="4"/>
    </bean>
    <bean id="string-encoder" class="io.netty.handler.codec.string.StringEncoder"/>

    <bean id="length-decoder" class="org.apache.camel.component.netty4.ChannelHandlerFactories" factory-method="newLengthFieldBasedFrameDecoder">
        <constructor-arg value="1048576"/>
        <constructor-arg value="0"/>
        <constructor-arg value="4"/>
        <constructor-arg value="0"/>
        <constructor-arg value="4"/>
    </bean>
    <bean id="string-decoder" class="io.netty.handler.codec.string.StringDecoder"/>
```

その後、bean 名は netty エンドポイント定義で、コンマ区切りのリストとして、または List（例：List）を含めることができます。

```
 from("direct:multiple-codec").to("netty4:tcp://localhost:{{port}}?encoders=#encoders&sync=false");

 from("netty4:tcp://localhost:{{port}}?decoders=#length-decoder,#string-decoder&sync=false").to("mock:multiple-codec");
```

または XML を使用します。

```
<camelContext id="multiple-netty-codecs-context" xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:multiple-codec"/>
        <to uri="netty4:tcp://localhost:5150?encoders=#encoders&amp;sync=false"/>
    </route>
    <route>
        <from uri="netty4:tcp://localhost:5150?decoders=#length-decoder,#string-
decoder&amp;sync=false"/>
        <to uri="mock:multiple-codec"/>
    </route>
</camelContext>
```

229.6. 完了後にチャンネルを閉じる

サーバーとして動作する場合は、クライアント変換の完了時にチャネルを閉じる必要がある場合があります。
これは、endpoint オプション disconnect=true を設定するだけで実行できます。

ただし、以下のようにメッセージごとに Camel に指示することもできます。
Camel にチャネルを閉じるよう指示するには、CamelNettyCloseChannelWhenComplete キーでヘッダーをブール値 true に追加する必要があります。
たとえば、以下の例では bye メッセージをクライアントに書き直した後にチャンネルを閉じます。

```
from("netty4:tcp://localhost:8080").process(new Processor() {
    public void process(Exchange exchange) throws Exception {
        String body = exchange.getIn().getBody(String.class);
        exchange.getOut().setBody("Bye " + body);
        // some condition which determines if we should close
        if (close) {

exchange.getOut().setHeader(NettyConstants.NETTY_CLOSE_CHANNEL_WHEN_COMPLETE,
true);
        }
    }
});
```

カスタムチャネルパイプラインファクトリーを追加して、以下を完全に制御

229.7. カスタムパイプライン

カスタムチャンネルパイプラインは、カスタムハンドラー、エンコーダー(s)、デコーダーを挿入することで、ハンドラー/インターセプターチェーン上で完全な制御を提供します。

カスタムパイプラインを追加するには、コンテキストレジストリー（JNDIRegistry、または camel-spring ApplicationContextRegistry など）を使用してコンテキストで、カスタムチャネルパイプラインファクトリーを作成して登録する必要があります。

カスタムパイプラインファクトリーは、以下のように構築する必要があります。

- プロデューサーにリンクされたチャネルパイプラインファクトリーは、抽象クラス ClientPipelineFactory を拡張する必要があります。
- コンシューマーにリンクされたチャネルパイプラインファクトリーは、抽象クラス ServerInitializerFactory を拡張する必要があります。
- クラスは、カスタムハンドラー、エンコーダー、およびデコーダーを挿入するために、initChannel（）メソッドをオーバーライドする必要があります。initChannel（） メソッドを上書きすると、パイプラインに接続しているハンドラー、エンコーダー、またはデコーダーのないパイプラインが作成されます。

以下の例は、ServerInitializerFactory ファクトリーの作成方法を示しています。

229.7.1. カスタムパイプラインファクトリーの使用

```
public class SampleServerInitializerFactory extends ServerInitializerFactory {
    private int maxLineSize = 1024;

    protected void initChannel(Channel ch) throws Exception {
        ChannelPipeline channelPipeline = ch.pipeline();

        channelPipeline.addLast("encoder-SD", new StringEncoder(CharsetUtil.UTF_8));
        channelPipeline.addLast("decoder-DELIM", new
DelimiterBasedFrameDecoder(maxLineSize, true, Delimiters.lineDelimiter()));
        channelPipeline.addLast("decoder-SD", new StringDecoder(CharsetUtil.UTF_8));
        // here we add the default Camel ServerChannelHandler for the consumer, to allow Camel
to route the message etc.
        channelPipeline.addLast("handler", new ServerChannelHandler(consumer));
    }
}
```

その後、カスタムチャネルパイプラインファクトリーをレジストリーに追加し、以下の方法で camel ルートでインスタンス化/使用率を使用できるようにします。

```
Registry registry = camelContext.getRegistry();
```

```
ServerInitializerFactory factory = new TestServerInitializerFactory();
registry.bind("spf", factory);
context.addRoutes(new RouteBuilder() {
  public void configure() {
     String netty_ssl_endpoint =
       "netty4:tcp://localhost:5150?serverInitializerFactory=#spf"
     String return_string =
       "When You Go Home, Tell Them Of Us And Say,"
       + "For Your Tomorrow, We Gave Our Today.";

     from(netty_ssl_endpoint)
      .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          exchange.getOut().setBody(return_string);
        }
      }
  }
});
```

229.8. NETTY ボットおよびワーカースレッドプールの再利用

Netty には、boss と worker の 2 種類のスレッドプールがあります。デフォルトでは、各 Netty コンシューマーおよびプロデューサーにはプライベートスレッドプールがあります。複数のコンシューマーまたはプロデューサー間でこれらのスレッドプールを再利用する場合は、スレッドプールを作成し、レジストリーに登録する必要があります。

たとえば Spring XML を使用する場合、以下のように 2 つのワーカースレッドを持つ NettyWorkerPoolBuilder を使用して共有ワーカースレッドプールを作成できます。

```
<!-- use the worker pool builder to help create the shared thread pool -->
<bean id="poolBuilder" class="org.apache.camel.component.netty.NettyWorkerPoolBuilder">
  <property name="workerCount" value="2"/>
</bean>

<!-- the shared worker thread pool -->
<bean id="sharedPool" class="org.jboss.netty.channel.socket.nio.WorkerPool"
      factory-bean="poolBuilder" factory-method="build" destroy-method="shutdown">
</bean>
```

ヒント

boss スレッドプールには、Netty コンシューマー用の org.apache.camel.component.netty4.NettyServerBossPoolBuilder ビルダーと、Netty プロデューサー用の org.apache.camel.component.netty4.NettyClientBossPoolBuilder があります。

その後、以下のように URI に workerPool オプションを設定して、このワーカープールを参照できます。

```
<route>
  <from uri="netty4:tcp://localhost:5021?
textline=true&amp;sync=true&amp;workerPool=#sharedPool&amp;usingExecutorService=false"/>
  <to uri="log:result"/>
  ...
</route>
```

別のルートがある場合は、共有ワーカープールを参照できます。

```
<route>
  <from uri="netty4:tcp://localhost:5022?
textline=true&amp;sync=true&amp;workerPool=#sharedPool&amp;usingExecutorService=false"/>
  <to uri="log:result"/>
  ...
</route>
```

などになります。

229.9. 要求/応答のある単一接続で同時メッセージの多重化

netty プロデューサー経由でリクエスト/リプライメッセージングに Netty を使用する場合、デフォルトでは各メッセージは共有されていない接続（プールされた）経由で送信されます。これにより、Camel でさらにルーティングするために、返信が自動的に正しいリクエストスレッドにマップできるようになります。つまり、リクエスト/リプライメッセージ間の相関は、リクエストの送信に使用されたのと同じ接続で返信されるため、追加設定なしで行われます。この接続は他の接続と共有されません。応答が返されると、接続は接続プールに戻され、他のユーザーが再利用できます。

ただし、単一の共有接続で同時リクエスト/レスポンスを多重化する場合は、producerPoolEnabled=false を設定して接続プールをオフにする必要があります。今回のリリースより、応答が順不同に発生すると、インターリーブレスポンスで問題が発生する可能性があることを意味します。そのため、リクエストとリプライメッセージの両方に相関 ID が必要になります。これにより、Camel でメッセージの処理を継続する Camel コールバックに返信を適切に関連付けることができます。これを行うには、NettyCamelStateCorrelationManager を相関マネージャーとして実装し、correlationManager=#myManager オプションを使用して設定する必要があります。

注記

カスタム相関マネージャーを構築するときに TimeoutCorrelationManagerSupport を拡張することを推奨します。これにより、タイムアウトと、他の複雑な機能も実装する必要があります。

229.10. 関連項目

- *Netty HTTP*
- *MINA*

第230章 NETTY4 HTTP コンポーネント

Camel バージョン 2.14 から利用可能

netty4-http コンポーネントは Netty4 で HTTP トランスポートを調整する Netty4 コンポーネントのエクステンションです。

この Camel コンポーネントは、プロデューサーとコンシューマーエンドポイントの両方をサポートします。

INFO: Stream.Netty はストリームベースであるため、受信する入力はストリームとして Camel に送信されます。つまり、は 1 回 のみストリームのコンテンツを読み取ることができます。メッセージボディーが空であるか、データに複数回アクセスする必要がある場合（マルチキャストの実行や再配信エラー処理など）、ストリームキャッシュを使用するか、メッセージボディーを複数回再読み取りできる String に変換する必要があります。Netty4 HTTP は、io.netty.handler.codec.http.HttpObjectAggregator を使用してストリーム全体をメモリーに読み取り、完全な http メッセージ全体を構築することに注意してください。ただし、生成されるメッセージは、1 度に判読できるストリームベースのメッセージになります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-netty4-http</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

230.1. URI 形式

netty コンポーネントの URI スキームは以下のようになります。

```
netty4-http:http://localhost:8080[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

INFO: Query parameters vs endpoint options.Camel が URI クエリーパラメーターおよびエンドポイントオプションを認識する方法について理解している場合があります。たとえば、エンドポイント URI を - netty4-http:http/example.com?myParam=myValue&compression=true のように作成でき

ます。この例では、myParam は HTTP パラメーターですが、圧縮 は Camel エンドポイントオプションです。このような状況で Camel によって使用されるストラテジーは、利用可能なエンドポイントオプションを解決し、URI から削除することです。この例では、Netty HTTP プロデューサーによってエンドポイントに送信される HTTP リクエストは、 http//example.com?myParam=myValue のように表示されます。圧縮 エンドポイントオプションはターゲット URL から解決され、削除されるためです。また、動的ヘッダー（CamelHttpQueryなど）を使用してエンドポイントオプションを指定できない点に留意してください。エンドポイントオプションは、エンドポイント URI 定義レベルでのみ指定できます（DSL 要素 に対する ような）。

230.2. HTTP オプション

INFO: 多くのオプション。重要： このコンポーネントは Netty4 のすべてのオプションを継承します。そのため、 Netty4 ドキュメント も確認してください。
UDP トランスポートに関連するオプションなど、この Netty4 HTTP コンポーネントを使用する場合は、Netty4 のいくつかのオプションは適用されないことに注意してください。

Netty4 HTTP コンポーネントは、以下に示す 8 つのオプションをサポートします。

Name	説明	デフォルト	Type
nettyHttpBinding (advanced)	Netty および Camel Message API への/からのバインディングにカスタムの org.apache.camel.component.netty4.http.NettyHttpBinding を使用するには、以下を行います。		NettyHttpBinding
設定 （共通）	エンドポイントの作成時に NettyConfiguration を設定として使用します。		NettyHttpConfiguration
headerFilterStrategy (advanced)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用してヘッダーをフィルターします。		HeaderFilterStrategy
securityConfiguration （セキュリティー）	セキュアな Web リソースを設定するための org.apache.camel.component.netty4.http.NettyHttpSecurityConfiguration を参照します。		NettyHttpSecurityConfiguration
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
maximumPoolSize (advanced)	使用されている場合は EventExecutorGroup のスレッドプールサイズ。デフォルト値は 16 です。	16	int
executorService (advanced)	指定の EventExecutorGroup を使用する場合		EventExecutorGroup

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Netty4 HTTP エンドポイントは、URI 構文を使用して設定します。

```
netty4-http:protocol:host:port/path
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

230.2.1. パスパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
protocol	**必須**: http または https のいずれかを使用するプロトコル。		文字列
host	localhost などのローカルホスト名、またはコンシューマーの場合は 0.0.0.0 が **必要** になります。プロデューサーの使用時にリモート HTTP サーバーのホスト名。		文字列
port	ホストのポート番号		int
path	リソースパス		文字列

230.2.2. クエリーパラメーター（79 パラメーター）：

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
bridgeEndpoint (common)	オプションが true の場合、プロデューサーは Exchange.HTTP_URI ヘッダーを無視し、リクエストにエンドポイントの URI を使用します。また、throwExceptionOnFailure を false に設定して、プロデューサーが障害応答をすべて返信するようにすることもできます。ブリッジモードで動作しているコンシューマーは、gzip 圧縮および WWW URL フォームエンコーディングを省略します（Exchange.SKIP_GZIP_ENCODING および Exchange.SKIP_WWW_FORM_URLENCODED ヘッダーを消費したエクスチェンジに追加します）。	false	boolean
切断 （共通）	Netty チャンネルの使用直後に切断する（クローズ）かどうか。コンシューマーとプロデューサーの両方に使用できます。	false	boolean
keepAlive (common)	非アクティブのためソケットが閉じられないようにする設定	true	boolean
reuseAddress (common)	ソケットの多重化を容易にするための設定	true	boolean
reuseChannel (common)	このオプションを使用すると、プロデューサーとコンシューマー（クライアントモード）は、Exchange の処理ライフサイクルのために同じ Netty チャンネルを再利用できます。これは、Camel ルートでサーバーを複数回呼び出し、同じネットワーク接続を使用する必要がある場合に便利です。このチャネルを使用すると、エクスチェンジが完了するまでチャネルが接続プールには返されません。または disconnect オプションが true に設定されている場合に切断されます。再利用されるチャンネルは、キーリンク NettyConstantsNETTY_CHANNEL を持つエクスチェンジプロパティーとして Exchange に保存され、ルーティング中にチャネルを取得して使用することができます。	false	boolean
同期 （共通）	エンドポイントを一方向または request-response として設定する設定	true	boolean
tcpNoDelay (common)	TCP プロトコルのパフォーマンスを改善するための設定	true	boolean

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
matchOnUriPrefix (consumer)	完全一致がない場合、Camel が URI プレフィックスと一致するターゲットコンシューマーの検索を試行するかどうか。	false	boolean
send503whenSuspended (consumer)	コンシューマーが停止した場合に HTTP ステータスコード 503 を送信するかどうか。オプションが false の場合、コンシューマーが中断された場合に Netty Acceptor がバインド解除されるため、クライアントは接続できなくなります。	true	boolean
Back log(consumer)	netty コンシューマー（サーバー）のバックログを設定できます。バックログは、OS に応じてベストエフォートである点に注意してください。このオプションを 200、500、1000 などの値に設定し、TCP スタックに accept キューが適しているかどうかを指示し、このオプションが設定されていない場合、バックログは OS の設定により異なります。		int
bossCount (consumer)	Netty が nio モードで動作している場合、Netty からのデフォルトの bossCount パラメーターを使用します。これは 1 です。ユーザーはこの操作を使用して Netty からデフォルトの bossCount をオーバーライドできます。	1	int
bossGroup (consumer)	NettyEndpoint 全体でサーバー側の新しい接続を処理するために使用できる BossGroup を設定します。		EventLoopGroup
chunkedMaxContentLength (consumer)	Netty HTTP サーバーで受信されるチャンクされたフレームごとの最大コンテンツの長さ（バイト単位）。	1048576	int
圧縮 （コンシューマー）	クライアントが HTTP ヘッダーからサポートしている場合は、Netty HTTP サーバーで圧縮に gzip/deflate の使用を許可します。	false	boolean
disconnectOnNoReply (consumer)	sync が有効になっている場合、送信先の応答がない場合に NettyConsumer を指定します。	true	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
httpMethodRestrict (consumer)	Netty HTTP コンシューマーで HTTP メソッドを無効にします。カンマで区切って複数指定できます。		文字列
mapHeaders (consumer)	このオプションを有効にすると、Netty から Camel メッセージへのバインディング時にヘッダーもマッピングされます（ヘッダーとして Camel メッセージに追加されます）。このオプションを無効にして無効にすることができます。ヘッダーは、Netty HTTP リクエスト io.netty.handler.codec.HttpRequest インスタンスを返す getHttpRequest（）メソッドを使用して、org.apache.camel.component.netty.http.NettyHttpMessage メッセージからアクセスできます。	true	boolean
maxHeaderSize (consumer)	すべてのヘッダーの最大長。各ヘッダーの長さの合計がこの値を超えると、io.netty.handler.codec.TooLongFrameException が発生します。	8192	int
nettyServerBootstrapFactory (consumer)	カスタム NettyServerBootstrapFactory を使用する場合		NettyServerBootstrap Factory
nettySharedHttpServer (consumer)	共有された Netty HTTP サーバーを使用します。詳細は、「Netty HTTP Server の例」を参照してください。		NettySharedHttpServer
noReplyLogLevel (consumer)	同期が有効な場合、ロギングに使用するログレベルを NettyConsumer に指示し、返信する応答はありません。	WARN	LoggingLevel
serverClosedChannelExceptionCaughtLogLevel (consumer)	サーバー(NettyConsumer)が java.nio.channels.ClosedChannelException を取得する場合、このログレベルを使用してログに記録されます。これは、クライアントが突然切断され、Nettyサーバーで閉じられた例外が一杯になる可能性があるため、閉じられたチャンネル例外のロギングを避けるために使用されます。	DEBUG	LoggingLevel

Name	説明	デフォルト	Type
serverExceptionCaughtLog Level (consumer)	サーバー(NettyConsumer)が例外をキャッチすると、このログレベルを使用してログに記録されます。	WARN	LoggingLevel
serverInitializerFactory (consumer)	カスタムの ServerInitializerFactory を使用する場合		ServerInitializerFactory
traceEnabled (consumer)	この Netty HTTP コンシューマーの HTTP TRACE を有効にするかどうかを指定します。デフォルトでは、TRACE はオフになっています。	false	boolean
urlDecodeHeaders (consumer)	このオプションが有効な場合には、Netty から Camel メッセージへのバインディング時にヘッダー値がデコードされます（例： %20 はスペース文字です）。このオプションは、デフォルトの org.apache.camel.component.netty.http.NettyHttpBinding によって使用されるため、カスタムの org.apache.camel.component.netty4.http.NettyHttpBinding を実装する場合は、このオプションに応じてヘッダーをデコードする必要があります。	false	boolean
usingExecutorService (consumer)	順序付けされたスレッドプールを使用して、同じチャネルでイベントが順序付け処理されるかどうか。	true	boolean
connectTimeout (producer)	ソケット接続が利用可能になるまで待機する時間。値はミリ秒単位です。	10000	int
cookieHandler (producer)	HTTP セッションを維持するためのクッキーハンドラーの設定		CookieHandler
requestTimeout (producer)	リモートサーバーを呼び出すときに Netty プロデューサーにタイムアウトを使用できます。デフォルトではタイムアウトは使用されていません。値はミリ秒単位であるため、たとえば 30000 は 30 秒です。requestTimeout は Netty の ReadTimeoutHandler を使用してタイムアウトを発生させます。		Long
throwExceptionOnFailure (producer)	リモートサーバーからの応答に失敗した場合に HttpOperationFailedException のスローを無効にするオプション。これにより、HTTP ステータスコードに関係なくすべての応答を取得できます。	true	boolean
clientInitializerFactory (producer)	カスタムの ClientInitializerFactory を使用する場合		ClientInitializerFactory

Name	説明	デフォルト	Type
lazyChannelCreation (producer)	Camel プロデューサーの起動時にリモートサーバーが稼働していない場合に、例外を回避するためにチャネルをレイジーに作成できます。	true	boolean
okStatusCodeRange (producer)	正常な応答とみなされるステータスコード。値は含まれます。コンマで区切られた複数の範囲を定義できます（例： 200-204,209,301-304）。各範囲は、1 つの数字またはダッシュを含む from から でなければなりません。デフォルトの範囲は 200-299 です。	200-299	文字列
producerPoolEnabled (producer)	プロデューサープールが有効かどうか。重要： これをオフにした場合には、プロデューサーに単一の共有接続が使用されます。また、リクエスト/リプライを行う場合でも使用します。つまり、応答が順不同でなければ、インターリーブレスポンスで問題が発生する可能性があることを意味します。そのため、リクエストとリプライメッセージの両方に相関 ID が必要になります。これにより、Camel でメッセージの処理を継続する Camel コールバックに返信を適切に関連付けることができます。そのためには、NettyCamelStateCorrelationManager を相関マネージャーとして実装し、correlationManager オプションを使用して設定する必要があります。詳細は、correlationManager オプションも参照してください。	true	boolean
producerPoolMaxActive (producer)	特定の時間にプールによって割り当て可能なオブジェクト数の上限を設定します（クライアントにチェックするか、待機しているオブジェクト数）。制限なしに負の値を使用してください。	-1	int
producerPoolMaxIdle (producer)	プール内のアイドルインスタンスの数の上限を設定します。	100	int
producerPoolMinEvictable Idle (producer)	アイドルオブジェクトによりエビクションの対象となる前にオブジェクトがプール内にアイドル状態である可能性がある最小時間（ミリ秒単位）を設定します。	300000	Long
producerPoolMinIdle (producer)	エビクトスレッド（アクティブな場合）が新規オブジェクトを生成する前に、プロデューサープールで許可されるインスタンスの最小数を設定します。		int
useRelativePath (producer)	HTTP リクエストで相対パスを使用するかどうかを設定します。	false	boolean

Name	説明	デフォルト	Type
allowSerializedHeaders (advanced)	transferExchange が true の場合にのみ TCP に使用されます。true に設定すると、ヘッダーおよびプロパティーでシリアライズ可能なオブジェクトがエクスチェンジに追加されます。それ以外の場合は、Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。	false	boolean
bootstrapConfiguration (advanced)	このエンドポイントを設定するには、カスタムに設定された NettyServerBootstrapConfiguration を使用します。		NettyServerBootstrap Configuration
channelGroup (advanced)	明示的な ChannelGroup を使用します。		ChannelGroup
Configuration (advanced)	このエンドポイントを設定するには、カスタムに設定された NettyHttpConfiguration を使用します。		NettyHttpConfiguration
disableStreamCache (advanced)	Netty HttpRequestgetContent（）または HttpResponsetgetContent（）からの raw 入力ストリームがキャッシュされているかどうかを決定します（Camel はストリームを軽量メモリーベースのストリームキャッシュに読み取ります）。デフォルトでは、Camel は Netty 入力ストリームをキャッシュして、複数回ロードし、Camel がストリームからすべてのデータを取得できるようにします。ただし、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。たとえば、ファイルまたは他の永続ストアに直接ストリーミングする場合などに、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。このオプションを有効にすると、Netty ストリームが追加設定なしで複数回読み取ることができないため、Netty raw ストリームでリーダーインデックスを手動でリセットする必要がある点に留意してください。また、Netty HTTP サーバー/HTTP クライアントが処理を行うと Netty は Netty ストリームを自動的に閉じます。つまり、非同期ルーティングエンジンが使用されている場合、org.apache.camel.Exchange のルーティングを継続する非同期スレッドは Netty ストリームを読み取りできなくなる可能性があります。これは、Netty が閉じられているため、Netty ストリームの読み取りができません。	false	boolean
headerFilterStrategy (advanced)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用してヘッダーをフィルターします。		HeaderFilterStrategy

Name	説明	デフォルト	Type
nativeTransport (advanced)	NIO の代わりにネイティブトランスポートを使用するかどうか。ネイティブトランスポートはホストのオペレーティングシステムを活用し、一部のプラットフォームでのみサポートされます。使用しているホストオペレーティングシステムの netty JAR を追加する必要があります。詳細は、http://netty.io/wiki/native-transports.htmlを参照してください。	false	boolean
nettyHttpBinding (advanced)	Netty および Camel Message API への/からのバインディングにカスタムの org.apache.camel.component.netty4.http.NettyHttpBinding を使用するには、以下を行います。		NettyHttpBinding
オプション （詳細）	オプション. を接頭辞として使用して追加の netty オプションを設定できます。たとえば、options.child.keepAlive=false を指定して netty オプションの child.keepAlive=false を設定します。使用できるオプションについては、Netty のドキュメントを参照してください。		マップ
receiveBufferSize (advanced)	受信接続中に使用される TCP/UDP バッファーサイズ。サイズはバイト単位です。	65536	int
receiveBufferSizePredictor (advanced)	バッファーサイズの予測を設定します。詳細は、Jetty のドキュメント およびこのメールスレッド を参照してください。		int
sendBufferSize (advanced)	アウトバウンド通信中に使用される TCP/UDP バッファーサイズ。サイズはバイト単位です。	65536	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
transferException (advanced)	有効で Exchange がコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object のコンテンツタイプとして応答でシリアライズされたかどうか。プロデューサー側では、例外は HttpOperationFailedException ではなくデシリアライズされ、そのままスローされます。原因となる例外はシリアライズされている必要があります。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean

Name	説明	デフォルト	Type
transferExchange (advanced)	TCP にのみ使用されます。ボディーだけでなく、ネットワーク上でエクスチェンジを転送できます。次のフィールドが転送されます。本文、Out ボディー、フォールトボディー、ヘッダー、送信ヘッダー、エクスチェンジプロパティー、エクスチェンジ例外。これには、オブジェクトがシリアライズ可能である必要があります。Camel はシリアル化不可能なオブジェクトをすべて除外し、WARN レベルでログに記録されます。	false	boolean
workerCount (advanced)	Netty が nio モードで機能する場合、Netty からのデフォルトの workerCount パラメーターを使用します。これは cpu_core_threads2 です。ユーザーはこの操作を使用して Netty からデフォルトの workerCount をオーバーライドできます。		int
workerGroup (advanced)	明示的な EventLoopGroup を boss スレッドプールとして使用する。たとえば、複数のコンシューマーまたはプロデューサーとスレッドプールを共有する場合などです。デフォルトでは、各コンシューマーまたはプロデューサーには 2 x cpu の数コアスレッドを持つ独自のワーカープールがあります。		EventLoopGroup
decoder (codec)	単一のデコーダーを使用するには **非推奨** になりました。このオプションは非推奨となっています。代わりにエンコーダーを使用してください。		ChannelHandler
decoders (codec)	使用されるデコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値をプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。		文字列
encoder (codec)	単一の **エンコーダーを使用するには非推奨** になりました。このオプションは非推奨となっています。代わりにエンコーダーを使用してください。		ChannelHandler
encoders (codec)	使用されるエンコーダーの一覧。コンマで区切られた値がある String を使用し、値をレジストリーで検索できます。値をプレフィックスに付けることを忘れないようにしてください。そのため、Camel はルックアップを行う必要があります。		文字列
enabledProtocols (security)	SSL を使用する際に有効にするプロトコル	TLSv1, TLSv1.1, TLSv1.2	文字列

Name	説明	デフォルト	Type
keyStoreFile (security)	暗号化に使用されるクライアント側の証明書キーストア		ファイル
keyStoreFormat (security)	ペイロードの暗号化に使用するキーストア形式。設定されていない場合、デフォルトで JKS に設定されます。		文字列
keyStoreResource (security)	暗号化に使用されるクライアント側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		文字列
needClientAuth (security)	SSL の使用時にサーバーがクライアント認証を必要とするかどうかを設定します。	false	boolean
passphrase （セキュリティー）	SSH を使用して送信されたペイロードの暗号化/復号化に使用するパスワード設定		文字列
securityConfiguration （セキュリティー）	セキュアな Web リソースを設定するための org.apache.camel.component.netty4.http.NettyHttpSecurityConfiguration を参照します。		NettyHttpSecurityConfiguration
securityOptions (security)	マップからのキー/値のペアを使用した NettyHttpSecurityConfiguration の設定		マップ
securityProvider (security)	ペイロードの暗号化に使用するセキュリティープロバイダー。設定されていない場合、デフォルトは SunX509 に設定されます。		文字列
ssl （セキュリティー）	SSL 暗号化がこのエンドポイントに適用されるかどうかを指定する設定	false	boolean
sslClientCertHeaders (security)	有効にすると、Netty コンシューマーはサブジェクト名、発行者名、シリアル番号、有効な日付範囲などのクライアント証明書に関する情報を持つヘッダーで Camel メッセージを強化します。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters
sslHandler (security)	SSL ハンドラーを返すために使用できるクラスへの参照		SslHandler
trustStoreFile (security)	暗号化に使用されるサーバー側の証明書キーストア		ファイル

Name	説明	デフォルト	Type
trustStoreResource (security)	暗号化に使用されるサーバー側の証明書キーストア。デフォルトではクラスパスからロードされますが、classpath:、file:、または http: をプレフィックスとして指定して、異なるシステムからリソースをロードすることもできます。		文字列

230.3. メッセージヘッダー

以下のヘッダーをプロデューサーで使用することで、HTTP リクエストを制御できます。

Name	タイプ	説明
CamelHttpMethod	**文字列**	GET、POST、TRACE などの HTTP メソッドの使用を制御できます。タイプは **io.netty.handler.codec.http.HttpMethod** インスタンスにもできます。
CamelHttpQuery	**文字列**	エンドポイント設定を上書きする URI クエリーパラメーターを **String** 値として指定できます。& 記号を使用して、複数のパラメーターを分離します。例：**foo=bar&beer=yes**
CamelHttpPath	**文字列**	エンドポイント設定を上書きする URI コンテキストパスおよびクエリーパラメーターを **String** 値として指定できます。これにより、同じリモート http サーバーを呼び出すために同じプロデューサーを再利用できますが、動的なコンテキストパスとクエリーパラメーターを使用することができます。
Content-Type	**文字列**	HTTP ボディーの content-type を設定します。例： **text/plain; charset="UTF-8"**
CamelHttpResponseCode	**int**	HTTP ステータスコードの使用を許可します。デフォルトでは、成功には 200 を使用し、失敗した場合は 500 を使用します。

ルートが Netty4 HTTP エンドポイントから起動すると、以下のヘッダーが meta-data として提供されます。

表の説明は、from("netty4-http:http:0.0.0.0:8080/myapp")...

Name	タイプ	説明
CamelHttpMethod	**文字列**	GET、POST、TRACE などの使用される HTTP メソッド。
CamelHttpUrl	**文字列**	プロトコル、ホスト、ポートなどの URL: **http://0.0.0.0:8080/myapp**
CamelHttpUri	**文字列**	プロトコル、ホストおよびポートなどの URI: **/myapp**
CamelHttpQuery	**文字列**	**foo=bar&beer=yes**などのクエリーパラメーター
CamelHttpRawQuery	**文字列**	**foo=bar&beer=yes** などのクエリーパラメーター。コンシューマーに到達すると（URL のデコード前）、raw フォームに保存されます。
CamelHttpPath	**文字列**	追加のコンテキストパス。この値は、context-path **/myapp** というクライアントの場合は空になります。クライアントが **/myapp/mystuff** を呼び出す場合、このヘッダーの値は **/mystuff** になります。つまり、ルートエンドポイントに設定された context-path の後の値です。
CamelHttpCharacterEncoding	**文字列**	content-type ヘッダーの文字セット。
CamelHttpAuthentication	**文字列**	ユーザーが HTTP **Basic** を使用して認証されている場合は、Basic の値でこのヘッダーが追加されます。
Content-Type	**文字列**	コンテンツタイプ（指定されている場合）。例： **text/plain; charset="UTF-8"**

230.4. NETTY タイプへのアクセス

このコンポーネントは、Exchange でのメッセージ実装として org.apache.camel.component.netty4.http.NettyHttpMessage を使用します。これにより、以下のよ

うに、エンドユーザーは、必要に応じて元の Netty リクエスト/レスポンスインスタンスにアクセスできるようになります。 元の応答はいつでもアクセスできない可能性があることに注意してください。

```
io.netty.handler.codec.http.HttpRequest request =
exchange.getIn(NettyHttpMessage.class).getHttpRequest();
```

230.5. 例

以下のルートでは、ハードコーディングされた「Bye World」メッセージを返す Netty4 HTTP を HTTP サーバーとして使用します。

```
from("netty4-http:http://0.0.0.0:8080/foo")
  .transform().constant("Bye World");
```

そして、Camel を使用してこの HTTP サーバーを以下のように ProducerTemplate を使用して呼び出すことができます。

```
String out = template.requestBody("netty4-http:http://localhost:8080/foo", "Hello World",
String.class);
System.out.println(out);
```

そして、「Bye World」を出力として戻ります。

230.6. NETTY のワイルドカードを一致させる方法

デフォルトでは、Netty4 HTTP は URI に完全に一致します。ただし、Netty にプレフィックスに一致するよう指示することができます。たとえば、以下のようになります。

```
from("netty4-http:http://0.0.0.0:8123/foo").to("mock:foo");
```

Netty4 HTTP 上のルートでは、URI が完全に一致する場合にのみ HTTP が照合されるため、 http://0.0.0.0:8123/foo と入力しても http://0.0.0.0:8123/foo/bar を実行する場合に一致します。

そのため、ワイルドカードの一致を有効にする場合は、以下を実行します。

```
from("netty4-http:http://0.0.0.0:8123/foo?matchOnUriPrefix=true").to("mock:foo");
```

そのため、Netty は foo で始まるエンドポイントに一致します。

任意のエンドポイントに一致させる には、以下を行います。

```
from("netty4-http:http://0.0.0.0:8123?matchOnUriPrefix=true").to("mock:foo");
```

230.7. 同じポートで複数のルートを使用する

同じ CamelContext では、同じポート（例： io.netty.bootstrap.ServerBootstrap インスタンス）を共有する Netty4 HTTP からの複数のルートを持つことができます。これを実行するには、ルートが同じ io.netty.bootstrap.ServerBootstrap インスタンスを共有するため、ルートで複数のブートストラップオプションが同じである必要があります。インスタンスは、最初に作成されたルートからのオプションを使用して設定します。

ルートが同一で設定されるオプションは、org.apache.camel.component.netty4.NettyServerBootstrapConfiguration 設定クラスで定義されたすべてのオプションです。異なるオプションで別のルートを設定している場合、Camel は起動時に例外を発生させ、オプションが同一ではないことを示します。これを軽減するには、すべてのオプションが同一になるようにします。

以下は、同じポートを共有する 2 つのルートを使用した例です。

2 つのルートが同じポートを共有する

```
from("netty4-http:http://0.0.0.0:{{port}}/foo")
  .to("mock:foo")
  .transform().constant("Bye World");

from("netty4-http:http://0.0.0.0:{{port}}/bar")
  .to("mock:bar")
  .transform().constant("Bye Camel");
```

以下は、org.apache.camel.component.netty4.NettyServerBootstrapConfiguration オプションが 1st ルートと同じではない 2 番目のルートの例になります。これにより、起動時に Camel が失敗します。

同じポートを共有する 2 つのルートがありますが、2 番目のルートの設定が正しく設定され、起動時に失敗します。

```
from("netty4-http:http://0.0.0.0:{{port}}/foo")
  .to("mock:foo")
  .transform().constant("Bye World");

// we cannot have a 2nd route on same port with SSL enabled, when the 1st route is NOT
from("netty4-http:http://0.0.0.0:{{port}}/bar?ssl=true")
  .to("mock:bar")
  .transform().constant("Bye Camel");
```

230.7.1. 複数のルートでの同じサーバーブートストラップ設定の再使用

org.apache.camel.component.netty4.NettyServerBootstrapConfiguration タイプの単一のインスタンスで共通のサーバーブートストラップオプションを設定すると、Netty4 HTTP コンシューマーで bootstrapConfiguration オプションを使用して、すべてのコンシューマーで同じオプションを参照および再利用できます。

```
<bean id="nettyHttpBootstrapOptions"
class="org.apache.camel.component.netty4.NettyServerBootstrapConfiguration">
  <property name="backlog" value="200"/>
  <property name="connectionTimeout" value="20000"/>
  <property name="workerCount" value="16"/>
</bean>
```

また、以下のようにこのオプションを参照するルートで

```
<route>
  <from uri="netty4-http:http://0.0.0.0:{{port}}/foo?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>

<route>
  <from uri="netty4-http:http://0.0.0.0:{{port}}/bar?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>

<route>
  <from uri="netty4-http:http://0.0.0.0:{{port}}/beer?
bootstrapConfiguration=#nettyHttpBootstrapOptions"/>
  ...
</route>
```

230.7.2. OSGi コンテナーで複数のバンドルにまたがる複数のルートを持つ同じサーバーブートストラップ設定を再利用

詳細は、「Netty HTTP Server の例」を参照してください。

230.8. HTTP BASIC 認証の使用

Netty HTTP コンシューマーは、使用するセキュリティーレルム名を指定して HTTP Basic 認証をサポートします。以下に例を示します。

```
<route>
  <from uri="netty4-http:http://0.0.0.0:{{port}}/foo?securityConfiguration.realm=karaf"/>
  ...
</route>
```

Basic 認証を有効にするには、レルム名が必須です。デフォルトでは、JAAS ベースのオーセンティケーターが使用されます。これは指定されたレルム名（上記の例のkaraf）を使用し、認証に JAAS レルムと JAAS \{{LoginModule}}s を使用します。

Apache Karaf / ServiceMix のエンドユーザーには、そのまま使える karaf レルムがあります。そのため、上記のサンプルがこれらのコンテナーで追加設定なしで機能しなくなるからです。

230.8.1. Web リソースでの ACL の指定

org.apache.camel.component.netty4.http.SecurityConstraint では、Web リソースで制約を定義できます。また、org.apache.camel.component.netty.http.SecurityConstraintMapping は追加設定なしで提供され、ロールの追加や除外を簡単に定義できます。

たとえば、XML DSL で以下のように制約 Bean を定義します。

```
<bean id="constraint" class="org.apache.camel.component.netty4.http.SecurityConstraintMapping">
  <!-- inclusions defines url -> roles restrictions -->
  <!-- a * should be used for any role accepted (or even no roles) -->
  <property name="inclusions">
    <map>
      <entry key="/*" value="*"/>
      <entry key="/admin/*" value="admin"/>
      <entry key="/guest/*" value="admin,guest"/>
    </map>
  </property>
  <!-- exclusions is used to define public urls, which requires no authentication -->
  <property name="exclusions">
    <set>
      <value>/public/*</value>
    </set>
  </property>
</bean>
```

上記の制約は、以下のように定義されます。

- /* へのアクセスは制限され、すべてのロールが許可されます（また、ユーザーにロールがない場合も）
- /admin/* へのアクセスには admin ロールが必要です。
- /guest/* へのアクセスには admin ロールまたは guest ロールが必要です。
- /public/* へのアクセスは、認証を必要としない除外されるため、ログインせずにパブリックになります。

この制約を使用するには、以下のように Bean ID を参照する必要があります。

```
<route>
  <from uri="netty4-http:http://0.0.0.0:{{port}}/foo?
matchOnUriPrefix=true&amp;securityConfiguration.realm=karaf&amp;securityConfiguration.securityCon
straint=#constraint"/>
  ...
</route>
```

230.9. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- Netty

- *Netty HTTP サーバーの例*
- *Jetty*

第231章 OGNL 言語

Camel バージョン 1.1 で利用可能

Camel では、OGNL を Expression または Xml 設定として使用できます。

OGNL を使用して、メッセージフィルターまたは Recipient List の式として述語を作成できます。

OGNL ドット表記を使用して操作を呼び出すことができます。たとえば、getFamilyName メソッドを持つ POJO が含まれるボディーがある場合は、以下のように構文を作成できます。

```
"request.body.familyName"
  // or
"getRequest().getBody().getFamilyName()"
```

231.1. OGNL オプション

OGNL 言語は、以下に示す 1 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
trim	**true**	**ブール値**	値をトリミングして先頭および末尾の空白と改行を削除するかどうか。

231.2. 変数

変数	型	説明
this	エクスチェンジ	Exchange はルートオブジェクトです。
exchange	エクスチェンジ	Exchange オブジェクト
exception	Throwable	エクスチェンジの例外 (ある場合)

変数	型	説明
exchangeId	文字列	エクスチェンジ ID
fault	メッセージ	Fault メッセージ（ある場合）
request	メッセージ	exchange.in メッセージ
response	メッセージ	exchange.out メッセージ（ある場合）
properties	マップ	エクスチェンジプロパティー
property(name)	オブジェクト	指定の名前のプロパティー
property(name, type)	Type	指定の名前のプロパティー（指定のタイプ）

231.3. サンプル

たとえば、XML の メッセージフィルター 内で OGNL を使用できます。

```
<route>
  <from uri="seda:foo"/>
  <filter>
    <ognl>request.headers.foo == 'bar'</ognl>
    <to uri="seda:bar"/>
  </filter>
</route>
```

Java DSL を使用した例を以下に示します。

```
from("seda:foo").filter().ognl("request.headers.foo == 'bar'").to("seda:bar");
```

231.4. 外部リソースからのスクリプトの読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、「classpath:」、"file:"、または "http: " などのリソースから Camel に読み込むことができます。
これは、「 resource:scheme:location」 構文を使用して行われます。たとえば、実行可能なクラスパスのファイルを参照します。

```
.setHeader("myHeader").ognl("resource:classpath:myognl.txt")
```

231.5. 依存関係

camel ルートで OGNL を使用するには、OGNL 言語を実装する camel-ognl の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ognl</artifactId>
  <version>x.x.x</version>
</dependency>
```

そうでない場合は、OGNLも必要になります。

第232章 OLINGO2 コンポーネント

Camel バージョン 2.14 から利用可能

Olingo2 コンポーネントは Apache Olingo バージョン 2.0 API を使用して OData 2.0 準拠のサービスと対話します。多くの一般的な商用およびエンタープライズベンダーおよび製品は OData プロトコルをサポートします。サポートする製品のサンプルの一覧は、OData の Web サイト を参照してください。

Olingo2 コンポーネントは、読み取りフィード、デルタフィード、エンティティー、シンプルおよび複雑なプロパティー、リンク、数、カスタムおよび OData システムクエリーパラメーターの使用をサポートします。エンティティー、プロパティー、および関連付けリンクの更新をサポートします。また、単一の OData バッチ操作としてクエリーおよび変更リクエストの送信もサポートします。

コンポーネントは、OData サービスコネクションの HTTP コネクションパラメーターおよびヘッダーの設定をサポートします。これにより、ターゲット OData サービスに必要な SSL、OAuth2.0 などの設定が可能になります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-olingo2</artifactId>
  <version>${camel-version}</version>
</dependency>
```

232.1. URI 形式

```
olingo2://endpoint/<resource-path>?[options]
```

232.2. OLINGO2 オプション

Olingo2 コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
設定 （共通）	共有設定の使用		Olingo2Configuration

Name	説明	デフォルト	Type
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Olingo2 エンドポイントは URI 構文を使用します。

```
olingo2:apiName/methodName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

232.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
apiName	**必要な** 操作の種類		Olingo2ApiName
methodName	選択した操作に使用するサブ操作が **必要**		文字列

232.2.2. クエリーパラメーター（14 パラメーター）：

Name	説明	デフォルト	Type
connectTimeout (common)	HTTP 接続作成のタイムアウト（ミリ秒単位）。デフォルトは 30,000（30 秒）です。	30000	int
contentType (common)	Content-Type ヘッダーの値を使用して、JSON または XML メッセージ形式を指定できます。デフォルトは application/json;charset=utf-8 です。	application/json;charset=utf-8	文字列

Name	説明	デフォルト	Type
httpAsyncClientBuilder (common)	より複雑な HTTP クライアント設定用のカスタム HTTP 非同期クライアントビルダー、connectionTimeout、socketTimeout、proxy および sslContext を上書きします。socketTimeout をビルダーに指定する必要があります。指定しないと OData リクエストが無期限にブロックされる可能性があることに注意してください。		HttpAsyncClientBuilder
httpClientBuilder (common)	より複雑な HTTP クライアント設定用のカスタム HTTP クライアントビルダー、connectionTimeout、socketTimeout、proxy および sslContext を上書きします。socketTimeout をビルダーに指定する必要があります。指定しないと OData リクエストが無期限にブロックされる可能性があることに注意してください。		HttpClientBuilder
httpHeaders (common)	すべてのリクエストに挿入するカスタム HTTP ヘッダー。これには OAuth トークンなどが含まれます。		マップ
inBody (common)	エクスチェンジ In Body で渡されるパラメーターの名前を設定します。		文字列
プロキシー （共通）	HTTP プロキシーサーバーの設定		HttpHost
serviceUri (common)	ターゲット OData サービスベース URI（例：http://services.odata.org/OData/OData.svc）		文字列
socketTimeout (common)	HTTP リクエストのタイムアウト（ミリ秒単位）。デフォルトは 30,000（30 秒）です。	30000	int
sslContextParameters (common)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

232.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、エンドポイント名と次のオプションを使用できます。 プロデューサーエンドポイントは、特別なオプション inBody を使用することもできます。そのオプションには、値が Camel Exchange In メッセージに含まれる endpoint オプションの名前が含まれる必要があります。inBody オプションはデフォルトで、そのオプションを使用するエンドポイントの data に設定されます。

232.4. エンドポイントオプション

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は CamelOlingo2.<option> の形式である必要があります。inBody オプションはメッセージヘッダー（例： body = option）が CamelOlingo2.option ヘッダーを上書きすることに注意してください。さらに、クエリーパラメーターを指定することもできます。

Name	タイプ	説明
data	オブジェクト	OData リソースの作成または変更に使用される適切なタイプを持つデータ
keyPredicate	文字列	パラメーター化された OData リソースエンドポイントを作成するためのキー述語。キー述語の値がヘッダーで動的に提供される作成/更新の操作に便利です。
queryParams	java.util.Map<String,String>	OData システムオプションおよびカスタムクエリーオプション。詳細は、「OData 2.0 URI Conventions」を参照してください。

Name	タイプ	説明
resourcePath	文字列	OData リソースパス、キーの述語を含む場合とそうでない場合があります。
endpointHttpHeaders	java.util.Map<String, String>	エンドポイントに送信される動的 HTTP ヘッダー
responseHttpHeaders	java.util.Map<String, String>	エンドポイントからの動的な HTTP 応答ヘッダー

resourcePath オプションは、エンドポイントオプション ?resourcePath=<resource-path> またはヘッダー値 CamelOlingo2.resourcePath として、URI パスの一部として URI に指定することができることに注意してください。OData エンティティーキーの述語は、リソースパスの一部にすることができます。例：* Manufactur ers ('1') 。*' __1' はキー述語であるか、リソースパス Manufacturers および keyPredicate オプション '1' で個別に指定することができます。

エンドポイント	オプション	HTTP メソッド	結果ボディーのタイプ
batch	data, endpointHttpHeaders	マルチパート/中間のバッチリクエストのある POST	java.util.List<org.apache.camel.component.olingo2.api.batch.Olingo2BatchResponse>
create	data, resourcePath, endpointHttpHeaders	POST	他の OData リソース用の org.apache.olingo.odata2.api.ep.entry.ODataEntry（新しいエントリー org.apache.olingo.odata2.api.commons.HttpStatusCodes）
削除	resourcePath, endpointHttpHeaders	DELETE	org.apache.olingo.odata2.api.commons.HttpStatusCodes

エンドポイント	オプション	HTTP メソッド	結果ボディーのタイプ
merge	data, resourcePath, endpointHttpHeaders	MERGE	org.apache.olingo.odata2.api.commons.HttpStatusCodes
patch	data, resourcePath, endpointHttpHeaders	PATCH	org.apache.olingo.odata2.api.commons.HttpStatusCodes
read	queryParams, resourcePath, endpointHttpHeaders	GET	以下で説明されているようにクエリーされる OData リソースに依存します。
更新	data, resourcePath, endpointHttpHeaders	PUT	org.apache.olingo.odata2.api.commons.HttpStatusCodes

232.5. エンドポイント HTTP ヘッダー（2.20 以降）

*コンポーネントレベルの設定プロパティー **httpHeaders** は、静的な HTTP ヘッダー情報を提供します。ただし、システムによっては、エンドポイントとの間で動的なヘッダー情報を渡す必要があります。サンプルのユースケースは、動的なセキュリティートークンを必要とするシステムです。**endpointHttpHeaders** および **responseHttpHeaders** エンドポイントプロパティーはこの機能を提供します。**CamelOlingo2.endpointHttpHeaders** プロパティーでエンドポイントに渡す必要があるヘッダーを設定し、レスポンスヘッダーは **CamelOlingo2.responseHttpHeaders** プロパティーで返されます。どちらのプロパティーも **java.util.Map<String, String>** のタイプです。*

232.6. ODATA リソースタイプマッピング

読み取り エンドポイントとデータ型は、クエリーされる OData リソースや作成、または変更されるものによって異なります。

OData リソースタイプ	resourcePath および keyPredicate からのリソース URI	In または Out ボディーのタイプ
エンティティーデータモデル	$metadata	org.apache.olingo.odata2.api.edm.Edm
サービスドキュメント	/	org.apache.olingo.odata2.api.servicedocument.ServiceDocument
OData feed	<entity-set>	org.apache.olingo.odata2.api.ep.feed.ODataFeed
OData エントリー	<entity-set>(<key-predicate>)	org.apache.olingo.odata2.api.ep.entry.ODataEntry for Out body(response)java.util.Map<String, Object> for In body(request)
簡単なプロパティー	<entity-set>(<key-predicate>)/<simple-property>	Olingo EdmProperty の説明にあるように適切な Java データ型
シンプルなプロパティーの値	<entity-set>(<key-predicate>)/<simple-property>/$value	Olingo EdmProperty の説明にあるように適切な Java データ型

OData リソースタイプ	resourcePath および keyPredicate からのリソース URI	In または Out ボディーのタイプ
複雑なプロパティー	<entity-set>(<key-predicate>)/<complex-property>	java.util.Map<String, Object>
ゼロまたは 1 つの関連付けリンク	<entity-set>(<key-predicate>/$link/<one-to-one-entity-set-property>	要求の応答 java.util.Map<String, Object> の文字列（キープロパティー名と値）
ゼロまたは多数の関連付けリンク	<entity-set>(<key-predicate>/$link/<one-to-many-entity-set-property>	java.util.List<String>（応答 java.util.List<java.util.Map<String, Object>>）（要求のキープロパティー名と値の一覧を含む）
Count	<resource-uri>/$count	java.lang.Long

232.7. コンシューマーエンドポイント

read エンドポイントのみをコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、 consumer. プレフィックスと共に Scheduled Poll Consumer オプション を使用してエンドポイント呼び出しをスケジュールできます。デフォルトでは、配列またはコレクションを返すコンシューマーエンドポイントは要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 度実行されます。この動作は、エンドポイントプロパティー consumer.splitResult=false を設定することで無効にできます。

232.8. メッセージヘッダー

すべての URI オプションは、 CamelOlingo2 プレフィックスを持つプロデューサーエンドポイントのメッセージヘッダーに提供できます。

232.9. メッセージボディー

すべての結果メッセージ本文は、Olingo2Component によって使用される基礎となる Apache Olingo 2.0 API によって提供されるオブジェクトを使用します。プロデューサーエンドポイントは、 inBody エンドポイント URI パラメーターに受信メッセージボディーのオプション名を指定できます。配列またはコレクションを返すエンドポイントでは、consumer.splitResult が false に設定されていない限り、コンシューマーエンドポイントはすべての要素を個別のメッセージにマッピングします。

232.10. ユースケース

以下のルートは、Name プロパティーで指定された Manufacturer フィードから上位 5 エントリーを読み取ります。

```
from("direct:...")
    .setHeader("CamelOlingo2.$top", "5");
    .to("olingo2://read/Manufacturers?orderBy=Name%20asc");
```

以下のルートは、受信 ID ヘッダーのキープロパティー値を使用して Manufacturer エントリーを読み取ります。

```
from("direct:...")
    .setHeader("CamelOlingo2.keyPredicate", header("id"))
    .to("olingo2://read/Manufacturers");
```

以下のルートは、本文メッセージの java.util.Map<String, Object> を使用して Manufacturer エントリーを作成します。

```
from("direct:...")
    .to("olingo2://create/Manufacturers");
```

以下のルートは、manufacturer delta feed を 30 秒ごとにポーリングします。Bean blah は bean paramsBean を更新し、更新された !deltatoken プロパティーを ODataDeltaFeed 結果で返された値で追加します。最初のデルタトークンが不明なため、コンシューマーエンドポイントは ODataFeed 値を最初に生成し、後続のポーリングで ODataDeltaFeed を生成します。

```
from("olingo2://read/Manufacturers?
queryParams=#paramsBean&consumer.timeUnit=SECONDS&consumer.delay=30")
    .to("bean:blah");
```

第233章 OLINGO4 コンポーネント

Camel バージョン 2.19 から利用可能

Olingo4 コンポーネントは、 Apache Olingo バージョン 4.0 API を使用して OData 4.0 に準拠するサービスと対話します。バージョン 4.0 OData は OASIS 標準であり、一般的なオープンソースおよび商用ベンダーの数であり、このプロトコルをサポートします。サポートする製品のサンプルの一覧は、OData の Web サイト を参照してください。

Olingo4 コンポーネントは、カスタムおよび OData システムのクエリーパラメーターを使用したエンティティーセット、エンティティー、シンプルおよび複雑なプロパティー、数、数の読み取りをサポートします。エンティティーとプロパティーの更新をサポートします。また、単一の OData バッチ操作としてクエリーおよび変更リクエストの送信もサポートします。

コンポーネントは、OData サービスコネクションの HTTP コネクションパラメーターおよびヘッダーの設定をサポートします。これにより、ターゲット OData サービスに必要な SSL、OAuth2.0 などの設定が可能になります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-olingo4</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

233.1. URI 形式

```
olingo4://endpoint/<resource-path>?[options]
```

233.2. OLINGO4 オプション

Olingo4 コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
設定 （共通）	共有設定の使用		Olingo4Configuration
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Olingo4 エンドポイントは URI 構文を使用して設定されます。

```
olingo4:apiName/methodName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

233.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
apiName	**必要な** 操作の種類		Olingo4ApiName
methodName	選択した操作に使用するサブ操作が **必要**		文字列

233.2.2. クエリーパラメーター（14 パラメーター）：

Name	説明	デフォルト	Type
connectTimeout (common)	HTTP 接続作成のタイムアウト（ミリ秒単位）。デフォルトは 30,000（30 秒）です。	30000	int
contentType (common)	Content-Type ヘッダーの値を使用して、JSON または XML メッセージ形式を指定できます。デフォルトは application/json;charset=utf-8 です。	application/json;charset=utf-8	文字列

Name	説明	デフォルト	Type
httpAsyncClientBuilder (common)	より複雑な HTTP クライアント設定用のカスタム HTTP 非同期クライアントビルダー、connectionTimeout、socketTimeout、proxy および sslContext を上書きします。socketTimeout をビルダーに指定する必要があります。指定しないと OData リクエストが無期限にブロックされる可能性があることに注意してください。		HttpAsyncClientBuilder
httpClientBuilder (common)	より複雑な HTTP クライアント設定用のカスタム HTTP クライアントビルダー、connectionTimeout、socketTimeout、proxy および sslContext を上書きします。socketTimeout をビルダーに指定する必要があります。指定しないと OData リクエストが無期限にブロックされる可能性があることに注意してください。		HttpClientBuilder
httpHeaders (common)	すべてのリクエストに挿入するカスタム HTTP ヘッダー。これには OAuth トークンなどが含まれます。		マップ
inBody (common)	エクスチェンジ In Body で渡されるパラメーターの名前を設定します。		文字列
プロキシー （共通）	HTTP プロキシーサーバーの設定		HttpHost
serviceUri (common)	ターゲット OData サービスベース URI（例：http://services.odata.org/OData/OData.svc）		文字列
socketTimeout (common)	HTTP リクエストのタイムアウト（ミリ秒単位）。デフォルトは 30,000（30 秒）です。	30000	int
sslContextParameters (common)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

233.3. プロデューサーエンドポイント

プロデューサーエンドポイントは、エンドポイント名と次のオプションを使用できます。 プロデューサーエンドポイントは、特別なオプション inBody を使用することもできます。そのオプションには、値が Camel Exchange In メッセージに含まれる endpoint オプションの名前が含まれる必要があります。inBody オプションはデフォルトで、そのオプションを使用するエンドポイントの data に設定されます。

エンドポイントオプションは、エンドポイント URI またはメッセージヘッダーで動的に指定できます。メッセージヘッダー名は CamelOlingo4.<option> の形式である必要があります。inBody オプションはメッセージヘッダー（例： body = option）が CamelOlingo4.option ヘッダーを上書きすることに注意してください。さらに、クエリーパラメーターも指定できます。

resourcePath オプションは、エンドポイントオプション ?resourcePath=<resource-path> またはヘッダー値 CamelOlingo4.resourcePath として、URI パスの一部として URI に指定することができることに注意してください。OData エンティティーキーの述語は、リソースパスの一部にすることができます。例： **Manufactur ers ('1') 。'__1' はキー述語であるか、リソースパス Manufacturers および keyPredicate オプション '1' で個別に指定することができます。**

エンドポイント	オプション	HTTP メソッド	結果ボディーのタイプ

エンドポイント	オプション	HTTP メソッド	結果ボディーのタイプ
batch	data, endpointHttpHeaders	マルチパート/中間のバッチリクエストのある POST	java.util.List<org.apache.camel.component.olingo4.api.batch.Olingo4BatchResponse>
create	data, resourcePath, endpointHttpHeaders	POST	他の OData リソース用の org.apache.olingo.client.api.domain.ClientEntity（新しいエントリー：org.apache.olingo.commons.api.http.HttpStatusCode）
削除	resourcePath, endpointHttpHeaders	DELETE	org.apache.olingo.commons.api.http.HttpStatusCode
merge	data, resourcePath, endpointHttpHeaders	MERGE	org.apache.olingo.commons.api.http.HttpStatusCode
patch	data, resourcePath, endpointHttpHeaders	PATCH	org.apache.olingo.commons.api.http.HttpStatusCode

エンドポイント	オプション	HTTP メソッド	結果ボディーのタイプ
read	queryParams, resourcePath, endpointHttpHeaders	GET	以下で説明されているようにクエリーされる OData リソースに依存します。
更新	data, resourcePath, endpointHttpHeaders	PUT	org.apache.olingo.commons.api.http.HttpStatusCode

233.4. エンドポイント HTTP ヘッダー（CAMEL 2.20以降）

*コンポーネントレベルの設定プロパティー **httpHeaders** は、静的な HTTP ヘッダー情報を提供します。ただし、システムによっては、エンドポイントとの間で動的なヘッダー情報を渡す必要があります。サンプルのユースケースは、動的なセキュリティートークンを必要とするシステムです。**endpointHttpHeaders** および **responseHttpHeaders** エンドポイントプロパティーはこの機能を提供します。**CamelOlingo4.endpointHttpHeaders** プロパティーでエンドポイントに渡す必要があるヘッダーを設定し、レスポンスヘッダーは **CamelOlingo4.responseHttpHeaders** プロパティーで返されます。どちらのプロパティーも **java.util.Map<String, String>** のタイプです。*

233.5. ODATA リソースタイプマッピング

*読み取り エンドポイントとデータ型は、クエリーされる **OData** リソースや作成、または変更されるものによって異なります。*

OData リソースタイプ	resourcePath および keyPredicate からのリソース URI	In または Out ボディーのタイプ
エンティティーデータモデル	$metadata	org.apache.olingo.commons.api.edm.Edm
サービスドキュメント	/	org.apache.olingo.client.api.domain.ClientServiceDocument
OData エンティティーセット	<entity-set>	org.apache.olingo.client.api.domain.ClientEntitySet
OData エンティティー	<entity-set>(<key-predicate>)	org.apache.olingo.client.api.domain.ClientEntity for Out body(response)java.util.Map<String, Object> for In body(request)
簡単なプロパティー	<entity-set>(<key-predicate>)/<simple-property>	org.apache.olingo.client.api.domain.ClientPrimitiveValue
シンプルなプロパティーの値	<entity-set>(<key-predicate>)/<simple-property>/$value	org.apache.olingo.client.api.domain.ClientPrimitiveValue

OData リソースタイプ	resourcePath および keyPredicate からのリソース URI	In または Out ボディーのタイプ
複雑なプロパティー	<entity-set>(<key-predicate>)/<complex-property>	org.apache.olingo.client.api.domain.ClientComplexValue
Count	<resource-uri>/$count	java.lang.Long

233.6. コンシューマーエンドポイント

read エンドポイントのみをコンシューマーエンドポイントとして使用できます。コンシューマーエンドポイントは、 consumer. プレフィックスと共に Scheduled Poll Consumer オプション を使用してエンドポイント呼び出しをスケジュールできます。デフォルトでは、配列またはコレクションを返すコンシューマーエンドポイントは要素ごとに 1 つのエクスチェンジを生成し、それらのルートはエクスチェンジごとに 1 度実行されます。この動作は、エンドポイントプロパティー consumer.splitResult=false を設定することで無効にできます。

233.7. メッセージヘッダー

すべての URI オプションは、 CamelOlingo4 プレフィックスを持つプロデューサーエンドポイントのメッセージヘッダーに提供できます。

233.8. メッセージボディー

すべての結果メッセージ本文は、Olingo4Component によって使用される基礎となる Apache Olingo 4.0 API によって提供されるオブジェクトを使用します。プロデューサーエンドポイントは、 inBody エンドポイント URI パラメーターに受信メッセージボディーのオプション名を指定できます。配列またはコレクションを返すエンドポイントでは、consumer.splitResult が false に設定されていない限り、コンシューマーエンドポイントはすべての要素を個別のメッセージにマッピングします。

233.9. ユースケース

以下のルートは、FirstName プロパティーとして順序付けされた People エンティティーから上位 5 エントリーを読み取ります。

```
from("direct:...")
    .setHeader("CamelOlingo4.$top", "5");
    .to("olingo4://read/People?orderBy=FirstName%20asc");
```

以下のルートは、受信 id ヘッダーのキープロパティーの値を使用して、CtrlPorts エンティティーを読み取ります。

```
from("direct:...")
    .setHeader("CamelOlingo4.keyPredicate", header("id"))
    .to("olingo4://read/Airports");
```

以下のルートは、本文メッセージの ClientEntity を使用して People エンティティーを作成します。

```
from("direct:...")
    .to("olingo4://create/People");
```

第234章 OPENSHIFT コンポーネント（非推奨）

Camel バージョン 2.14 から利用可能

openshift コンポーネントは、OpenShift アプリケーションを管理するためのコンポーネントです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openshift</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

234.1. URI 形式

```
openshift:clientId[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

234.2. オプション

OpenShift コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
ユーザー名（セキュリティー）	openshift サーバーにログインするユーザー名です。		文字列
パスワード（セキュリティー）	openshift サーバーにログインするためのパスワード。		文字列
ドメイン (common)	ドメイン名。指定のない場合は、デフォルトのドメインが使用されます。		文字列

Name	説明	デフォルト	Type
server (common)	openshift サーバーの URL。指定されていない場合は、ローカルの openshift 設定ファイル /.openshift/express.conf のデフォルト値が使用されます。これも失敗すると、openshift.redhat.com が使用されます。		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

OpenShift エンドポイントは URI 構文を使用して設定します。

```
openshift:clientId
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

234.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
clientId	***必須***: クライアント ID		文字列

234.2.2. クエリーパラメーター（26 パラメーター）：

Name	説明	デフォルト	Type
ドメイン (common)	ドメイン名。指定のない場合は、デフォルトのドメインが使用されます。		文字列
パスワード (common)	openshift サーバーにログインするためのパスワードが ***必要です***。		文字列
server (common)	openshift サーバーの URL。指定されていない場合は、ローカルの openshift 設定ファイル /.openshift/express.conf のデフォルト値が使用されます。これも失敗すると、openshift.redhat.com が使用されます。		文字列

Name	説明	デフォルト	Type
username (common)	openshift サーバーにログインするためのユーザー名が **必要です**。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
アプリケーション （プロデューサー）	起動、停止、再起動、または状態を取得するアプリケーション名。		文字列
モード （プロデューサー）	メッセージボディーを pojo または json として出力するかどうか。pojo の場合、メッセージは List のタイプです。		文字列

Name	説明	デフォルト	Type
操作 （プロデューサー）	実行する操作（list、start、stop、restart、および state）。list 操作は、json 形式のすべてのアプリケーションに関する情報を返します。state 操作は、started、stoped などの状態を返します。他の操作はどの値も返しません。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service

Name	説明	デフォルト	Type
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

234.3. 例

234.3.1. すべてのアプリケーションの一覧表示

```
// sending route
from("direct:apps")
    .to("openshift:myClient?username=foo&password=secret&operation=list");
    .to("log:apps");
```

この場合、すべてのアプリケーションの情報は pojo として返されます。json 応答が必要な場合は、mode=json を設定します。

234.3.2. アプリケーションの停止

```
// stopping the foobar application
from("direct:control")
    .to("openshift:myClient?
username=foo&password=secret&operation=stop&application=foobar");
```

上記の例では、foobar という名前のアプリケーションを停止します。

ギアの状態変更のポーリング

コンシューマーは、ギアで状態変更をポーリングするために使用されます。たとえば、新しいギアの追加/削除/ライフサイクルの変更、例の開始、停止などです。

```
// trigger when state changes on our gears
from("openshift:myClient?username=foo&password=secret&delay=30s")
    .log("Event ${header.CamelOpenShiftEventType} on application ${body.name} changed state to ${header.CamelOpenShiftEventNewState}");
```

コンシューマーが Exchange を出力すると、ボディーには com.openshift.client.IApplication がメッセージのボディーとして含まれます。以下のヘッダーが含まれます。

ヘッダー	null を指定可能	説明
CamelOpenShiftEventType	非対応	イベントのタイプ。追加、削除、または変更が可能です。
CamelOpenShiftEventOldState	対応	イベントタイプが変更される場合の古い状態。
CamelOpenShiftEventNewState	非対応	イベントタイプの新しい状態

234.4. 関連項目

- *Configuring Camel (Camel の設定)*

- *コンポーネント*
- *エンドポイント*
- *はじめに*

第235章 OPENSHIFT ビルド設定コンポーネント

Camel バージョン 2.17 から利用可能

OpenShift Build Config コンポーネントは Kubernetes コンポーネントの 1 つで、 kubernetes のビルド設定操作を実行するプロデューサーを提供します。

235.1. コンポーネントオプション

Openshift Build Config コンポーネントにはオプションがありません。

235.2. エンドポイントオプション

Openshift Build Config エンドポイントは URI 構文を使用して設定します。

```
openshift-build-configs:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

235.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

235.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	使用する Kubernetes API バージョン		文字列
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		文字列
kubernetesClient (producer)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient

Name	説明	デフォルト	Type
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値

Name	説明	デフォルト	Type
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

第236章 OPENSHIFT ビルドコンポーネント

Camel バージョン 2.17 から利用可能

Kubernetes ビルドコンポーネント は、kubernetes ビルド操作を実行するプロデューサーを提供する Kubernetes コンポーネントの 1 つです。

236.1. コンポーネントオプション

Openshift ビルドコンポーネントにはオプションがありません。

236.2. エンドポイントオプション

Openshift ビルドのエンドポイントは、URI 構文を使用して設定します。

```
openshift-builds:masterUrl
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

236.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
masterUrl	**必要な** Kubernetes マスターの URL		文字列

236.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	使用する Kubernetes API バージョン		文字列
dnsDomain (producer)	ServiceCall EIP に使用される dns ドメイン		文字列
kubernetesClient (producer)	指定されている場合に使用するデフォルトの KubernetesClient		KubernetesClient

Name	説明	デフォルト	Type
操作 （プロデューサー）	Kubernetes で実行するプロデューサーの操作		文字列
portName (producer)	ServiceCall EIP に使用されるポート名		文字列
connectionTimeout (advanced)	Kubernetes API サーバーへの要求を行う際に使用する接続タイムアウト（ミリ秒単位）。		整数
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
caCertData (security)	CA 証明書データ		文字列
caCertFile (security)	CA 証明書ファイル		文字列
clientCertData (security)	クライアント証明書データ		文字列
clientCertFile (security)	クライアント証明書ファイル		文字列
clientKeyAlgo (security)	クライアントによって使用されるキーアルゴリズム		文字列
clientKeyData (security)	クライアントキーデータ		文字列
clientKeyFile (security)	クライアントキーファイル		文字列
clientKeyPassphrase (security)	クライアントキーのパスフレーズ		文字列
oauthToken (security)	認証トークン		文字列
パスワード （セキュリティー）	Kubernetes に接続するためのパスワード		文字列
trustCerts (security)	使用する証明書が信頼できるかどうかを定義します。		ブール値

Name	説明	デフォルト	Type
ユーザー名 （セキュリティー）	Kubernetes に接続するためのユーザー名		文字列

236.3. OPENSTACK コンポーネント

Camel 2.19 から利用可能

openstack コンポーネントは、OpenStack アプリケーションを管理するためのコンポーネントです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

OpenStack サービス	Camel コンポーネント	説明
OpenStack Cinder	openstack-cinder	OpenStack cinder を管理するコンポーネント
OpenStack Glance	openstack-glance	OpenStack glance を管理するコンポーネント
OpenStack Keystone	openstack-keystone	OpenStack keystone を管理するコンポーネント
OpenStack Neutron	openstack-neutron	OpenStack neutron を維持するコンポーネント
OpenStack Nova	openstack-nova	OpenStack nova を管理するコンポーネント
OpenStack Swift	openstack-swift	OpenStack swift を維持するコンポーネント

第237章 OPENSTACK CINDER コンポーネント

Camel バージョン 2.19 から利用可能

openstack-cinder コンポーネントにより、メッセージを OpenStack ブロックストレージサービスに送信できます。

237.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョンに置き換える必要があります。

237.2. URI 形式

```
openstack-cinder://hosturl[?options]
```

URI にクエリーオプションを追加するには、以下の形式で ?options=value&option2=value&...

237.3. URI オプション

OpenStack Cinder コンポーネントにはオプションがありません。

OpenStack Cinder エンドポイントは URI 構文を使用して設定します。

```
openstack-cinder:host
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

237.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
host	**必要な** OpenStack ホストの URL		文字列

237.3.2. クエリーパラメーター（9 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	OpenStack API バージョン	V3	文字列
config (producer)	OpenStack の設定		コンフィグ
ドメイン （プロデューサー）	認証ドメイン	default	文字列
操作 （プロデューサー）	実行する操作		文字列
password (producer)	**必要な** OpenStack パスワード		文字列
プロジェクト （プロデューサー）	**必要な** プロジェクト ID		文字列
サブシステム （プロデューサー）	**必要な** OpenStack Cinder サブシステム		文字列
username (producer)	**必要な** OpenStack ユーザー名		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

237.4. 用途

サブシステムごとに以下の設定を使用できます。

237.5. VOLUMES

237.5.1. ボリュームプロデューサーで実行できる操作

操作	説明
create	新規ボリュームを作成します。
get	ボリュームを取得します。
getAll	すべてのボリュームを取得します。
getAllTypes	ボリューム種別を取得します。
更新	ボリュームを更新します。
削除	ボリュームを削除します。

237.5.2. ボリュームプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	ボリュームの ID。
name	**文字列**	ボリューム名。
description	**文字列**	ボリュームの説明。
size	**整数**	ボリュームのサイズ。
volumeType	**文字列**	ボリュームタイプ。
imageRef	**文字列**	イメージの ID。
snapshotId	**文字列**	スナップショットの ID。
isBootable	**ブール値**	起動可能です。

より正確なボリューム設定が必要な場合は、org.openstack4j.model.storage.block.Volume タイプの新規オブジェクトを作成し、メッセージボディーに送信できます。

237.6. SNAPSHOTS

237.6.1. スナップショットプロデューサーで実行できる操作

操作	説明
create	新規スナップショットを作成します。
get	スナップショットを取得します。
getAll	すべてのスナップショットを取得します。
更新	スナップショットを更新します。
削除	スナップショットを削除します。

237.6.2. スナップショットプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	サーバーの ID。
name	**文字列**	サーバー名。
description	**文字列**	スナップショットの説明。
VolumeId	**文字列**	ボリューム ID。
force	**ブール値**	force.

より詳細なサーバー設定が必要な場合は、org.openstack4j.model.storage.block.VolumeSnapshot タイプの新規オブジェクトを作成して、メッセージボディーに送信できます。

237.7. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*
- *OpenStack コンポーネント*

第238章 OPENSTACK GLANCE COMPONENT

Camel バージョン 2.19 から利用可能

openstack-glance コンポーネントにより、メッセージを OpenStack イメージサービスに送信できます。

238.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-openstack</artifactId>
    <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョンに置き換える必要があります。

238.2. URI 形式

```
openstack-glance://hosturl[?options]
```

URI にクエリーオプションを追加するには、以下の形式で ?options=value&option2=value&...

238.3. URI オプション

OpenStack Glance コンポーネントにはオプションがありません。

OpenStack Glance エンドポイントは、URI 構文を使用して設定します。

```
openstack-glance:host
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

238.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
host	**必要な** OpenStack ホストの URL		文字列

238.3.2. クエリーパラメーター（8 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	OpenStack API バージョン	V3	文字列
config (producer)	OpenStack の設定		コンフィグ
ドメイン （プロデューサー）	認証ドメイン	default	文字列
操作 （プロデューサー）	実行する操作		文字列
password (producer)	**必要な** OpenStack パスワード		文字列
プロジェクト （プロデューサー）	**必要な** プロジェクト ID		文字列
username (producer)	**必要な** OpenStack ユーザー名		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

238.4. 用途

操作	説明
reserve	イメージを予約します。
create	新規イメージを作成します。

操作	説明
更新	イメージを更新する。
upload	イメージをアップロードします。
get	イメージを取得します。
getAll	すべてのイメージを取得します。
削除	イメージを削除します。

238.4.1. Glance プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	フレーバーの ID。
name	**文字列**	フレーバー名。
diskFormat	**org.openstack4j.model.image.DiskFormat**	フレーバー VCPU の数。
containerFormat	**org.openstack4j.model.image.ContainerFormat**	RAM のサイズ。
owner	**文字列**	イメージの所有者。
isPublic	**ブール値**	公開中です。

ヘッダー	Type	説明
minRam	**Long**	最小 ram。
minDisk	**Long**	最小ディスク
size	**Long**	サイズ。
checksum	**文字列**	checksum。
properties	**マップ**	イメージの属性。

238.5. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*
- *OpenStack コンポーネント*

第239章 OPENSTACK KEYSTONE COMPONENT

Camel バージョン 2.19 から利用可能

openstack-keystone コンポーネントにより、メッセージを OpenStack Identity サービスに送信できます。

openstack-keystone コンポーネントは、Identity API v3 のみをサポートしています。

239.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-openstack</artifactId>
    <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョンに置き換える必要があります。

239.2. URI 形式

```
openstack-keystone://hosturl[?options]
```

URI にクエリーオプションを追加するには、以下の形式で ?options=value&option2=value&…

239.3. URI オプション

OpenStack Keystone コンポーネントにはオプションがありません。

OpenStack Keystone のエンドポイントは、URI 構文を使用して設定します。

```
openstack-keystone:host
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

239.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
host	**必要な** OpenStack ホストの URL		文字列

239.3.2. クエリーパラメーター（8 パラメーター）：

Name	説明	デフォルト	Type
config (producer)	OpenStack の設定		コンフィグ
ドメイン （プロデューサー）	認証ドメイン	default	文字列
操作 （プロデューサー）	実行する操作		文字列
password (producer)	**必要な** OpenStack パスワード		文字列
プロジェクト （プロデューサー）	**必要な** プロジェクト ID		文字列
サブシステム （プロデューサー）	**必要な** OpenStack Keystone サブシステム		文字列
username (producer)	**必要な** OpenStack ユーザー名		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

239.4. 用途

サブシステムごとに以下の設定を使用できます。

239.5. ドメイン

239.5.1. ドメインプロデューサーで実行できる操作

操作	説明
create	新規ドメインを作成します。
get	ドメインを取得します。
getAll	すべてのドメインを取得します。
更新	ドメインを更新します。
削除	ドメインを削除します。

239.5.2. ドメインプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	ドメインの ID。
name	**文字列**	ドメイン名。
description	**文字列**	ドメインの説明。

より正確なドメイン設定が必要な場合は、org.openstack4j.model.identity.v3.Domain タイプの新しいオブジェクトを作成して、メッセージボディーに送信できます。

239.6. GROUPS

239.6.1. グループプロデューサーで実行できる操作

操作	説明
create	新規グループを作成します。
get	グループを取得します。

操作	説明
getAll	すべてのグループを取得します。
更新	グループを更新します。
削除	グループを削除します。
addUserToGroup	ユーザーをグループに追加します。
checkUserGroup	グループのユーザーであるかどうかを確認します。
removeUserFromGroup	グループからユーザーを削除します。

239.6.2. グループプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
groupId	**文字列**	グループの ID。
name	**文字列**	グループ名。
userId	**文字列**	ユーザーの ID。
domainId	**文字列**	ドメインの ID。
description	**文字列**	グループの説明。

より正確なグループ設定が必要な場合は、org.openstack4j.model.identity.v3.Group タイプの新規オブジェクトを作成して、メッセージボディーに送信できます。

239.7. PROJECTS

239.7.1. プロジェクトプロデューサーで実行できる操作

操作	説明
create	新規プロジェクトを作成します。
get	プロジェクトを取得します。
getAll	すべてのプロジェクトを取得します。
更新	プロジェクトを更新します。
削除	プロジェクトを削除します。

239.7.2. プロジェクトプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	プロジェクトの ID。
name	**文字列**	プロジェクト名。
description	**文字列**	プロジェクトの説明。
domainId	**文字列**	ドメインの ID。
parentId	**文字列**	親プロジェクト ID。

より正確なプロジェクト設定が必要な場合は、org.openstack4j.model.identity.v3.Project タイプの新規オブジェクトを作成し、メッセージボディーに送信できます。

239.8. リージョン

239.8.1. Region プロデューサーで実行できる操作

操作	説明
create	新規リージョンを作成します。

操作	説明
get	リージョンを取得します。
getAll	すべてのリージョンを取得します。
更新	リージョンを更新します。
削除	リージョンを削除します。

239.8.2. Region プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	リージョン ID。
description	**文字列**	リージョンの説明。

より正確なリージョン設定が必要な場合は、org.openstack4j.model.identity.v3.Region タイプの新規オブジェクトを作成し、メッセージボディーに送信できます。

239.9. USERS

239.9.1. ユーザープロデューサーで実行できる操作

操作	説明
create	新規ユーザーを作成します。
get	ユーザーを取得します。
getAll	すべてのユーザーを取得します。
更新	ユーザーを更新します。
削除	ユーザーを削除します。

239.9.2. ユーザープロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	ユーザーの ID。
name	**文字列**	ユーザー名。
description	**文字列**	ユーザーの説明。
domainId	**文字列**	ドメインの ID。
password	**文字列**	ユーザーのパスワード。
email	**文字列**	ユーザーのメール。

より正確なユーザー設定が必要な場合は、org.openstack4j.model.identity.v3.User タイプの新しいオブジェクトを作成して、メッセージボディーに送信できます。

239.10. 関連項目

- ***Configuring Camel (Camel の設定)***
- ***コンポーネント***
- ***エンドポイント***
- ***はじめに***
- ***OpenStack コンポーネント***

第240章 OPENSTACK NEUTRON コンポーネント

Camel バージョン 2.19 から利用可能

openstack-neutron コンポーネントにより、メッセージを OpenStack ネットワークサービスに送信できます。

240.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョンに置き換える必要があります。

240.2. URI 形式

```
openstack-neutron://hosturl[?options]
```

URI にクエリーオプションを追加するには、以下の形式で ?options=value&option2=value&...

240.3. URI オプション

OpenStack Neutron コンポーネントにはオプションがありません。

OpenStack Neutron のエンドポイントは、URI 構文を使用して設定します。

```
openstack-neutron:host
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

240.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
host	**必要な** OpenStack ホストの URL		文字列

240.3.2. クエリーパラメーター（9 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	OpenStack API バージョン	V3	文字列
config (producer)	OpenStack の設定		コンフィグ
ドメイン （プロデューサー）	認証ドメイン	default	文字列
操作 （プロデューサー）	実行する操作		文字列
password (producer)	**必要な** OpenStack パスワード		文字列
プロジェクト （プロデューサー）	**必要な** プロジェクト ID		文字列
サブシステム （プロデューサー）	**必要な** OpenStack Neutron サブシステム		文字列
username (producer)	**必要な** OpenStack ユーザー名		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

240.4. 用途

サブシステムごとに以下の設定を使用できます。

240.5. NETWORKS

240.5.1. ネットワークプロデューサーで実行できる操作

操作	説明
create	新規ネットワークを作成します。
get	ネットワークを取得します。
getAll	すべてのネットワークを取得します。
削除	ネットワークを削除します。

240.5.2. ネットワークプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	ネットワークの ID。
name	**文字列**	ネットワーク名。
tenantId	**文字列**	テナント ID。
adminStateUp	**ブール値**	AdminStateUp ヘッダー。
networkType	**org.openstack4j.model.network.NetworkType**	ネットワークタイプ。
physicalNetwork	**文字列**	物理ネットワーク
segmentId	**文字列**	セグメント ID。

ヘッダー	Type	説明
isShared	**ブール値**	共有されている。
isRouterExternal	**ブール値**	ルーター外部である。

より詳細なネットワーク設定が必要な場合は、org.openstack4j.model.network.Network タイプの新規オブジェクトを作成して、メッセージのボディーに送信できます。

240.6. SUBNETS

240.6.1. サブネットプロデューサーで実行できる操作

操作	説明
create	新規サブネットを作成します。
get	サブネットを取得します。
getAll	すべてのサブネットを取得します。
削除	サブネットを削除します。
action	サブネットでアクションを実行します。

240.6.2. サブネットプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	サブネットの ID。
name	**文字列**	サブネット名。
networkId	**文字列**	ネットワーク ID。

ヘッダー	Type	説明
enableDHCP	**ブール値**	DHCP を有効にします。
gateway	**文字列**	ゲートウェイ。

サブネット設定をさらに正確にする必要がある場合は、org.openstack4j.model.network.Subnet タイプの新規オブジェクトを作成して、メッセージのボディーに送信できます。

240.7. ポート

240.7.1. ポートプロデューサーで実行できる操作

操作	説明
create	新しいポートを作成します。
get	ポートを取得します。
getAll	すべてのポートを取得します。
更新	ポートを更新します。
削除	ポートを削除します。

240.7.2. ポートプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
name	**文字列**	ポート名。
networkId	**文字列**	ネットワーク ID。

ヘッダー	Type	説明
tenantId	**文字列**	テナント ID。
deviceId	**文字列**	デバイス ID。
macAddress	**文字列**	MAC アドレス。

240.8. ルーター

240.8.1. ルータープロデューサーで実行できる操作

操作	説明
create	新規ルーターを作成します。
get	ルーターを取得します。
getAll	すべてのルーターを取得します。
更新	ルーターを更新します。
削除	ルーターを削除します。
attachInterface	インターフェースを割り当てます。
detachInterface	インターフェースの割り当てを解除します。

240.8.2. ポートプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
name	**文字列**	ルーター名。
routerId	**文字列**	ルーター ID。

ヘッダー	Type	説明
subnetId	**文字列**	サブネット ID。
portId	**文字列**	ポート ID。
interfaceType	**org.openstack4j.model.network.AttachInterfaceType**	インターフェースタイプ。
tenantId	**文字列**	テナント ID。

240.9. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*
- *OpenStack コンポーネント*

第241章 OPENSTACK NOVA コンポーネント

Camel バージョン 2.19 から利用可能

openstack-nova コンポーネントにより、メッセージを OpenStack Compute サービスに送信できます。

241.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョンに置き換える必要があります。

241.2. URI 形式

```
openstack-nova://hosturl[?options]
```

URI にクエリーオプションを追加するには、以下の形式で ?options=value&option2=value&...

241.3. URI オプション

OpenStack Nova コンポーネントにはオプションがありません。

OpenStack Nova エンドポイントは、URI 構文を使用して設定します。

```
openstack-nova:host
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

241.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
host	**必要な** OpenStack ホストの URL		文字列

241.3.2. クエリーパラメーター（9 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	OpenStack API バージョン	V3	文字列
config (producer)	OpenStack の設定		コンフィグ
ドメイン （プロデューサー）	認証ドメイン	default	文字列
操作 （プロデューサー）	実行する操作		文字列
password (producer)	**必要な** OpenStack パスワード		文字列
プロジェクト （プロデューサー）	**必要な** プロジェクト ID		文字列
サブシステム （プロデューサー）	**必要な** OpenStack Nova サブシステム		文字列
username (producer)	**必要な** OpenStack ユーザー名		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

241.4. 用途

サブシステムごとに以下の設定を使用できます。

241.5. フレーバー

241.5.1. フレーバープロデューサーで実行できる操作

操作	説明
create	新規フレーバーを作成します。
get	フレーバーを取得します。
getAll	すべてのフレーバーを取得します。
削除	フレーバーを削除します。

241.5.2. フレーバープロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	フレーバーの ID。
name	**文字列**	フレーバー名。
VCPU	**整数**	フレーバー VCPU の数。
ram	**整数**	RAM のサイズ。
disk	**整数**	ディスクのサイズ。
swap	**整数**	swap のサイズ。
rxtxFactor	**整数**	Rxtx Factor。

より正確なフレーバー設定が必要な場合は、org.openstack4j.model.compute.Flavor タイプの新規オブジェクトを作成して、メッセージのボディーに送信することができます。

241.6. サーバー

241.6.1. サーバープロデューサーで実行できる操作

操作	説明
create	新しいサーバーを作成します。
createSnapshot	サーバーのスナップショットを作成します。
get	サーバーを取得します。
getAll	すべてのサーバーを取得します。
削除	サーバーを削除します。
action	サーバーでアクションを実行します。

241.6.2. Server プロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
ID	**文字列**	サーバーの ID。
name	**文字列**	サーバー名。
ImageId	**文字列**	イメージ ID。
FlavorId	**文字列**	使用されるフレーバーの ID。
KeypairName	**文字列**	キーペアの名前。
NetworkId	**文字列**	ネットワーク ID。
AdminPassword	**文字列**	新規サーバーの管理者パスワード。

ヘッダー	Type	説明
action	**org.openstack4j.model.compute.Action**	実行するアクション。

より詳細なサーバー設定が必要な場合は、org.openstack4j.model.compute.ServerCreate タイプの新しいオブジェクトを作成して、メッセージボディーに送信します。

241.7. キーペア

241.7.1. キーペアプロデューサーで実行できる操作

操作	説明
create	新しいキーペアを作成します。
get	キーペアを取得します。
getAll	すべてのキーペアを取得します。
削除	キーペアを削除します。

241.7.2. キーペアプロデューサーが評価するメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
name	**文字列**	キーペア名。

241.8. 関連項目

- *Configuring Camel (Camel の設定)*

- *コンポーネント*
- *エンドポイント*
- *はじめに*
- *OpenStack コンポーネント*

第242章 OPENSTACK SWIFT コンポーネント

Camel バージョン 2.19 から利用可能

openstack-swift コンポーネントを使用すると、メッセージを OpenStack オブジェクトストレージサービスに送信できます。

242.1. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openstack</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョンに置き換える必要があります。

242.2. URI 形式

```
openstack-swift://hosturl[?options]
```

URI にクエリーオプションを追加するには、以下の形式で ?options=value&option2=value&…

242.3. URI オプション

OpenStack Swift コンポーネントにはオプションがありません。

OpenStack Swift エンドポイントは、URI 構文を使用して設定します。

```
openstack-swift:host
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

242.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
host	**必要な** OpenStack ホストの URL		文字列

242.3.2. クエリーパラメーター（9 パラメーター）：

Name	説明	デフォルト	Type
apiVersion (producer)	OpenStack API バージョン	V3	文字列
config (producer)	OpenStack の設定		コンフィグ
ドメイン （プロデューサー）	認証ドメイン	default	文字列
操作 （プロデューサー）	実行する操作		文字列
password (producer)	**必要な** OpenStack パスワード		文字列
プロジェクト （プロデューサー）	**必要な** プロジェクト ID		文字列
サブシステム （プロデューサー）	**必要な** OpenStack Swift サブシステム		文字列
username (producer)	**必要な** OpenStack ユーザー名		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

242.4. 用途

サブシステムごとに以下の設定を使用できます。

242.5. CONTAINERS

242.5.1. コンテナープロデューサーで実行できる操作

操作	説明
create	新規コンテナーを作成します。
get	コンテナーを取得します。
getAll	すべてのコンテナーを取得します。
更新	コンテナーを更新します。
削除	コンテナーを削除します。
getMetadata	メタデータを取得します。
createUpdateMetadata	メタデータを作成/更新します
deleteMetadata	メタデータを削除します。

242.5.2. ボリュームプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。
name	**文字列**	コンテナー名。
X-Container-Meta-	**マップ**	コンテナーメタデータのプレフィックス。
X-Versions-Location	**文字列**	バージョンの場所。
X-Container-Read	**文字列**	ACL - 読み込まれたコンテナー。
X-Container-Write	**文字列**	ACL - コンテナーの書き込み
limit	**整数**	オプションの表示 - 制限。
marker	**文字列**	List options: marker

ヘッダー	Type	説明
end_marker	**文字列**	List options: 終了マーカー。
delimiter	**Character**	List options: delimiter
path	**文字列**	list options - path.

より正確なコンテナー設定が必要な場合は、コンテナーの一覧を表示し、メッセージボディーに送信するために org.openstack4j.model.storage.object.options.CreateUpdateContainerOptions （作成または更新操作の場合）または org.openstack4j.model.storage.object.options.ContainerListOptions の新しいオブジェクトを作成できます。

242.6. OBJECTS

242.6.1. オブジェクトプロデューサーで実行できる操作

操作	説明
create	新規オブジェクトを作成します。
get	オブジェクトを取得します。
getAll	すべてのオブジェクトを取得します。
更新	オブジェクトを更新します。
削除	オブジェクトを削除します。
getMetadata	メタデータを取得します。
createUpdateMetadata	メタデータを作成/更新します

242.6.2. オブジェクトプロデューサーによって評価されるメッセージヘッダー

ヘッダー	Type	説明
operation	**文字列**	実行する操作。

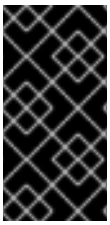
ヘッダー	Type	説明
containerName	**文字列**	コンテナー名。
objectName	**文字列**	オブジェクト名。

242.7. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*
- *OpenStack コンポーネント*

第243章 OPENTRACING コンポーネント

Camel 2.19 から利用可能



重要

Camel 2.21 以降では、OpenTracing Java API バージョン 0.31 以降と互換性のある OpenTracing コンplaint トレーサーを使用する必要があります。

camel-opentracing コンポーネントは、OpenTracing を使用した Camel メッセージのトレースとタイミングに使用されます。

イベント（スパン）は、Camel に送信される受信および送信メッセージに対してキャプチャーされます。

サポートされるトレーサーの一覧は、OpenTracing の Web サイトを参照してください。

243.1. 設定

OpenTracing トレーサーの設定プロパティーは次のとおりです。

オプション	デフォルト	説明
excludePatterns		パターンに一致する Camel メッセージのトレースを無効にする除外パターンを設定します。コンテンツは Set<String> で、キーはパターンです。このパターンは Intercept のルールを使用します。

Camel アプリケーションに分散トレーシングを提供するように OpenTracing トレーサーを設定する方法は 3 つあります。

243.1.1. explicit

選択した OpenTracing 準拠のトレーサーに関連する特定の依存関係と共に、camel-opentracing コンポーネントを POM に含めます。

OpenTracing サポートを明示的に設定するには、OpenTracing トレーサー をインスタンス化し、Camel コンテキストを初期化します。オプションで トレーサー を指定したり、レジストリー または ServiceLoader を使用して暗黙的に検出することもできます。

```
OpenTracingTracer ottracer = new OpenTracingTracer();
// By default it uses a Noop Tracer, but you can override it with a specific OpenTracing
implementation.
ottracer.setTracer(...);
// And then initialize the context
ottracer.init(camelContext);
```

XML で OpenTracing トレーサーを使用するには、OpenTracing トレーサー Bean を定義するだけです。Camel は自動的に検出して使用します。

```
<bean id="tracer" class="..."/>
<bean id="ottracer" class="org.apache.camel.opentracing.OpenTracingTracer">
  <property name="tracer" ref="tracer"/>
</bean>
```

243.1.2. Spring Boot

Spring Boot を使用している場合は、 camel-opentracing-starter 依存関係を追加し、@CamelOpenTracing でメインクラスにアノテーションを付けて OpenTracing をオンにできます。

トレーサー は、トレーサー Bean がアプリケーションによって定義されていない限り、Camel コンテキストの レジストリー または ServiceLoader から暗黙的に取得されます。

243.1.3. Java エージェント

3 つ目のアプローチは、Java エージェントを使用して OpenTracing サポートを自動的に設定する方法です。

選択した OpenTracing 準拠のトレーサーに関連する特定の依存関係と共に、camel-opentracing コンポーネントを POM に含めます。

OpenTracing Java エージェントは、以下の依存関係に関連付けられます。

```
<dependency>
  <groupId>io.opentracing.contrib</groupId>
```

```
    <artifactId>opentracing-agent</artifactId>
  </dependency>
```

使用される トレーサー は、Camel コンテキスト レジストリー または ServiceLoader から暗黙的に読み込まれます。

このエージェントがどのように使用されるかは、アプリケーションの実行方法によって異なります。camel-example-opentracing の Service2 はエージェントをローカルフォルダーにダウンロードし、exec-maven-plugin を使用して -javaagent コマンドラインオプションを指定してサービスを起動します。

243.2. 例

3つの方法で OpenTracing を設定して、camel-example-opentracingの例を示します。

第244章 OPTAPLANNER コンポーネント

Camel バージョン 2.13 から利用可能

optaplanner: コンポーネントは、OptaPlanner のメッセージに含まれるプランニングの問題を解決します。
たとえば、解決しない場合は、未解決の Vehicle Routing の問題を解決します。

コンポーネントは BestSolutionChangedEvent リスナーとしてコンシューマーをサポートし、ソリューションおよび ProblemFactChange を処理するためのプロデューサーをサポートします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-optaplanner</artifactId>
  <version>x.x.x</version><!-- use the same version as your Camel core version -->
</dependency>
```

244.1. URI 形式

```
optaplanner:solverConfig[?options]
```

solverConfig は SolverConfig のクラスパスローカル URI です（例：/org/foo/barSolverConfig.xml）。

URI にクエリーオプションを追加するには、?option=value&option=value&…

244.2. OPTAPLANNER オプション

OptaPlanner コンポーネントにはオプションがありません。

OptaPlanner エンドポイントは、URI 構文を使用して設定します。

```
optaplanner:configFile
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

244.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
configFile	**必須**: Solver ファイルへのロケーションを指定します。		文字列

244.2.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
solverId (common)	Solver インスタンスキーのユーザーに solverId を指定します。	DEFAULT_SOLVER	文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
async (producer)	非同期モードで操作を実行するよう指定します。	false	boolean
threadPoolSize (producer)	async が true の場合に使用するスレッドプールサイズを指定します。	10	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

244.3. メッセージヘッダー

Name	デフォルト値	Type	コンテキスト	説明
Camel OptaPlannerSolverId	null	文字列	共有	使用する solverId を指定します。
Camel OptaPlannerIsAsync	PUT	文字列	プロデューサー	現在のスレッドをブロックするのではなく、別のスレッドを使用してソリューションインスタンスを送信するかどうかを指定します。

244.4. メッセージボディー

Camel は IN ボディーの計画問題を取り、OUT ボディーで返します(v 2.16)。IN ボディーオブジェクトは以下のユースケースをサポートします。

- ボディーが Solution のインスタンスである場合、solverId で識別されるソルバーと、同期または非同期的に、ソルバーを使用して解決されます。
- ボディーが ProblemFactChange のインスタンスの場合には、addProblemFactChange がトリガーされます。処理が非同期の場合、till isEveryProblemFactChangeProcessed 待機してから結果が返されます。
- 上記のタイプのいずれにもボディーがない場合、プロデューサーは solverId で識別されるソルバーから最適な結果を返します。

244.5. 終了

SolverConfig で指定される限り、解決には時間が かかります。

```
<solver>
  ...
  <termination>
    <!-- Terminate after 10 seconds, unless it's not feasible by then yet -->
    <terminationCompositionStyle>AND</terminationCompositionStyle>
    <secondsSpentLimit>10</secondsSpentLimit>
    <bestScoreLimit>-1hard/0soft</bestScoreLimit>
```

```
  </termination>
  ...
<solver>
```

244.5.1. サンプル

OptaPlanner の ActiveMQ キューにあるプランニングの問題を解決します。

```
from("activemq:My.Queue").
  .to("optaplanner:/org/foo/barSolverConfig.xml");
```

OptaPlanner を REST サービスとして公開します。

```
from("cxfrs:bean:rsServer?bindingStyle=SimpleConsumer")
  .to("optaplanner:/org/foo/barSolverConfig.xml");
```

244.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第245章 PAHO コンポーネント

Camel バージョン 2.16 から利用可能

Paho コンポーネントは、Eclipse Paho ライブラリーを使用して MQTT メッセージングプロトコルのコネクターを提供します。Paho は最も人気の高い MQTT ライブラリーの 1 つです。そのため、これを Java プロジェクトに統合する場合は、Camel Paho コネクターを使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-paho</artifactId>
  <version>x.y.z</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Paho アーティファクトは Maven Central でホストされないため、Eclipse Paho リポジトリーを POM xml ファイルに追加する必要があります。

```
<repositories>
 <repository>
  <id>eclipse-paho</id>
  <url>https://repo.eclipse.org/content/repositories/paho-releases</url>
  <snapshots>
   <enabled>false</enabled>
  </snapshots>
 </repository>
</repositories>
```

245.1. URI 形式

```
paho:topic[?options]
```

ここで、topic はトピックの名前です。

245.2. オプション

Paho コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
brokerUrl (common)	MQTT ブローカーの URL。		文字列
clientId (common)	MQTT クライアント識別子。		文字列
connectOptions（詳細）	クライアント接続オプション		MqttConnectOptions
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Paho エンドポイントは URI 構文を使用して設定します。

```
paho:topic
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

245.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
topic	トピックに **必要な** 名前		文字列

245.2.2. クエリーパラメーター（14 パラメーター）：

Name	説明	デフォルト	Type
autoReconnect (common)	接続が失われた場合、クライアントはサーバーへの再接続を自動的に試行します。	true	boolean
brokerUrl (common)	MQTT ブローカーの URL。	tcp://localhost:1883	文字列
clientId (common)	MQTT クライアント識別子。		文字列

Name	説明	デフォルト	Type
connectOptions (common)	クライアント接続オプション		MqttConnectOptions
filePersistenceDirectory (common)	ファイルの永続プロバイダーによって使用されるベースディレクトリー。		文字列
パスワード (common)	MQTT ブローカーに対する認証に使用するパスワード		文字列
永続性 （共通）	使用するクライアントの永続性 - メモリーまたはファイル。	MEMORY	PahoPersistence
QoS （共通）	クライアント品質のサービスレベル(0-2)	2	int
保持され ない（共通）	retain オプション	false	boolean
userName (common)	MQTT ブローカーに対する認証に使用するユーザー名		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

245.3. HEADERS

Paho コンポーネントでは、以下のヘッダーが認識されます。

ヘッダー	Java 定数	エンドポイントタイプ	値のタイプ	説明
CamelMqttTopic	PahoConstants.MQTT_TOPIC	コンシューマー	文字列	トピックの名前
CamelPahoOverrideTopic	PahoConstants.CAMEL_PAHO_OVERRIDE_TOPIC	プロデューサー	文字列	エンドポイントに指定されたトピックの代わりに上書きおよび送信するトピックの名前

245.4. デフォルトのペイロードタイプ

デフォルトでは、Camel Paho コンポーネントは MQTT メッセージから抽出されたバイナリーペイロード上で動作します。

```
// Receive payload
byte[] payload = (byte[]) consumerTemplate.receiveBody("paho:topic");

// Send payload
byte[] payload = "message".getBytes();
producerTemplate.sendBody("paho:topic", payload);
```

しかし、Camel のビルドイン タイプの変換 API は、自動データ型変換を実行できます。以下の例では、Camel は自動的にバイナリーペイロードを String （および逆）に変換します。

```
// Receive payload
String payload = consumerTemplate.receiveBody("paho:topic", String.class);

// Send payload
String payload = "message";
producerTemplate.sendBody("paho:topic", payload);
```

245.5. サンプル

たとえば、以下のスニペットは、Camel ルーターと同じホストにインストールされた MQTT ブローカーからメッセージを読み取ります。

```
from("paho:some/queue")
    .to("mock:test");
```

以下のスニペットは、MQTT ブローカーにメッセージを送信します。

```
from("direct:test")
    .to("paho:some/target/queue");
```

たとえば、リモート MQTT ブローカーからメッセージを読み取る方法は次のとおりです。

```
from("paho:some/queue?brokerUrl=tcp://iot.eclipse.org:1883")
    .to("mock:test");
```

また、ここではデフォルトのトピックを上書きし、動的トピックに設定します。

```
from("direct:test")
    .setHeader(PahoConstants.CAMEL_PAHO_OVERRIDE_TOPIC,
simple("${header.customerId}"))
    .to("paho:some/target/queue");
```

第246章 OSGI PAX LOGGING コンポーネント

Camel バージョン 2.6 で利用可能

paxlogging コンポーネントは、OSGi 環境で使用することで、PaxLogging イベントを受信して処理できます。

246.1. 依存関係

Maven ユーザーは以下の依存関係を pom.xmlに追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-paxlogging</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel（2.6.0 以降）の実際のバージョンに置き換える必要があります。

246.2. URI 形式

```
paxlogging:appender[?options]
```

アペンダー は、Pax アペンダーの名前で、PaxLogging サービス設定で設定する必要があります。

246.3. URI オプション

OSGi PAX Logging コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
bundleContext (consumer)	OSGi BundleContext は Camel によって自動的にインジェクトされます。		BundleContext

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

OSGi PAX Logging エンドポイントは、URI 構文を使用して設定します。

```
paxlogging:appender
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

246.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
appender	***必須*** Appender は、Pax アペンダーの名前で、PaxLogging サービス設定で設定する必要があります。		文字列

246.3.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

246.4. メッセージボディー

in メッセージのボディーは受信した PaxLoggingEvent に設定されます。

246.5. 使用例

```
<route>
    <from uri="paxlogging:camel"/>
    <to uri="stream:out"/>
</route>
```

設定:

```
log4j.rootLogger=INFO, out, osgi:VmLogAppender, osgi:camel
```

第247章 PDF コンポーネント

Camel バージョン 2.16 から利用可能

PDF: コンポーネントは PDF ドキュメントからコンテンツを作成、変更、または抽出する機能を提供します。このコンポーネントは、基礎となるライブラリーとして Apache PDFBox を使用して PDF ドキュメントと連携しています。

PDF コンポーネントを使用するには、Maven ユーザーは以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-pdf</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

247.1. URI 形式

PDF コンポーネントはプロデューサーエンドポイントのみをサポートします。

```
pdf:operation[?options]
```

247.2. オプション

PDF コンポーネントにはオプションがありません。

PDF エンドポイントは、URI 構文を使用して設定します。

```
pdf:operation
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

247.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
operation	**必要な** 操作タイプ		PdfOperation

247.2.2. クエリーパラメーター（9 パラメーター）：

Name	説明	デフォルト	Type
フォント （プロデューサー）	フォント	Helvetica	PDFont
fontSize (producer)	フォントのサイズ（ピクセル）	14	float
marginBottom (producer)	ピクセルの余白	20	int
marginLeft (producer)	下余白（ピクセルの余白）	20	int
marginRight (producer)	ピクセルの右余白	40	int
marginTop (producer)	ピクセルの上余白	20	int
pageSize (producer)	ページサイズ	A4	PDRectangle
textProcessingFactory (producer)	使用するテキスト処理。自動フォーマット： テキストは単語でスライスされてから、行に含まれる単語の最大数は pdf ドキュメントに書き込まれます。このストラテジーを使用すると、この行に収まらないすべての単語は、改行後（line-termination 作成ストラテジー用のクラスセット）に移動します。テキストが行終了シンボルでスライスされてから、そのテキストは行と一致しているかどうかに関わらず書き込まれます。	lineTermination	TextProcessingFactory
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

247.3. HEADERS

ヘッダー	説明
pdf-document	**append** 操作に **必須** ヘッダーで、他のすべての操作で無視されます。想定されるタイプは PDDocument です。追加操作に使用される PDF ドキュメントを保存します。
protection-policy	想定されるタイプは https://pdfbox.apache.org/docs/1.8.10/javadocs/org/apache/pdfbox/pdmodel/encryption/ProtectionPolicy.html[ProtectionPolicy] です。指定すると、PDF ドキュメントが暗号化されます。
decryption-material	想定されるタイプは https://pdfbox.apache.org/docs/1.8.10/javadocs/org/apache/pdfbox/pdmodel/encryption/DecryptionMaterial.html[DecryptionMaterial] です。PDF ドキュメントが暗号化されている場合は **必須** ヘッダーです。

247.4. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

- -

第248章 POSTGRESSQL EVENT COMPONENT

Camel バージョン 2.15 から利用可能

これは Apache Camel のコンポーネントで、PostgreSQL 8.3 以降に追加された LISTEN/NOTIFY コマンドに関連する PostgreSQL イベントの生成/消費を可能にします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-pgevent</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

URI 形式

pgevent コンポーネントは、以下の 2 つのスタイルのエンドポイント URI 表記を使用します。

```
pgevent:datasource[?parameters]
pgevent://host:port/database/channel[?parameters]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

248.1. オプション

PostgresSQL Event コンポーネントにはオプションがありません。

PostgresSQL Event エンドポイントは、URI 構文を使用して設定します。

```
pgevent:host:port/database/channel
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

248.1.1. パスパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
host	ホスト名とポートを使用してデータベースに接続します。	localhost	文字列
port	ホスト名とポートを使用してデータベースに接続します。	5432	整数
database	**必須** データベース名		文字列
channel	**必須**。チャネル名		文字列

248.1.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
データソース (common)	ホスト名およびポートを使用する代わりに、指定の javax.sql.DataSource を使用して接続する場合。		DataSource
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
Pass （セキュリティー）	ログインのパスワード		文字列

Name	説明	デフォルト	Type
ユーザー（セキュリティー）	ログインのユーザー名	postgres	文字列

248.2. 関連項目

- ***Configuring Camel (Camel の設定)***
- ***コンポーネント***
- ***エンドポイント***
- ***はじめに***

第249章 PGP DATAFORMAT

Camel バージョン 2.9 で利用可能

PGP Data Format は、Java Cryptographic Extension を Camel に統合し、Camel の一般的なマーシャルおよびアンマーシャリングフォーマットメカニズムを使用して、メッセージのシンプルで柔軟な暗号化および復号化を可能にします。元の平文に復号化するために、cyphertext および unmarshalling の暗号化を意味するマーシャリングを想定しています。このデータ形式は、対称（共有キー）暗号化とデイジションのみを実装します。

249.1. PGPDATAFORMAT オプション

PGP の dataformat は、以下に示す 15 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明
keyUserid		文字列	暗号化時に使用する PGP キーリングのキーのユーザー ID。また、ユーザー ID の一部にすることもできます。たとえば、ユーザー ID が Test User の場合は、Test User を使用するか、ユーザー ID に対応できます。
signatureKeyUserid		文字列	署名に使用される PGP キーリングのユーザー ID（暗号化中）または署名の検証（復号化中）です。署名の検証プロセス中に、指定のユーザー ID は、検証に使用できるパブリックキーリングから公開鍵を制限します。署名検証にユーザー ID が指定されていない場合、公開キーリング内の公開鍵は検証に使用できます。また、ユーザー ID の一部にすることもできます。たとえば、ユーザー ID が Test User の場合は、Test User を使用するか、User ID に対応できます。
password		文字列	秘密鍵を開く際に使用されるパスワード（暗号化には使用されません）。
signaturePassword		文字列	署名に使用する秘密鍵を開く時に使用するパスワード（暗号化中）。
keyFileName		文字列	キーリングのファイル名です。クラスパスリソースとしてアクセスできる必要があります（ただし、file: プレフィックスを使用してファイルシステムの場所を指定できます）。
signatureKeyFileName		文字列	署名に使用するキーリングのファイル名（暗号化中）または署名の検証（復号化時に）にアクセスできる必要があります。クラスパスリソースとしてアクセスできる必要があります（ただし、file: prefix を使用してファイルシステムの場所を指定できます）。

Name	デフォルト	Java タイプ	説明
signatureKeyRing		文字列	バイトアレイとしての署名/検証に使用されるキーリング。signatureKeyFileName および signatureKeyRing を同時に設定することはできません。
armored	false	ブール値	このオプションを使用すると、PGP は暗号化されたテキストを base64 でエンコードし、コピー/paste などで利用できるようにします。
integrity	true	ブール値	暗号化ファイルに整合性チェック/署名を追加します。デフォルト値は true です。
provider		文字列	Java Cryptography Extension(JCE)プロバイダー。デフォルトは Bouncy Castle(BC)です。または、IAIK JCE プロバイダーなどを使用できます。この場合、プロバイダーは事前に登録され、Bouncy Castle プロバイダーが事前に登録されてはいけません。Sun JCE プロバイダーは機能しません。
algorithm		整数	対称キー暗号化アルゴリズム。可能な値は org.bouncycastle.bcpg.SymmetricKeyAlgorithmTags で定義します（例： 2(= TRIPLE DES)、3(= CAST5)、4(= BLOWFISH)、6(= DES)、7(= AES_128)。暗号化のみに関連します。
compressionAlgorithm		整数	圧縮アルゴリズム。許容値は org.bouncycastle.bcpg.CompressionAlgorithmTags で定義されます。たとえば、0(= UNCOMPRESSED)、1(= ZIP)、2(= ZLIB)、3(= BZIP2)で定義されます。暗号化のみに関連します。
hashAlgorithm		整数	署名ハッシュアルゴリズム。許容値は org.bouncycastle.bcpg.HashAlgorithmTags で定義されます。たとえば、2(= SHA1)、8(= SHA256)、9(= SHA384)、10(= SHA512)、11(=SHA224)で定義されます。署名のみに関連します。

Name	デフォルト	Java タイプ	説明
signatureVerificationOption		文字列	アンマーシャリング時に署名を検証する動作を制御します。4つの値を使用できます：（オプション）: PGP メッセージには署名が含まれる場合もあります。署名が含まれる場合には、署名の検証が行われます。PGP メッセージには少なくとも 1 つの署名が含まれている必要があります。これが例外(PGPException)がスローされない場合は、例外(PGPException)がスローされます。署名の検証が実行されます。ignore: PGP メッセージで取得された署名は無視されます。署名の検証は実行されません。no_signature_allowed: PGP メッセージには署名を含めることができません。それ以外の場合は、例外(PGPException)がスローされます。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

249.2. PGPDATAFORMAT MESSAGE HEADERS

以下のヘッダーをメッセージに動的に適用すると、PGPDataFormat オプションを上書きできます。

Name	タイプ	説明
CamelPGPDataFormatKeyFileName	文字列	**Camel 2.11.0 以降**、キーリングのファイル名は、既存の設定を PGPDataFormat に直接オーバーライドします。
CamelPGPDataFormatEncryptionKeyRing	byte[]	**Camel 2.12.1 以降**、暗号化キーリングは、既存の設定を PGPDataFormat に直接オーバーライドします。
CamelPGPDataFormatKeyUserid	文字列	**Camel 2.11.0 以降**、PGP キーリングのキーのユーザー ID は、PGPDataFormat に直接既存の設定を上書きします。

Name	タイプ	説明
CamelPGPDataFormatKeyUserids	**List<String>**	camel 2.12.2 **以降**: PGP キーリングのキーのユーザー ID。PGPDataFormat に直接既存の設定を上書きします。
CamelPGPDataFormatKeyPassword	**文字列**	Camel 2.11.0 **以降**、秘密鍵を開く際に使用されるパスワード。PGPDataFormat で直接既存の設定を上書きします。
CamelPGPDataFormatSignatureKeyFileName	**文字列**	Camel 2.11.0 **以降**、署名キーリングのファイル名は、既存の設定を PGPDataFormat に直接オーバーライドします。
CamelPGPDataFormatSignatureKeyRing	**byte[]**	Camel 2.12.1 **以降**、署名キーリングは、既存の設定を PGPDataFormat に直接上書きします。
CamelPGPDataFormatSignatureKeyUserid	**文字列**	Camel 2.11.0 **以降**、PGP キーリングの署名キーのユーザー ID は、PGPDataFormat に直接既存の設定を上書きします。
CamelPGPDataFormatSignatureKeyUserids	**List<String>**	Camel 2.12.3 **以降**、PGP キーリングの署名キーのユーザー ID は、PGPDataFormat に直接既存の設定を上書きします。

Name	タイプ	説明
CamelPGPDataFormatSignatureKeyPassword	**文字列**	**Camel 2.11.0 以降**、署名の秘密鍵を開く際に使用されるパスワード。PGPDataFormat で直接既存の設定を上書きします。
CamelPGPDataFormatEncryptionAlgorithm	**int**	**Camel 2.12.2 以降**、対称鍵暗号化アルゴリズムにより、既存の設定が PGPDataFormat に直接上書きされます。
CamelPGPDataFormatSignatureHashAlgorithm	**int**	**Camel 2.12.2 以降**、署名ハッシュアルゴリズム。これにより、既存の設定が PGPDataFormat で直接上書きされます。
CamelPGPDataFormatCompressionAlgorithm	**int**	**Camel 2.12.2 以降**、圧縮アルゴリズムにより、既存の設定が PGPDataFormat で直接上書きされます。
CamelPGPDataFormatNumberOfEncryptionKeys	**整数**	*Camel 2.12.3 **以降** 、シンボリックキーの暗号化に使用する公開鍵の数。暗号化プロセス中に PGPDataFormat により設定されます。

Name	タイプ	説明
CamelPGPDataFormatNumberOfSigningKeys	**整数**	*Camel 2.12.3 **以降** 、署名の作成に使用する秘密鍵の数。署名プロセス中に PGPDataFormat により設定されます。

249.3. PGPDATAFORMAT を使用した暗号化

以下の例では、Bouncy Castle Java ライブラリー を使用してファイルの暗号化/復号化に一般的な PGP 形式を使用します。

以下の例では、署名 + 暗号化を実行し、署名の検証 + 復号化を実行します。署名と暗号化の両方に同じキーリングを使用しますが、明らかに異なるキーを使用することができます。

または、Spring を使用します。

249.3.1. 前の例を使用するには、以下が必要です。

- *データの暗号化に使用される公開鍵が含まれるパブリックキーリングファイル。*
- *データの復号化に使用する鍵が含まれるプライベートキーリングファイル。*
- *キーリングのパスワード*

249.3.2. キーリングの管理

キーリングを管理するには、コマンドラインツールを使用します。鍵を管理する最も簡単な方法であると見なします。その方法が必要な場合は、http://www.bouncycastle.org/java.html から Java ライブラリーを使用することもできます。

linux にコマンドラインユーティリティーをインストールします。

```
apt-get install gnupg
```

キーリングを作成し、セキュアなパスワードを入力します。

```
gpg --gen-key
```

他人の公開鍵をインポートして、ファイルを暗号化できるようにする必要がある場合。

```
gpg --import <filename.key
```

以下のファイルが存在し、サンプルの実行に使用できます。

```
ls -l ~/.gnupg/pubring.gpg ~/.gnupg/secring.gpg
```

[[crypto-PGPDecrypting/VerifyingofMessagesEncrypted/SignedbyDifferentPrivate/PublicKeys]] PGP Decrypting/Verifying of Messages Encrypted/Signed by different # Private/Public Keys

Camel 2.12.2 以降

PGP Data Formater は、異なる公開鍵で暗号化または異なる秘密鍵で署名されたメッセージを復号化および検証できます。唯一の方法として、シークレットキーリング、パブリックキーリングの対応する公開鍵、およびパスフレーズアクセッサーのパスフレーズを指定します。

```
Map<String, String> userId2Passphrase = new HashMap<String, String>(2);
// add passphrases of several private keys whose corresponding public keys have been used to encrypt the messages
userId2Passphrase.put("UserIdOfKey1","passphrase1"); // you must specify the exact User ID!
userId2Passphrase.put("UserIdOfKey2","passphrase2");
PGPPassphraseAccessor passphraseAccessor = new PGPPassphraseAccessorDefault(userId2Passphrase);

PGPDataFormat pgpVerifyAndDecrypt = new PGPDataFormat();
pgpVerifyAndDecrypt.setPassphraseAccessor(passphraseAccessor);
// the method getSecKeyRing() provides the secret keyring as byte array containing the private keys
pgpVerifyAndDecrypt.setEncryptionKeyRing(getSecKeyRing()); // alternatively you can use setKeyFileName(keyfileName)
// the method getPublicKeyRing() provides the public keyring as byte array containing the public keys
pgpVerifyAndDecrypt.setSignatureKeyRing((getPublicKeyRing()); // alternatively you can use
```

```
setSignatureKeyFileName(signatgureKeyfileName)
// it is not necessary to specify the encryption or signer  User Id

from("direct:start")
     ...
     .unmarshal(pgpVerifyAndDecrypt) // can decrypt/verify messages encrypted/signed by
different private/public keys
     ...
```

- この機能は、鍵交換をサポートするのに特に便利です。復号化のために秘密鍵を交換する場合は、古いまたは対応する公開鍵で暗号化される期間メッセージを許可できます。または、送信者が署名側の秘密鍵を交換する必要がある場合は、期間、古い署名者キー、または新しい署名側のキーを受け入れることができます。

- テクニカル背景： PGP 暗号化データには、データの暗号化に使用した公開鍵のキー ID が含まれます。このキー ID は、データを復号化するためにシークレットキーリングで秘密鍵を見つけるために使用できます。署名を検証する公開鍵を見つけるためにも同じメカニズムを使用します。したがって、アンマーシャリングにはユーザー ID を指定する必要はありません。

249.4. PGP 署名の検証時の署名アイデンティティーの制限

Camel 2.12.3 以降。

署名の検証は、署名の正確性を検証するだけでなく、署名が特定のアイデンティティーまたは特定のアイデンティティーのセットから取得されていることも確認する必要がある場合。したがって、署名の検証に使用できる公開鍵の数をパブリックキーリングから制限できます。

署名ユーザー ID

```
// specify the User IDs of the expected signer identities
 List<String> expectedSigUserIds = new ArrayList<String>();
 expectedSigUserIds.add("Trusted company1");
 expectedSigUserIds.add("Trusted company2");

 PGPDataFormat pgpVerifyWithSpecificKeysAndDecrypt = new PGPDataFormat();
 pgpVerifyWithSpecificKeysAndDecrypt.setPassword("my password"); // for decrypting with
private key
 pgpVerifyWithSpecificKeysAndDecrypt.setKeyFileName(keyfileName);
 pgpVerifyWithSpecificKeysAndDecrypt.setSignatureKeyFileName(signatgureKeyfileName);
 pgpVerifyWithSpecificKeysAndDecrypt.setSignatureKeyUserids(expectedSigUserIds); // if
you have only one signer identity then you can also use setSignatureKeyUserid("expected
Signer")

from("direct:start")
```

```
...
.unmarshal(pgpVerifyWithSpecificKeysAndDecrypt)
...
```

- PGP コンテンツに複数の署名がある場合、1 つの署名を検証するとすぐに検証に成功します。

- 検証の署名側のアイデンティティーを制限しない場合は、署名キーのユーザー ID を指定しないでください。この場合、公開鍵内のすべての公開鍵が考慮されます。

249.5. 1 つの PGP データフォーマットでの複数の署名

Camel 2.12.3 以降。

PGP 仕様では、1 つの PGP データフォーマットに、異なるキーの複数の署名を含めることができます。Camel 2.13.3 以降、シークレットキーリング内の複数の秘密鍵に関連する署名ユーザー ID を指定して、このような PGP コンテンツを作成できます。

複数の署名

```
PGPDataFormat pgpSignAndEncryptSeveralSignerKeys = new PGPDataFormat();
pgpSignAndEncryptSeveralSignerKeys.setKeyUserid(keyUserid); // for encrypting, you can also use setKeyUserids if you want to encrypt with several keys
pgpSignAndEncryptSeveralSignerKeys.setKeyFileName(keyfileName);
pgpSignAndEncryptSeveralSignerKeys.setSignatureKeyFileName(signatgureKeyfileName);
pgpSignAndEncryptSeveralSignerKeys.setSignaturePassword("sdude"); // here we assume that all private keys have the same password, if this is not the case then you can use setPassphraseAccessor

List<String> signerUserIds = new ArrayList<String>();
signerUserIds.add("company old key");
signerUserIds.add("company new key");
pgpSignAndEncryptSeveralSignerKeys.setSignatureKeyUserids(signerUserIds);

from("direct:start")
        ...
        .marshal(pgpSignAndEncryptSeveralSignerKeys)
        ...
```

249.6. PGP DATA FORMAT MARSHALER におけるサブキーおよびキーフラグのサポート

*Camel 2.12.3 以降。
* OpenPGP V4 キー にはプライマリーキーとサブキーを持たせることができます。キーの使用はキー

フラグ で表さ れます。たとえば、2 つのサブキーを持つプライマリーキーを設定できます。プライマリーキーは、他のキーを認定するためにのみ使用されます(Key Flag 0x01)、最初のサブキーが署名(Key Flag 0x02)にのみ使用され、2 つ目のサブキーは暗号化にのみ使用されます（Key Flag 0x04 または 0x08）。PGP Data Format マーシャラは、署名と暗号化の適切なキーを決定するために、プライマリーキーとサブキーのこれらのキーフラグを考慮します。プライマリーキーとそのサブキーが同じユーザー ID を持つため、この設定が必要です。

249.7. カスタムキーアクセサーのサポート

*Camel 2.13.0.
以降、暗号化/署名のためにカスタムキーアクセサーを実装できます。上記の PGPDataFormat クラスは、署名/暗号化または検証/復号化に使用する必要があるキーを事前に選択します。キーの選択方法に特別な要件がある場合は、代わりに PGPKeyAccessDataFormat クラスを使用して、インターフェース PGPPublicKeyAccessor および PGPSecretKeyAccessor を Bean として実装する必要があります。プロセッサーが呼び出される際にキーリングが解析されるたびにないように、キーをキャッシュするデフォルトの実装 DefaultPGPPublicKeyAccessor および DefaultPGPSecretKeyAccessor があります。

PGPKeyAccessDataFormat には、password、keyFileName、encryptionKeyRing、SignaturePassword、SignatureKeyFileName、および signatureKeyRing を除く PGPDataFormat と同じオプションがあります。

249.8. 依存関係

camel ルートで PGP データ形式を使用するには、以下の依存関係を pom に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-crypto</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

249.9. 関連項目

- データフォーマット
- crypto(Digital Signatures)
- http://www.bouncycastle.org/java.html

第250章 PROPERTIES コンポーネント

Camel バージョン 2.3 の時点で利用可能

250.1. URI 形式

```
properties:key[?options]
```

key は、検索するプロパティーのキーです。

250.2. オプション

Properties コンポーネントは、以下に示す 17 個のオプションをサポートします。

Name	説明	デフォルト	Type
ロケーション （共通）	プロパティーを読み込む場所の一覧。このオプションはデフォルトのロケーションを上書きし、このオプションのロケーションのみを使用します。		リスト
Location (common)	プロパティーを読み込む場所の一覧。複数のロケーションを分ける場合はコンマを使用できます。このオプションはデフォルトのロケーションを上書きし、このオプションのロケーションのみを使用します。		文字列
encoding (common)	ファイルシステムまたはクラスパスからプロパティーファイルを読み込むときに使用するエンコーディング。エンコーディングが設定されていない場合、プロパティーファイルはリンク java.util.Propertiesload(java.io.InputStream)に記載されている ISO-8859-1 エンコーディング(latin-1)を使用して読み込まれます。		文字列
propertiesResolver (common)	カスタムの PropertiesResolver を使用するには、以下を行います。		PropertiesResolver
propertiesParser (common)	カスタムプロパティーParser の使用		PropertiesParser
キャッシュ （共通）	ロードされたプロパティーをキャッシュするかどうか。デフォルト値は true です。	true	boolean

Name	説明	デフォルト	Type
propertyPrefix (advanced)	解決する前にプロパティー名の前にオプションの接頭辞が追加されます。		文字列
propertySuffix (advanced)	解決する前にプロパティー名に追加される任意の接尾辞。		文字列
fallbackToUnaugmented Property (advanced)	true の場合、最初に propertyPrefix および propertySuffix を使用してプロパティー名の解決を試行し、指定したプレーンプロパティー名をフォールバックします。false の場合、拡張されたプロパティー名のみが検索されます。	true	boolean
defaultFallbackEnabled (common)	false の場合、コンポーネントはコロンセパレーターの後に検索するキーのデフォルトの検索を試行しません。	true	boolean
ignoreMissingLocation (common)	ロケーションが見つからない場合（プロパティーファイルが見つからないなど）をサイレントに無視するかどうか。	false	boolean
prefixToken (advanced)	置き換えるプロパティーを識別するために使用される接頭辞トークンの値を設定します。null の値を設定すると、デフォルトのトークンが復元されます（リンク DEFAULT_PREFIX_TOKEN）。	{{	文字列
suffixToken (advanced)	置き換えるプロパティーを識別するために使用される接尾辞トークンの値を設定します。null の値を設定すると、デフォルトのトークンが復元されます（リンク DEFAULT_SUFFIX_TOKEN）。	}}	文字列
initialProperties (advanced)	ロケーションが解決される前に使用される初期プロパティーを設定します。		プロパティー
overrideProperties (advanced)	プロパティーが存在する場合に、優先され、最初に使用するオーバーライドプロパティーの特殊なリストを設定します。		プロパティー
systemPropertiesMode (common)	システムプロパティーモードを設定します。	2	int
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Properties エンドポイントは、URI 構文を使用して設定します。

```
properties:key
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

250.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
key	プレースホルダーとして使用する **必須** のプロパティーキー		文字列

250.2.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
ignoreMissingLocation (common)	ロケーションが見つからない場合（プロパティーファイルが見つからないなど）をサイレントに無視するかどうか。	false	boolean
ロケーション （共通）	プロパティーを読み込む場所の一覧。複数のロケーションを分ける場合はコンマを使用できます。このオプションはデフォルトのロケーションを上書きし、このオプションのロケーションのみを使用します。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler

Name	説明	デフォルト	Type
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

ヒント

Java コードからプロパティーを解決する。CamelContext 上の resolvePropertyPlaceholders メソッドを使用して、Java コードからプロパティーを解決できます。

250.3. PROPERTYPLACEHOLDER の使用

Camel 2.3 の時点で利用可能

Camel は camel-core で新しい PropertiesComponent を提供するようになり、Camel エンドポイント URI の定義時にプロパティープレースホルダーを使用できるようになりました。

これは、Spring の < property-placeholder > タグを使用している場合と同じように機能します。ただし、Spring には、サードパーティーフレームワークが Spring プロパティープレースホルダーを完全なものに利用することを妨げる制限があります。詳細は「How do I use Spring Property Placeholder with Camel XML」を参照してください。

ヒント

Spring と Camel プロパティープレースホルダーのブリッジから、Spring プロパティープレースホルダーを Camel とブリッジできます。詳細は、以下を参照してください。

通常、プロパティープレースホルダーは以下を使用します。

- *エンドポイントの検索または作成*

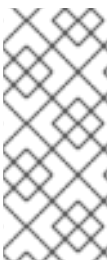
- *レジストリーでの Bean のルックアップ*
- *Spring XML での追加サポート（以下の例を参照）*
- *Camel Properties コンポーネントでの Blueprint PropertyPlaceholder の使用*
- *@PropertyInject を使用した POJO へのプロパティーのインジェクト*
- *プロパティーが存在しない場合のデフォルト値を使用した Camel 2.14.1*
- *Camel 2.14.1 には、追加設定なしの関数が含まれており、OS 環境変数、JVM システムプロパティー、またはサービスのイディオムからプロパティー値を検索します。*
- *Camel 2.14.1: プロパティーコンポーネントにプラグインできるカスタム関数を使用する Camel 2.14.1。*

250.4. 構文

Camel のプロパティープレースホルダーを使用する構文は {{key}}（例： {{ file.uri}}）を使用します。file.uri はプロパティーキーに置き換えます。

エンドポイント URI の部分でプロパティープレースホルダーを使用できます。たとえば、URI のパラメーターにプレースホルダーを使用できます。

Camel 2.14.1 以降では、キーを持つプロパティーが存在しない場合に使用するデフォルト値を指定できます（例： example file.url:/some/path ）。ここでは、デフォルト値はコロンの後のテキスト（例： /some/path）になります。



注記

プロパティーキーにはコロンを使用しないでください。コロンは、Camel 2.14.1 以降でサポートされるデフォルト値を提供する際に区切り文字トークンとして使用されます。

250.5. PROPERTYRESOLVER

Camel はプラグ可能なメカニズムを提供します。これにより、3 番目の部分が独自のリゾルバーでプロパティーを検索できます。Camel は、ファイルシステム、クラスパス、またはレジストリーからプロパティーを読み込めるデフォルトの実装 org.apache.camel.component.properties.DefaultPropertiesResolver を提供します。場所には、以下のいずれかのプレフィックスを付けることができます。

- ref: Camel 2.4: レジストリーでルックアップする
- File: ファイル システムから読み込む
- classpath: クラスパスからロードします（接頭辞が指定されていない場合もデフォルトになります）。
- Blueprint: Camel 2.7: 特定の OSGi Blueprint プレースホルダーサービスを使用する

250.6. 場所の定義

PropertiesResolver は、プロパティーを解決する場所を把握しておく必要があります。1 から多くの場所を定義できます。1 つの String プロパティーで場所を定義した場合は、以下のように複数の場所をコンマで区切ることができます。

```
pc.setLocation("com/mycompany/myprop.properties,com/mycompany/other.properties");
```

Camel 2.19.0 から利用可能

オプション 属性（デフォルトでは false）を設定することで、欠落している場合に破棄できる場所を設定できます。

```
pc.setLocations(
    "com/mycompany/override.properties;optional=true"
    "com/mycompany/defaults.properties");
```

250.7. 場所でのシステムおよび環境変数の使用

Camel 2.7 で利用可能

この場所は、JVM システムプロパティーおよび OS 環境変数のプレースホルダーの使用をサポートするようになりました。

以下に例を示します。

```
location=file:${karaf.home}/etc/foo.properties
```

上記の場所で、キー karaf.home を持つ JVM システムプロパティーを使用して File スキームを使用して場所を定義しました。

代わりに OS 環境変数を使用するには、env をプレフィックスとして付加する必要があります。

```
location=file:${env:APP_HOME}/etc/foo.properties
```

APP_HOME は OS 環境です。

以下のように、同じ場所に複数のプレースホルダーを配置することができます。

```
location=file:${env:APP_HOME}/etc/${prop.name}.properties
```

#=== system and environment variables to configure property prefix and suffix

Camel 2.12.5、2.13.3、2.14.0 から利用可能

propertyPrefix、propertySuffix 設定プロパティーは、JVM システムプロパティーおよび OS 環境変数のプレースホルダーの使用をサポートします。

たとえば、 PropertiesComponent を以下のプロパティーファイルで設定している場合は、以下のようになります。

```
dev.endpoint = result1
test.endpoint = result2
```

次に、以下のルート定義を使用します。

```
PropertiesComponent pc = context.getComponent("properties", PropertiesComponent.class);
pc.setPropertyPrefix("${stage}.");
// ...
context.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start").to("properties:mock:{{endpoint}}");
    }
});
```

システムプロパティー ステージ を dev （メッセージを mock:result1にルーティングされる）または test （メッセージは mock:result2にルーティングされます）に変更すると、ターゲットエンドポイントを変更できます。

250.8. JAVA DSL での設定

以下の例のように、名前 プロパティー の下に PropertiesComponent を作成して登録する必要があります。

```
PropertiesComponent pc = new PropertiesComponent();
pc.setLocation("classpath:com/mycompany/myprop.properties");
context.addComponent("properties", pc);
```

250.9. SPRING XML での設定

Spring XML は 2 種類の設定を提供します。Spring Bean を PropertiesComponent として定義し、Java DSL での動作と類似しています。または、< propertyPlaceholder> タグを使用すること もできます。

```
<bean id="properties" class="org.apache.camel.component.properties.PropertiesComponent">
    <property name="location" value="classpath:com/mycompany/myprop.properties"/>
</bean>
```

< propertyPlaceholder> タグ を使用すると、以下のような設定がより新しいになります。

```
<camelContext ...>
  <propertyPlaceholder id="properties" location="com/mycompany/myprop.properties"/>
</camelContext>
```

location タグでプロパティーの場所を設定すると問題なく機能しますが、Camel 2.19.0 から開始するためのリソースが多数ある場合は、専用の propertiesLocation でプロパティーの場所を設定できます。

```
<camelContext ...>
  <propertyPlaceholder id="myPropertyPlaceholder">
    <propertiesLocation
      resolver = "classpath"
      path     = "com/my/company/something/my-properties-1.properties"
      optional = "false"/>
    <propertiesLocation
      resolver = "classpath"
      path     = "com/my/company/something/my-properties-2.properties"
      optional = "false"/>
    <propertiesLocation
      resolver = "file"
      path     = "${karaf.home}/etc/my-override.properties"
      optional = "true"/>
  </propertyPlaceholder>
</camelContext>
```

ヒント

XML
Camel 2.10 以降では、キャッシュオプションの指定が Spring と Blueprint XML の両方で、キャッシュオプションの値の指定をサポートします。

250.10. レジストリーからのプロパティーの使用

Camel 2.4
で利用可能： たとえば、OSGi の場合、プロパティーを java.util.Properties オブジェクトとして返すサービスを公開する場合があります。

その後、以下のように Properties コンポーネントを設定できます。

```
<propertyPlaceholder id="properties" location="ref:myProperties"/>
```

myProperties は、OSGi レジストリーでの検索に使用する ID に置き換えます。ref: プレフィックスを使用して、レジストリーのプロパティーを検索する必要があることを Camel に伝えます。

250.11. PROPERTIES コンポーネントの使用例

エンドポイント URI でプロパティープレースホルダーを使用する場合は、Properties：コンポーネントを使用するか、URI でプレースホルダーを直接定義できます。前者から始めた両方のケースの例を示しています。

```
// properties
cool.end=mock:result

// route
from("direct:start").to("properties:{{cool.end}}");
```

エンドポイント URI の一部としてプレースホルダーを使用することもできます。

```
// properties
cool.foo=result

// route
from("direct:start").to("properties:mock:{{cool.foo}}");
```

上記の例では、to エンドポイントが mock:result に解決されます。

また、以下のようなプロパティーを参照することもできます。

```
// properties
cool.foo=result
cool.concat=mock:{{cool.foo}}

// route
from("direct:start").to("properties:mock:{{cool.concat}}");
```

Cool .concat が別のプロパティーを参照する方法に注目してください。

properties: コンポーネントは、Location オプションを使用して指定の URI の 場所 を上書きし、指定することもできます。

```
from("direct:start").to("properties:bar.end?locations=com/mycompany/bar.properties");
```

250.12. 例

プロパティーを使用せずにエンドポイント URI で直接プロパティープレースホルダーを使用することも できます。

```
// properties
cool.foo=result

// route
from("direct:start").to("mock:{{cool.foo}}");
```

そして、必要に応じて複数の場所で使用することができます。

```
// properties
cool.start=direct:start
cool.showid=true
cool.result=result

// route
from("{{cool.start}}")
    .to("log:{{cool.start}}?showBodyType=false&showExchangeId={{cool.showid}}")
    .to("mock:{{cool.result}}");
```

たとえば、ProducerTemplate を使用する場合は、プロパティープレースホルダーを使用することもできます。

```
template.sendBody("{{cool.start}}", "Hello World");
```

250.13. SIMPLE 言語の例

Simple 言語は、たとえば以下のルートでプロパティープレースホルダーの使用もサポートします。

```
// properties
cheese.quote=Camel rocks

// route
from("direct:start")
    .transform().simple("Hi ${body} do you think ${properties:cheese.quote}?");
```

Simple 言語で場所を指定することもできます。以下に例を示します。

```
// bar.properties
bar.quote=Beer tastes good
```

```
// route
from("direct:start")
    .transform().simple("Hi ${body}. ${properties:com/mycompany/bar.properties:bar.quote}.");
```

250.14. SPRING XML でサポートされる追加のプロパティープレースホルダー

プロパティープレースホルダーは、<package> , <packageScan>, <contextScan>, <jmxAgent>, <endpoint>, <routeBuilder>, <proxy > など多くの Camel Spring XML タグでもサポートされます。

以下の例では、< jmxAgent > タグのプロパティープレースホルダーがあります。

また、以下に示すように trace など、< camelContext> タグの さまざまな属性にプロパティープレースホルダーを定義することも可能です。

250.15. JVM システムプロパティーを使用したプロパティー設定の上書き

Camel 2.5
で利用可能。変更を取得するためにアプリケーションを再起動しなくても、JVM システムプロパティーを使用して実行時にプロパティー値を上書きできます。これは、コマンドラインから実行することもできます。そのためには、新しい値に置き換えるプロパティーと同じ名前の JVM システムプロパティーを作成します。これの例を以下に示します。

```
PropertiesComponent pc = context.getComponent("properties", PropertiesComponent.class);
pc.setCache(false);

System.setProperty("cool.end", "mock:override");
System.setProperty("cool.result", "override");

context.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start").to("properties:cool.end");
        from("direct:foo").to("properties:mock:{{cool.result}}");
    }
});
context.start();

getMockEndpoint("mock:override").expectedMessageCount(2);

template.sendBody("direct:start", "Hello World");
template.sendBody("direct:foo", "Hello Foo");

System.clearProperty("cool.end");
System.clearProperty("cool.result");

assertMockEndpointsSatisfied();
```

250.16. XML DSL での任意の種類の属性のプロパティープレースホルダーの使用

Camel 2.7 で利用可能

注記

OSGi Blueprint を使用する場合、これは 2.11. 1 または 2 .10.5 以降のみ動作します。

以前のバージョンでは、プレースホルダーをサポートする XML DSL の xs:string タイプ属性のみでした。たとえば、timeout 属性は xs:int タイプであるため、文字列の値をプレースホルダーキーとして設定することはできません。これは、特別なプレースホルダーの namespace を使用して Camel 2.7 以降で実行できるようになりました。

以下の例では、名前空間 http://camel.apache.org/schema/placeholder に prop プレフィックスを使用します。この接頭辞では、XML DSL の属性に prop プレフィックスを使用できます。Multicast で、オプション stopOnException がキー「stop」を持つプレースホルダーの値であることを示す方法に注目してください。

プロパティーファイルには、以下のように定義された値があります。

```
stop=true
```

250.17. CAMEL ルートでの BLUEPRINT プロパティープレースホルダーの使用

Camel 2.7 で利用可能

Camel は Blueprint をサポートし、プロパティープレースホルダーサービスも提供します。Camel は設定の規則をサポートするので、以下のように XML ファイルに OSGi Blueprint プロパティープレースホルダーを定義するだけです。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
         xsi:schemaLocation="
         http://www.osgi.org/xmlns/blueprint/v1.0.0
https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

    <!-- OSGI blueprint property placeholder -->
    <cm:property-placeholder id="myblueprint.placeholder" persistent-id="camel.blueprint">
```

```
        <!-- list some properties as needed -->
        <cm:default-properties>
            <cm:property name="result" value="mock:result"/>
        </cm:default-properties>
    </cm:property-placeholder>

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <!-- in the route we can use {{ }} placeholders which will lookup in blueprint
             as Camel will auto detect the OSGi blueprint property placeholder and use it -->
        <route>
            <from uri="direct:start"/>
            <to uri="mock:foo"/>
            <to uri="{{result}}"/>
        </route>
    </camelContext>
</blueprint>
```

250.17.1. Camel ルートでの OSGi Blueprint プロパティープレースホルダーの使用

デフォルトでは、Camel は OSGi Blueprint プロパティープレースホルダーサービスを検出し、使用します。これを無効にするには、< camelContext > 定義で useBlueprintPropertyResolver 属性を false に設定します。

250.17.2. プレースホルダー構文

Camel ルートのプレースホルダー {{ および }} に Camel 構文 を 使用すると、OSGi Blueprint から値を検索する方法に注目してください。

プレースホルダーの Blueprint 構文は ${ } です。そのため、< camelContext> 以外では ${ } 構文を使用する必要があります。ここで、< camelContext > 内では {{ および }} 構文 を 使用する必要があります。

OSGi Blueprint を使用すると構文の設定が可能になり、必要に応じてそれらを実際に合わせることができます。

また、ID により、特定の OSGi Blueprint プロパティープレースホルダーを明示的に参照することもできます。以下の例のように、Camel の < propertyPlaceholder> を使用する必要があります。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0"
           xsi:schemaLocation="
           http://www.osgi.org/xmlns/blueprint/v1.0.0
https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">
```

```
    <!-- OSGI blueprint property placeholder -->
    <cm:property-placeholder id="myblueprint.placeholder" persistent-id="camel.blueprint">
      <!-- list some properties as needed -->
      <cm:default-properties>
        <cm:property name="prefix.result" value="mock:result"/>
      </cm:default-properties>
    </cm:property-placeholder>

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
      <!-- using Camel properties component and refer to the blueprint property placeholder by
 its id -->
      <propertyPlaceholder id="properties" location="blueprint:myblueprint.placeholder"
                           prefixToken="[[" suffixToken="]]"
                           propertyPrefix="prefix."/>

      <!-- in the route we can use {{ }} placeholders which will lookup in blueprint -->
      <route>
        <from uri="direct:start"/>
        <to uri="mock:foo"/>
        <to uri="[[result]]"/>
      </route>
    </camelContext>
  </blueprint>
```

250.18. CAMEL の OSGI BLUEPRINT プレースホルダーへの明示的な参照

Blueprint スキームを使用して、ID で OSGi Blueprint プレースホルダーを参照する方法に注目してください。これにより、組み合わせて一致させることができます。たとえば、ロケーションに追加スキームを指定することもできます。たとえば、クラスパスからファイルを読み込むには、以下を行います。

```
location="blueprint:myblueprint.placeholder,classpath:myproperties.properties"
```

各場所はカンマで区切られます。

250.19. CAMELCONTEXT 外での BLUEPRINT プロパティープレースホルダーのオーバーライド

Camel 2.10.4 から利用可能

Blueprint XML ファイルで Blueprint プロパティープレースホルダーを使用する場合は、以下のように XML ファイルで直接プロパティーを宣言できます。

プロパティーの 1 つを参照する & lt;bean > があることに留意してください。また、Camel ルートでは {{ および }} 表記を使用して他の ルート を参照します。

ユニットテストからこれらの Blueprint プロパティーをオーバーライドする場合は、以下のようにすることができます。

これを実行するには、useOverridePropertiesWithConfigAdmin メソッドを上書きし、実装します。その後、指定の props パラメーターで上書きしたいプロパティーを配置できます。また、戻り値は、Blueprint XML ファイルで定義する < cm:property-placeholder> タグの persistence-id である必要があります。

250.20. BLUEPRINT プロパティープレースホルダーの .CFG または .PROPERTIES ファイルの使用

Camel 2.10.4 から利用可能

Blueprint XML ファイルで Blueprint プロパティープレースホルダーを使用する場合は、.properties または .cfg ファイルでプロパティーを宣言できます。Apache ServieMix / Karaf を使用する場合、このコンテナーには、命名 etc/pid.cfg で etc ディレクトリーのファイルからプロパティーを読み込む規則があります。pid は persistence-id です。

たとえば、Blueprint XML ファイルでは persistence-id="stuff" があり、これは設定ファイルを etc/stuff.cfg としてロードします。

この Blueprint XML ファイルをユニットテストする場合は、loadConfigAdminConfigurationFile を上書きし、以下のようにどのファイルを読み込むかを Camel に指示できるようになりました。

このメソッドは、2 つの値を持つ String[] を返す必要があることに注意してください。1st 値は、読み込む設定ファイルのパスです。2 番目の値は、< cm:property-placeholder> タグの persistence- id です。

stuff.cfg ファイルは、以下のようなプロパティープレースホルダーを持つ単純なプロパティーファイルです。

```
== this is a comment
greeting=Bye
```

250.21. .CFG ファイルの使用および BLUEPRINT プロパティープレースホルダーのプロパティーの上書き

両方を実行できます。完全な例を以下に示します。まず、Blueprint XML ファイルを使用します。

そして、ユニットテストクラスでは以下のように行います。

etc/stuff.cfg 設定ファイルには以下が含まれます。

```
greeting=Bye
echo=Yay
destination=mock:result
```

250.22. SPRING と CAMEL プロパティープレースホルダーのブリッジ

Camel 2.10 で利用可能

Spring Framework では、Apache Camel などのサードパーティーフレームワークが Spring プロパティープレースホルダーメカニズムにシームレスにフックすることはできません。しかし、Spring と Camel を簡単にブリッジするには、org.apache.camel.spring.spi.BridgePropertyPlaceholderConfigurer タイプである Spring Bean を宣言します。これは Spring org.springframework.beans.factory.config.PropertyPlaceholderConfigurer タイプです。

Spring と Camel をブリッジするには、以下のように単一の Bean を定義する必要があります。

Spring と Camel プロパティープレースホルダーのブリッジ

spring <context:property-placeholder> namespace を同時に使用することはでき ませ ん。これはできません。

この Bean を宣言した後、以下のように Spring スタイルと <camelContext> タグ内の Camel スタイルの両方を使用してプロパティープレースホルダーを定義できます。

ブリッジプロパティープレースホルダーの使用

hello Bean が ${ } 表記を使用して純粋な Spring プロパティープレースホルダーをどのように使用しているかに注目してください。Camel ルートでは、{{ および }} で Camel プレースホルダー表記 を 使

用します。

250.23. SPRING プロパティープレースホルダーの CAMELS SIMPLE 言語との宣言

Spring ブリッジプレースホルダーを使用する場合は spring ${ } 構文が Simple in Camel と競合するため注意してください。以下に例を示します。

```
<setHeader headerName="Exchange.FILE_NAME">
  <simple>{{file.rootdir}}/${in.header.CamelFileName}</simple>
</setHeader>
```

Spring プロパティープレースホルダーと競合します。$simple{ } を使用して Camel で Simple 言語を使用する必要があります。

```
<setHeader headerName="Exchange.FILE_NAME">
  <simple>{{file.rootdir}}/$simple{in.header.CamelFileName}</simple>
</setHeader>
```

別の方法として、ignoreUnresolvablePlaceholders オプションを true に、PropertyPlaceholderConfigurer を設定します。

250.24. CAMEL テストキットからのプロパティーの上書き

Camel 2.10 で利用可能

Camel でテストして Properties コンポーネントを使用する場合、ユニットテストのソースコード内で直接使用するプロパティーを指定したい場合があります。
Camel テストキットとして Camel 2.10 以降では、以下のようなメソッド（たとえば CamelTestSupport クラスが提供）できるようになりました。

- useOverridePropertiesWithPropertiesComponent
- ignoreMissingLocationWithPropertiesComponent

たとえば、ユニットテストクラスでは useOverridePropertiesWithPropertiesComponent メソッドを上書きし、使用するプロパティーが含まれる java.util.Properties を返すことができます。

250.24.1. ユニットテストソース内でのプロパティーを指定

これは、camel-test、camel-test-spring、camel-test-blueprint などの Camel Test Kit のいずれかから実行できます。

ignoreMissingLocationWithPropertiesComponent を使用すると、検出不可能なロケーションを無視するように Camel に指示することができます。たとえば、ユニットテストを実行する場合、プロパティーの場所にアクセスできない環境でユニットテストを実行します。

250.25. USING @PROPERTYINJECT

Camel 2.12 から利用可能

Camel では、フィールドおよびセッターメソッドに設定できる @PropertyInject アノテーションを使用して、POJO にプロパティープレースホルダーを注入することができます。

たとえば、以下のように RouteBuilder クラスで使用できます。

```
public class MyRouteBuilder extends RouteBuilder {

    @PropertyInject("hello")
    private String greeting;

    @Override
    public void configure() throws Exception {
        from("direct:start")
            .transform().constant(greeting)
            .to("{{result}}");
    }

}
```

greeting フィールドに @PropertyInject アノテーションが付けられ、キー "hello" を使用するように定義されていることに注意してください。Camel はこのキーでプロパティーを検索し、その値を String 型にインジェクトします。

キーに複数のプレースホルダーやテキストを使用することもできます。以下に例を示します。

```
@PropertyInject("Hello {{name}} how are you?")
private String greeting;
```

これにより、"name" キーを持つプレースホルダーが検索されます。

キーが存在しない場合には、デフォルト値を追加することもできます。以下に例を示します。

```
@PropertyInject(value = "myTimeout", defaultValue = "5000")
private int timeout;
```

250.26. 初期状態の関数の使用

Camel 2.14.1 から利用可能

Properties コンポーネントには、追加設定なしで以下の機能が含まれます。

- env: OS 環境変数からプロパティーを検索する関数
- sys - Java JVM システムプロパティーからプロパティーを検索する関数
- service - サービス命名規則を使用して OS 環境変数からプロパティーを検索する機能
- service.name - Camel 2.16 .1: ホスト名の部分のみを返すサービス命名規則を使用して、OS 環境変数からプロパティーを検索する関数
- service.port - Camel 2.16 .1: ポートの部分のみを返すサービス命名規則を使用して、OS 環境変数からプロパティーを検索する関数

これらの関数は、環境からの値の検索を容易にすることを目的としています。そのまま使えるように、そのまま使用することができますが、以下のように簡単に使用できます。

```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">

  <route>
    <from uri="direct:start"/>
    <to uri="{{env:SOMENAME}}"/>
```

```
    <to uri="{{sys:MyJvmPropertyName}}"/>
  </route>
</camelContext>
```

デフォルト値を使用することもできます。そのため、プロパティーが存在しない場合はデフォルト値を定義できます。ここでのデフォルト値は **log:foo**、**log:bar** です。

```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">

  <route>
    <from uri="direct:start"/>
    <to uri="{{env:SOMENAME:log:foo}}"/>
    <to uri="{{sys:MyJvmPropertyName:log:bar}}"/>
  </route>
</camelContext>
```

サービス関数は、サービス命名の **idiom** を使用して **OS** 環境変数で定義されるサービスを検索し、**hostname: port**を使用してサービスの場所を参照します。

- **NAME_*SERVICE_HOST***
- **NAME_*SERVICE_PORT***

つまり、サービスはプレフィックスとして **_SERVICE_HOST** および **_SERVICE_PORT** を使用します。サービスの名前が **FOO** である場合は、**OS** 環境変数を以下のように設定する必要があります。

```
export $FOO_SERVICE_HOST=myserver
export $FOO_SERVICE_PORT=8888
```

たとえば、**FOO** サービスがリモート **HTTP** サービスである場合、**Camel** エンドポイント **URI** でサービスを参照し、 ***HTTP*** コンポーネントを使用して **HTTP** 呼び出しを行うことができます。

```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">
  <route>
    <from uri="direct:start"/>
    <to uri="http://{{service:FOO}}/myapp"/>
  </route>
</camelContext>
```

また、サービスが定義されていない場合（ローカルホストでサービスを呼び出すなど）、ユニットテスト用にデフォルト値を使用できます。

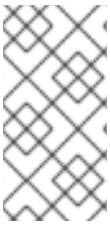
```
<camelContext xmlns="http://camel.apache.org/schema/blueprint">
  <route>
    <from uri="direct:start"/>
    <to uri="http://{`{service:FOO:localhost:8080}`}/myapp"/>
  </route>
</camelContext>
```

250.27. カスタム関数の使用

Camel 2.14.1 から利用可能

Properties コンポーネントを使用すると、プロパティープレースホルダーの解析中に使用できるサードパーティーの機能をプラグインできます。これらの関数は、データベースの検索、カスタム計算の実行、または何ではないかなど、プレースホルダーを解決するためのカスタムロジックを実行できます。関数の名前はプレースホルダーで使用される接頭辞になります。これは、以下のコード例で最も適しています。

```
<bean id="beerFunction" class="MyBeerFunction"/>

<camelContext xmlns="http://camel.apache.org/schema/blueprint">
  <propertyPlaceholder id="properties">
    <propertiesFunction ref="beerFunction"/>
  </propertyPlaceholder>

  <route>
    <from uri="direct:start"/>
    <to uri="{`{beer:FOO}`}"/>
    <to uri="{`{beer:BAR}`}"/>
  </route>
</camelContext>
```



注記

camel 2.19.0 からは、場所属性（propertyPlaceholder タグ）がより必須ではありません。

ここでは、Bean ID を使用するよう &lt ;propertyPlaceholder& gt; を定義した Camel XML ルートがあります（例： beerFunction ）。アクセラー関数は「beer 」 を名前として使用するため、プレースホルダー構文は beer:value で始まるビール機能をトリガーできます。

関数の実装は、以下のように 2 つのメソッドのみになります。

```
public static final class MyBeerFunction implements PropertiesFunction {

    @Override
    public String getName() {
        return "beer";
    }

    @Override
    public String apply(String remainder) {
        return "mock:" + remainder.toLowerCase();
    }
}
```

この関数は org.apache.camel.component.properties.PropertiesFunction インターフェースを実装する必要があります。getName メソッドは関数の名前です（例： beer）。apply メソッドは、カスタムロジックを実装する場所です。サンプルコードはユニットテストからのものであるため、モックエンドポイントを参照する値のみを返します。

Java コードからカスタム関数を登録するには、以下のようにします。

```
PropertiesComponent pc = context.getComponent("properties", PropertiesComponent.class);
pc.addFunction(new MyBeerFunction());
```

250.28. 関連項目

- Properties コンポーネント
- プロパティーで暗号化された値（パスワードなど）を使用する場合の Jasypt

第251章 PROTOBUF DATAFORMAT

Camel バージョン 2.2.0 で利用可能

第252章 PROTOBUF - プロトコルバッファー

"protocol Buffers - Google のデータ交換形式"

Camel は Data Format を提供し、Java と Protocol Buffer プロトコルをシリアライズします。この形式を xml で選択 したい理由をプロジェクトのサイトの詳細。プロトコルバッファーは言語に依存しないプラットフォームであるため、Camel ルートによって生成されたメッセージは他の言語実装によって消費される可能性があります。

API サイト
Protobuf 実装

Protobuf Java Tutorial

252.1. PROTOBUF オプション

Protobuf データフォーマットは、以下に示す 3 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
instanceClass		文字列	無効化解除時に使用するクラスの名前
contentTypeFormat	native	文字列	protobuf メッセージが Java からシリアライズ/デシリアライズされるコンテンツタイプ形式を定義します。形式は、ネイティブ protobuf フィールドまたは json フィールド表現のいずれかの native または json のいずれかになります。デフォルト値は native です。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

252.2. コンテンツタイプの形式（ CAMEL 2.19から開始）

JSON メッセージを解析してprotobuf 形式に変換し、ネイティブユーティリティーコンバーターを使用して解析し直すことができます。このオプションを使用するには、contentTypeFormat の値を 'json' に設定するか、2 番目のパラメーターで protobuf を呼び出します。デフォルトインスタンスが指

定されていない場合は、常にネイティブの protobuf 形式を使用してください。サンプルコードは、以下を示しています。

```
from("direct:marshal")
    .unmarshal()
    .protobuf("org.apache.camel.dataformat.protobuf.generated.AddressBookProtos$Person",
"json")
    .to("mock:reverse");
```

252.3. PROTOBUF の概要

Protobuf の使用方法に関する簡単な概要です。詳細は、完全なチュートリアルを参照してください。

252.4. PROTO 形式の定義

最初のステップでは、エクスチェンジのボディーの形式を定義します。これは .proto ファイルに次のように定義されます。

addressbook.proto

```
syntax = "proto2";

package org.apache.camel.component.protobuf;

option java_package = "org.apache.camel.component.protobuf";
option java_outer_classname = "AddressBookProtos";

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

```
message AddressBook {
  repeated Person person = 1;
}
```

252.5. JAVA クラスの生成

Protobuf SDK は、.proto ファイルで定義した形式の Java クラスを生成するコンパイラーを提供します。オペレーティングシステムが Protobuf Java コードジェネレーター maven プラグイン でサポートされている場合は、以下の設定を pom.xml に追加することで、protobuf Java コードの生成を自動化できます。

プロジェクトの pom.xml の < build > タグまたは ${os.detected.classifier} パラメーター内にオペレーティングシステムおよび CPU アーキテクチャー検出拡張を手動で挿入します。

```
<extensions>
  <extension>
    <groupId>kr.motd.maven</groupId>
    <artifactId>os-maven-plugin</artifactId>
    <version>1.4.1.Final</version>
  </extension>
</extensions>
```

プロジェクトの pom.xml の gRPC および protobuf Java コードジェネレータープラグイン <plugins > タグを挿入します。

```
<plugin>
  <groupId>org.xolstice.maven.plugins</groupId>
  <artifactId>protobuf-maven-plugin</artifactId>
  <version>0.5.0</version>
  <extensions>true</extensions>
  <executions>
    <execution>
      <goals>
        <goal>test-compile</goal>
        <goal>compile</goal>
      </goals>
      <configuration>
        <protocArtifact>com.google.protobuf:protoc:${protobuf-
version}:exe:${os.detected.classifier}</protocArtifact>
      </configuration>
    </execution>
  </executions>
</plugin>
```

手動で必要となる追加の対応言語に対してコンパイラーを実行することもできます。

```
protoc --java_out=. ./proto/addressbook.proto
```

これにより、Person および AddressBook の内部クラスが含まれる AddressBookProtos という名前の単一の Java クラスが生成されます。ビルダーも実装されます。生成されたクラスは com.google.protobuf.Message を実装します。これは、シリアライズメカニズムに必要です。このため、これらのクラスのみがエクスチェンジのボディーで使用されることが重要です。Camel は、com.google.protobuf.Message を実装しないクラスを使用するように Data Format に伝えると、ルートの作成時に例外をスローします。生成されたビルダーを使用して、既存のドメインクラスからデータを変換します。

252.6. JAVA DSL

ProtobufDataFormat インスタンスを作成して、Camel DataFormat のマーシャリングおよびアンマーシャリング API に渡すことができます。

```
ProtobufDataFormat format = new ProtobufDataFormat(Person.getDefaultInstance());

from("direct:in").marshal(format);
from("direct:back").unmarshal(format).to("mock:reverse");
```

または、DSL protobuf（）を使用して、デフォルトのインスタンスまたはデフォルトのインスタンスクラス名をこのように渡します。

```
// You don't need to specify the default instance for protobuf marshaling
from("direct:marshal").marshal().protobuf();
from("direct:unmarshalA").unmarshal()

.protobuf("org.apache.camel.dataformat.protobuf.generated.AddressBookProtos$Person")
      .to("mock:reverse");


from("direct:unmarshalB").unmarshal().protobuf(Person.getDefaultInstance()).to("mock:reverse");
```

252.7. SPRING DSL

以下の例は、protobuf データ型を設定する Protobuf を使用してアンマーシャリングする方法を示しています。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <unmarshal>
```

```
      <protobuf
instanceClass="org.apache.camel.dataformat.protobuf.generated.AddressBookProtos$Perso
n" />
    </unmarshal>
    <to uri="mock:result"/>
  </route>
</camelContext>
```

252.8. 依存関係

camel ルートで Protobuf を使用するには、このデータ形式を実装する camel-protobuf の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-protobuf</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

252.9. 関連項目

- Camel gRPC コンポーネント

第253章 PUBNUB コンポーネント

Camel バージョン 2.19 から利用可能

Camel Pub コンポーネントは、接続されたデバイスの PubNub データストリームネットワークとの通信に使用できます。このコンポーネントは pubnub java ライブラリー を使用します。

含まれるユースケース：

- チャット部屋： メッセージの送受信
- ロケーションと接続カード： フォロッキングカービングのディスパッチ
- スマートセンサー： データビジュアライゼーションのためにセンサーからデータを取得する
- 健全性： 患者のお気に入りのデバイスからの中核的なレートの監視
- マルチプレプレイスのゲーミング
- interactive media: audience-participating voting system

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-pubnub</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

253.1. URI 形式

```
pubnub:channel[?options]
```

channel は、公開またはサブスクライブする PubNub チャネルです。

253.2. オプション

PubNub コンポーネントにはオプションがありません。

PubNub エンドポイントは URI 構文を使用します。

```
pubnub:channel
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

253.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
channel	イベントのサブスクライブ/公開に使用するチャネルが **必要です**。		文字列

253.2.2. クエリーパラメーター（14 パラメーター）：

Name	説明	デフォルト	Type
UUID （共通）	デバイス識別子として使用する UUID。渡されていない場合はデフォルトの UUID が生成されます。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
withPresence （コンシューマー）	関連する存在情報にもサブスクライブします。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
操作 （プロデューサー）	実行する操作。PUBLISH: デフォルトチャネルのすべてのサブスクライバーにメッセージを送信します。FIRE: クライアントが BLOCKS イベントハンドラーにメッセージを送信できるようにします。これらのメッセージは、チャネルに登録されているイベントハンドラーに直接表示されます。HERENOW: チャネルに現在サブスクライブしている一意のユーザー ID の一覧と、合計占有数を含むチャンネルの現在の状態に関する情報を取得します。WHERENOW: uuid がサブスクライブしているチャンネル一覧に関する情報を取得します。GETSTATE: サブスクライバー uuid に固有のキーと値のペアを取得するのに使用します。状態情報は、キー/値のペア SETSTATE の JSON オブジェクトとして提供されます。これは、サブスクライバーの uuid GETHISTORY: 取得チャネルの過去のメッセージの取得に固有するキーと値のペアを設定するのに使用します。		文字列
pubnub (advanced)	レジストリーの Pubnub クライアントへの参照。		PubNub
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
authKey (security)	Access Manager を使用する場合、クライアントは制限のあるすべての要求でこの authKey を使用します。		文字列
cipherKey (security)	暗号が渡されると、PubNub への/からのトラフィックはすべて暗号化されます。		文字列
publishKey (security)	PubNub アカウントから取得した公開キー。メッセージをパブリッシュする際に必須です。		文字列

Name	説明	デフォルト	Type
secretKey (security)	メッセージの署名に使用されるシークレットキー。		文字列
セキュア（セキュリティー）	セキュアな送信には SSL を使用します。	true	boolean
subscribeKey (security)	PubNub アカウントから取得したサブスクライブキー。チャネルにサブスクライブしたり、存在イベントをリッスンする場合に必要です。		文字列

253.3. サブスクライブ時のメッセージヘッダー

Name	説明
CamelPubNubTimeToken	イベントのタイムスタンプ。
CamelPubNubChannel	メッセージが属するチャネル。

253.4. メッセージボディー

*メッセージボディーには、**Objects**、**Arrays**、**Ints**、**Strings** などの **JSON** シリアライズ可能なデータを含めることができます。メッセージデータには特別な **Java V4** クラスや関数が含まれないでください。これらはシリアライズされないためです。文字列コンテンツには、任意の単一バイトまたはマルチバイト **UTF-8** を含めることができます。*

*送信時のオブジェクトのシリアライズ。完全なオブジェクトをメッセージペイロードとして渡します。**pubnub** はオブジェクトのシリアライズを処理します。*

*メッセージボディーを受け取ると、**PubNub API** が提供するオブジェクトを使用します。*

253.5. 例

253.5.1. イベントの公開

*生成時のデフォルトの操作。以下のスニペットは、**PosjoBean** によって生成されたイベントをチャネル **iot** に公開します。*

```
from("timer:mytimer")
    // generate some data as POJO.
    .bean(PojoBean.class)
    .to("pubnub:iot?publishKey=mypublishKey");
```

253.5.2. 失敗したイベント aka BLOCKS イベントハンドラー

呼び出し可能なすべてのサーバーレス機能については、https://www.pubnub.com/blocks-catalog/ を参照してください。位置情報ルックアップの例

```
from("timer:geotimer")
    .process(exchange -> exchange.getIn().setBody(new Foo("bar", "TEXT")))
    .to("pubnub:eon-maps-geolocation-input?
operation=fire&publishKey=mypubkey&subscribeKey=mysubkey");

from("pubnub:eon-map-geolocation-output?subscribeKey=mysubkey)
    // geolocation output will be logged here
    .log("${body}");
```

253.5.3. イベントのサブスクライブ

以下のスニペットは、iot チャネルのイベントをリッスンします。withPresens オプションを追加すると、チャネル参加(Live asf)イベントも受信されます。

```
from("pubnub:iot?subscribeKey=mySubscribeKey")
    .log("${body}")
    .to("mock:result");
```

253.5.4. 操作の実行

こちらの概要： チャネルに現在サブスクライブしている一意のユーザー ID の一覧やチャネルの合計占有数など、チャンネルの現在の状態に関する情報を取得します。

```
from("direct:control")
    .to("pubnub:myChannel?
publishKey=mypublishKey&subscribeKey=mySubscribeKey&operation=herenow")
    .to("mock:result");
```

wherenow : uuid がサブスクライブしているチャンネル一覧に関する情報を取得します。

```
from("direct:control")
    .to("pubnub:myChannel?
publishKey=mypublishKey&subscribeKey=mySubscribeKey&operation=wherenow&uuid=spy
```

```
onme")
    .to("mock:result");
```

setState: サブスクライバー uuid に固有のキーと値のペアを設定するのに使用します。

```
from("direct:control")
    .bean(StateGenerator.class)
    .to("pubnub:myChannel?
publishKey=mypublishKey&subscribeKey=mySubscribeKey&operation=setstate&uuid=myuui
d");
```

gethistory: チャネルの履歴メッセージを取得します。

```
from("direct:control")
    .to("pubnub:myChannel?
publishKey=mypublishKey&subscribeKey=mySubscribeKey&operation=gethistory");
```

テストディレクトリーに、PubNub 機能の一部を示すいくつかの例があります。これらには、PubNub アカウントが必要です。そこからパブリッシュとサブスクライブキーを取得できます。

サンプル PubNubSensorExample には PubNub が提供するサブスクライブキーがすでに含まれているため、これはアカウントなしで実行できます。この例では、センサーデータの無限ストリームにサブスクライブする PubNub コンポーネントを示しています。

253.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- RSS

第254章 QUARTZ コンポーネント（非推奨）

Camel バージョン 1.0 で利用可能

quartz: コンポーネントは Quartz スケジューラー 1.x を使用してスケジュールされたメッセージの配信を提供します。
それぞれのエンドポイントは、異なるタイマー（Quartz 用語、Trigger および JobDetail）を表します。

ヒント

Quartz 2.xを使用している場合、Camel 2.12 以降では Quartz2 コンポーネントを使用する必要があります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-quartz</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

254.1. URI 形式

```
quartz://timerName?options
quartz://groupName/timerName?options
quartz://groupName/timerName?cron=expression
quartz://timerName?cron=expression
```

コンポーネントは CronTrigger または SimpleTrigger のいずれかを使用します。cron 式が指定されていない場合、コンポーネントは単純なトリガーを使用します。groupName が指定されていない場合、quartz コンポーネントは Camel グループ名を使用します。

URI にクエリーオプションを追加するには、?option=value&option=value&…

254.2. オプション

Quartz コンポーネントは、以下に示す 8 個のオプションをサポートします。

Name	説明	デフォルト	Type
factory （詳細）	スケジューラーの作成に使用されるカスタム SchedulerFactory を使用するには、以下を実行します。		SchedulerFactory
スケジューラー (advanced)	新しいスケジューラーを作成する代わりに、設定したカスタム Quartz スケジューラーを使用します。		スケジューラー
プロパティー （コンシューマー）	Quartz スケジューラーを設定するためのプロパティー		プロパティー
propertiesFile (consumer)	クラスパスからロードするプロパティーのファイル名		文字列
startDelayedSeconds (scheduler)	quartz スケジューラーが起動するまでの待機時間（秒単位）。		int
autoStartScheduler (consumer)	スケジューラーを自動起動するかどうか。このオプションはデフォルトの true です。	true	boolean
enableJmx (consumer)	JMX から Quartz スケジューラーを管理できる Quartz JMX を有効にするかどうか。このオプションはデフォルトの true です。	true	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Quartz エンドポイントは、URI 構文を使用して設定します。

```
quartz:groupName/timerName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

254.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
groupName	使用する quartz グループ名。グループ名とタイマー名の組み合わせは一意である必要があります。	Camel	文字列
timerName	使用する quartz タイマー名が **必要です**。グループ名とタイマー名の組み合わせは一意である必要があります。		文字列

254.2.2. クエリーパラメーター（13 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
cron （コンシューマー）	トリガーのタイミングを定義する cron 式を指定します。		文字列
deleteJob (consumer)	true に設定すると、ルートの停止時にトリガーが自動的に削除されます。false に設定された場合は、スケジューラーに残ります。false に設定すると、ユーザーが camel Uri で事前設定されたトリガーを再利用できることを意味します。名前が一致することを確認します。deleteJob と pauseJob の両方が true に設定されているわけではないことに注意してください。	true	boolean
fireNow (consumer)	単純なトリガーを使用して開始したときにスケジューラー asap を実行するかどうか（このオプションは cron をサポートしません）	false	boolean
pauseJob （コンシューマー）	true に設定すると、トリガーはルートが停止したときに自動的に一時停止します。false に設定された場合は、スケジューラーに残ります。false に設定すると、ユーザーが camel Uri で事前設定されたトリガーを再利用できることを意味します。名前が一致することを確認します。deleteJob と pauseJob の両方が true に設定されているわけではないことに注意してください。	false	boolean

Name	説明	デフォルト	Type
startDelayedSeconds (consumer)	quartz スケジューラーが起動するまでの待機時間（秒単位）。		int
Stateful (consumer)	デフォルトジョブの代わりに Quartz StatefulJob を使用します。	false	boolean
usingFixedCamelContextName (consumer)	true の場合、JobDataMap は CamelContext 名を直接使用して CamelContext を参照します（false の場合、JobDataMap はデプロイ中に変更可能な CamelContext 管理名を使用します）。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
jobParameters (advanced)	ジョブに追加オプションを設定します。		マップ
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
triggerParameters (advanced)	トリガーで追加オプションを設定するには、以下を行います。		マップ

***StatefulJob** を使用する場合、**JobDataMap** はジョブの実行ごとに再構成されるため、次の実行の状態を保持します。*

INFO: OSGi で実行し、quartz ルートを持つ複数のバンドル がある場合には、Apache ServiceMix や Apache Karaf などの OSGi で実行し、Quartz エンドポイントから開始する Camel ルートを持つ複数のバンドルがある場合は、OSGi コンテナーの QuartzScheduler で必要となるため、ID が一意である <camelContext> に ID を割り当てるようにしてください。
<camelContext> に id を設定しない場合、
一意の ID が自動的に割り当てられ、問題はありません。

254.3. QUARTZ.PROPERTIES ファイルの設定

デフォルトでは、Quartz はクラスパスの org/quartz ディレクトリーで quartz.properties ファイルを検索します。WAR デプロイメントを使用している場合は、WEB-INF/classes/org/quartz で quartz.properties を削除するだけです。

しかし、Camel Quartz コンポーネントでは、プロパティーを設定することもできます。

パラメーター	デフォルト	Type	説明
properties	**null**	**プロパティー**	Camel 2.4: **java.util.Properties** インスタンスを設定できます。
propertiesFile	**null**	**文字列**	Camel 2.4: クラスパスからロードするプロパティーのファイル名

これを行うには、以下のように Spring XML でこれを設定します。

```
<bean id="quartz" class="org.apache.camel.component.quartz.QuartzComponent">
    <property name="propertiesFile" value="com/mycompany/myquartz.properties"/>
</bean>
```

254.4. JMX での QUARTZ スケジューラーの有効化

JMX を有効にするには、quartz スケジューラープロパティーを設定する必要があります。これは通常、設定ファイルの true 値に "org.quartz.scheduler.jmx.export" オプションを設定します。

Camel 2.13 以降では、Camel は明示的に無効にならない限り、このオプションを true に自動設定します。

254.5. QUARTZ スケジューラーの起動

以下に例を示します。

```
<bean id="quartz" class="org.apache.camel.component.quartz.QuartzComponent">
    <property name="startDelayedSeconds" value="5"/>
</bean>
```

254.6. クラスタリング

Camel 2.4 で利用可能

クラスター化されたモードで Quartz を使用する場合、たとえば JobStore はクラスター化されます。その後、Camel 2.4 以降では、Quartz コンポーネントはノードが停止/シャットダウン時にトリガーを一時停止/削除し ません。これにより、トリガーがクラスター内の他のノードで稼働を継続できます。

注記: クラスター化されたノードで実行している場合、エンドポイントの一意のジョブ名/グループを確保するためにチェックは行われません。

254.7. メッセージヘッダー

Camel は Quartz Execution Context からのゲッターをヘッダー値として追加します。次のヘッダーが追加されました。
calendar、fireTime,jobDetail,jobInstance,jobRuntTime,mergedJobDataMap,nextFireTime,previousFireTime,refireCount,result, result
,scheduledFireTime,scheduler,trigger,triggerName,triggerGroup

fireTime ヘッダーには、エクスチェンジが発生したときの java.util.Date が含まれます。

254.8. CRON トリガーの使用

Quartz は、タイマーを便利な形式で指定する Cron のような式 をサポートします。これらの式は cron URI パラメーターで使用できますが、有効な URI エンコーディングを維持するには、スペースの代わりに + を使用することができます。Quartz は、cron 式 の使用方法に関する簡単なチュートリアル を提供します。

たとえば、以下は平日に 12pm(noon)から 6pm までの 5 分ごとにメッセージを実行します。

```
from("quartz://myGroup/myTimerName?cron=0+0/5+12-18+?+*+MON-
FRI").to("activemq:Totally.Rocks");
```

これは cron 式を使用するのと同じです。

```
0 0/5 12-18 ? * MON-FRI
```

以下の表は、有効な URI 構文を保持するために使用する URI 文字エンコーディングを示しています。

URI 文字	cron 文字
+	スペース

254.9. タイムゾーンの指定

Camel 2.8.1
利用可能なQuartz スケジューラーでは、トリガーごとにタイムゾーンを設定できます。たとえば、国のタイムゾーンを使用するには、以下のように実行できます。

```
quartz://groupName/timerName?cron=0+0/5+12-18+?+*+MON-FRI&trigger.timeZone=Europe/Stockholm
```

timeZone 値は、java.util.TimeZone で使用できる値です。

Camel 2.8.0 以前のバージョンでは、これをエンドポイント URI から設定できるようにカスタム String を java.util.TimeZone Type Converter に指定する必要があります。
Camel 2.8.1 以降では、camel-core にこのような型コンバーターが含まれています。

254.10. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- Quartz2

- *Timer*

第255章 QUARTZ2 COMPONENT

Camel バージョン 2.12 から利用可能

quartz2: コンポーネントは Quartz スケジューラー 2.x を使用してスケジュールされたメッセージの配信を提供します。
それぞれのエンドポイントは、異なるタイマー（Quartz 用語、Trigger および JobDetail）を表します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-quartz2</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

注記： Quartz 2.x API は Quartz 1.x と互換性がありません。古い Quartz 1.x 上に維持する必要がある場合は、代わりに古い Quartz コンポーネントを使用してください。

255.1. URI 形式

```
quartz2://timerName?options
quartz2://groupName/timerName?options
quartz2://groupName/timerName?cron=expression
quartz2://timerName?cron=expression
```

コンポーネントは CronTrigger または SimpleTrigger のいずれかを使用します。cron 式が指定されていない場合、コンポーネントは単純なトリガーを使用します。groupName が指定されていない場合、quartz コンポーネントは Camel グループ名を使用します。

URI にクエリーオプションを追加するには、?option=value&option=value&…

255.2. オプション

Quartz2 コンポーネントは、以下に示す 11 個のオプションをサポートします。

Name	説明	デフォルト	Type
autoStartScheduler (scheduler)	スケジューラーを自動起動するかどうか。このオプションはデフォルトの true です。	true	boolean
startDelayedSeconds (scheduler)	quartz スケジューラーが起動するまでの待機時間（秒単位）。		int
prefixJobNameWith EndpointId (consumer)	quartz ジョブの前にエンドポイント ID を付けるかどうか。このオプションはデフォルトの false です。	false	boolean
enableJmx (consumer)	JMX から Quartz スケジューラーを管理できる Quartz JMX を有効にするかどうか。このオプションはデフォルトの true です。	true	boolean
プロパティー（コンシューマー）	Quartz スケジューラーを設定するためのプロパティー		プロパティー
propertiesFile (consumer)	クラスパスからロードするプロパティーのファイル名		文字列
prefixInstanceName (consumer)	Quartz Scheduler インスタンス名に CamelContext 名のプレフィックスを追加するかどうか。これは、各 CamelContext にデフォルトで独自の Quartz スケジューラーインスタンスを使用するようにするためにデフォルトで有効になっています。このオプションを false に設定すると、複数の CamelContext 間で Quartz スケジューラーインスタンスを再利用できます。	true	boolean
interruptJobsOn Shutdown (scheduler)	シャットダウン時にジョブを中断するかどうか。これによりスケジューラーがより迅速にシャットダウンし、実行中のジョブを中断しようとします。これを有効にすると、中断されたために実行中のジョブが失敗する場合があります。	false	boolean
schedulerFactory (advanced)	スケジューラーの作成に使用されるカスタム SchedulerFactory を使用するには、以下を実行します。		SchedulerFactory
スケジューラー (advanced)	新しいスケジューラーを作成する代わりに、設定したカスタム Quartz スケジューラーを使用します。		スケジューラー
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Quartz2 エンドポイントは、URI 構文を使用して設定します。

```
quartz2:groupName/triggerName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

255.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
groupName	使用する quartz グループ名。グループ名とタイマー名の組み合わせは一意である必要があります。	Camel	文字列
triggerName	使用する quartz タイマー名が **必要です**。グループ名とタイマー名の組み合わせは一意である必要があります。		文字列

255.2.2. クエリーパラメーター（19 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
cron （コンシューマー）	トリガーのタイミングを定義する cron 式を指定します。		文字列
deleteJob (consumer)	true に設定すると、ルートの停止時にトリガーが自動的に削除されます。false に設定された場合は、スケジューラーに残ります。false に設定すると、ユーザーが camel Uri で事前設定されたトリガーを再利用できることを意味します。名前が一致することを確認します。deleteJob と pauseJob の両方が true に設定されているわけではないことに注意してください。	true	boolean

Name	説明	デフォルト	Type
durableJob (consumer)	ジョブが孤立した後も保存すべきかどうか（トリガーはポイントなし）。	false	boolean
pauseJob （コンシューマー）	true に設定すると、トリガーはルートが停止したときに自動的に一時停止します。false に設定された場合は、スケジューラーに残ります。false に設定すると、ユーザーが camel Uri で事前設定されたトリガーを再利用できることを意味します。名前が一致することを確認します。deleteJob と pauseJob の両方が true に設定されているわけではないことに注意してください。	false	boolean
recoverableJob (consumer)	'recovery' または 'fail-over' の状況が発生した場合にジョブを再実行すべきかどうかをスケジューラーに指示します。	false	boolean
Stateful (consumer)	デフォルトジョブの代わりに Quartz PersistJobDataAfterExecution および DisallowConcurrentExecution を使用します。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
customCalendar (advanced)	特定の日付の範囲を回避するためにカスタムカレンダーを指定します。		calendar
jobParameters (advanced)	ジョブに追加オプションを設定します。		マップ
prefixJobNameWithEndpoint Id (advanced)	ジョブ名の前にエンドポイント ID を付けるかどうか	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
triggerParameters (advanced)	トリガーで追加オプションを設定するには、以下を行います。		マップ

Name	説明	デフォルト	Type
usingFixedCamelContextName (advanced)	true の場合、JobDataMap は CamelContext 名を直接使用して CamelContext を参照します（false の場合、JobDataMap はデプロイ中に変更可能な CamelContext 管理名を使用します）。	false	boolean
autoStartScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
fireNow (scheduler)	true の場合、SimpleTrigger の使用時にルートの開始時にトリガーが実行されます。	false	boolean
startDelayedSeconds (scheduler)	quartz スケジューラーが起動するまでの待機時間（秒単位）。		int
triggerStartDelay (scheduler)	スケジューラーがすでに起動している場合は、ジョブの開始前にエンドポイントが完全に起動されるように、現在の時刻後にトリガーを少し開始します。	500	Long

たとえば、以下のルーティングルールは、2 つのタイマーイベントを mock:results エンドポイントに対して実行します。

```
from("quartz2://myGroup/myTimerName?
trigger.repeatInterval=2&trigger.repeatCount=1").routeId("myRoute")
    .to("mock:result");
```

stateful=true を使用する場合、JobDataMap はジョブの実行するたびに再永続するため、次の実行の状態を保持します。

INFO: OSGi で実行し、quartz ルートを持つ複数のバンドル がある場合には、Apache ServiceMix や Apache Karaf などの OSGi で実行し、Quartz2 エンドポイントから開始する Camel ルートを持つ複数のバンドルがある場合は、OSGi コンテナーの QuartzScheduler で必要となるため、この id が一意である <camelContext> に ID を割り当てるようにしてください。<camelContext> に id を設定しないと、一意の ID が自動的に割り当てられ、問題はありません。

255.3. QUARTZ.PROPERTIES ファイルの設定

デフォルトでは、Quartz はクラスパスの org/quartz ディレクトリーで quartz.properties ファイルを検索します。WAR デプロイメントを使用している場合は、WEB-INF/classes/org/quartz で quartz.properties を削除するだけです。

しかし、Camel Quartz2 コンポーネントでは、プロパティーを設定することもできます。

パラメーター	デフォルト	Type	説明
properties	null	プロパティー	java.util.Properties インスタンスを設定できます。
propertiesFile	null	文字列	クラスパスからロードするプロパティーのファイル名

これを行うには、以下のように Spring XML でこれを設定します。

```
<bean id="quartz2" class="org.apache.camel.component.quartz2.QuartzComponent">
    <property name="propertiesFile" value="com/mycompany/myquartz.properties"/>
</bean>
```

255.4. JMX での QUARTZ スケジューラーの有効化

JMX を有効にするには、quartz スケジューラープロパティーを設定する必要があります。
これは通常、設定ファイルの true 値に "org.quartz.scheduler.jmx.export" オプションを設定します。

Camel 2.13 以降では、Camel は明示的に無効にならない限り、このオプションを true に自動設定します。

255.5. QUARTZ スケジューラーの起動

Quartz2 コンポーネントは、Quartz スケジューラーを遅延させたり、自動的に開始したりしないようにするオプションを提供します。

以下に例を示します。

```
<bean id="quartz2" class="org.apache.camel.component.quartz2.QuartzComponent">
    <property name="startDelayedSeconds" value="5"/>
</bean>
```

255.6. クラスタリング

クラスター化されたモードで Quartz を使用する場合、たとえば JobStore はクラスター化されます。この場合、Quartz2 コンポーネントはノードが停止/シャットダウンされるとトリガーを一時停止/削除し ません。これにより、トリガーがクラスター内の他のノードで稼働を継続できます。

注記: クラスター化されたノードで実行している場合、エンドポイントの一意のジョブ名/グループを確保するためにチェックは行われません。

255.7. メッセージヘッダー

Camel は Quartz Execution Context からのゲッターをヘッダー値として追加します。次のヘッダーが追加されました。
calendar、 fireTime,jobDetail,jobInstance,jobRuntTime,mergedJobDataMap,nextFireTime,previousFireTime,refireCount,result, result
,scheduledFireTime,scheduler,trigger,triggerName,triggerGroup

fireTime ヘッダーには、エクスチェンジが発生したときの java.util.Date が含まれます。

255.8. CRON トリガーの使用

Quartz は、タイマーを便利な形式で指定する Cron のような式 をサポートします。これらの式は cron URI パラメーターで使用できますが、有効な URI エンコーディングを維持するには、スペースの代わりに + を使用することができます。

たとえば、以下は平日に 12pm(noon)から 6pm までの 5 分ごとにメッセージを実行します。

```
from("quartz2://myGroup/myTimerName?cron=0+0/5+12-18+?+*+MON-FRI")
    .to("activemq:Totally.Rocks");
```

これは cron 式を使用するのと同じです。

```
0 0/5 12-18 ? * MON-FRI
```

以下の表は、有効な URI 構文を保持するために使用する URI 文字エンコーディングを示しています。

URI 文字	cron 文字
+	スペース

255.9. タイムゾーンの指定

Quartz スケジューラーでは、トリガーごとにタイムゾーンを設定できます。たとえば、国のタイムゾーンを使用するには、以下のように実行できます。

```
quartz2://groupName/timerName?cron=0+0/5+12-18+?+*+MON-FRI&trigger.timeZone=Europe/Stockholm
```

timeZone 値は、java.util.TimeZone で使用できる値です。

255.10. USING QUARTZSCHEDULEDPOLLCONSUMERSCHEDULER

Quartz2 コンポーネントは、File や FTP コンシューマーなどのポーリングコンシューマーに cron ベースのスケジューリングを使用できる Polling Consumer スケジューラーを提供します。

たとえば、2 秒あたりに cron ベースの式を使用してファイルをポーリングするには、Camel ルートを以下のように定義できます。

```
from("file:inbox?scheduler=quartz2&scheduler.cron=0/2+*+*+*+*+?")
  .to("bean:process");
```

scheduler=quartz2 を定義して、Camel に Quartz2 ベースのスケジューラーを使用するように指示します。次に、scheduler.xxx オプションを使用してスケジューラーを設定します。Quartz2 スケジューラーでは、cron オプションを設定する必要があります。

以下のオプションがサポートされます。

パラメーター	デフォルト	Type	説明
quartzScheduler	**null**	**org.quartz.Scheduler**	カスタムの Quartz スケジューラーを使用するには、以下を行います。設定されていない場合は、Quartz2 コンポーネントから共有スケジューラーが使用されます。

パラメーター	デフォルト	Type	説明
cron	**null**	**文字列**	**必須**: ポーリングをトリガーする cron 式を定義します。
triggerId	**null**	**文字列**	トリガー ID を指定します。指定がない場合は UUID が生成され、使用されます。
triggerGroup	**QuartzScheduledPollConsumerScheduler**	**文字列**	トリガーグループを指定します。
timeZone	**デフォルト**	**TimeZone**	CRON トリガーに使用するタイムゾーン。

重要： エンドポイント URI からこれらのオプションを設定する場合は、scheduler をプレフィックスとして指定してください。たとえば、トリガー ID およびグループを設定するには、以下を実行します。

```
from("file:inbox?scheduler=quartz2&scheduler.cron=0/2+*+*+*+*+?
&scheduler.triggerId=myId&scheduler.triggerGroup=myGroup")
    .to("bean:process");
```

Spring には CRON スケジューラーもあります。そのため、以下を使用することもできます。

```
from("file:inbox?scheduler=spring&scheduler.cron=0/2+*+*+*+*+?")
    .to("bean:process");
```

第256章 RABBITMQ コンポーネント

Camel バージョン 2.12 から利用可能

rabbitmq: コンポーネントを使用すると、RabbitMQ インスタンスからメッセージを生成および消費できます。このコンポーネントは、RabbitMQ AMQP クライアントを使用して、汎用 AMQP コンポーネントに純粋な RabbitMQ アプローチを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rabbitmq</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

256.1. URI 形式

以前の構文は 非推奨 となっています。

```
rabbitmq://hostname[:port]/exchangeName?[options]
```

代わりに、hostname と port がコンポーネントレベルに設定されるか、代わりに uri クエリーパラメーターとして指定できます。

新しい構文は以下のとおりです。

```
rabbitmq:exchangeName?[options]
```

hostname は、実行中の rabbitmq インスタンスまたはクラスターのホスト名です。port は任意で、指定されていない場合には、RabbitMQ クライアントのデフォルト(5672)にデフォルト設定されます。エクスチェンジ名は、生成されたメッセージの送信先を決定します。コンシューマーの場合、エクスチェンジ名はキューがバインドするエクスチェンジを決定します。

256.2. オプション

RabbitMQ コンポーネントは、以下に示す 49 個のオプションをサポートしています。

Name	説明	デフォルト	Type
hostname (common)	実行中の RabbitMQ インスタンスまたはクラスターのホスト名。		文字列
portNumber (common)	実行中の rabbitmq インスタンスまたはクラスターを使用するホストのポート番号。	5672	int
ユーザー名 （セキュリティー）	認証されたアクセスの場合のユーザー名	guest	文字列
パスワード （セキュリティー）	認証されたアクセスのパスワード	guest	文字列
vhost (common)	チャネルの vhost	/	文字列
アドレス （共通）	このオプションを設定すると、camel-rabbitmq はオプションアドレスの設定に基づいて接続の作成を試みます。addresses の値は、server1:12345、server2:12345 などの文字列です。		文字列
connectionFactory (common)	カスタムの RabbitMQ 接続ファクトリーを使用するには、以下を行います。このオプションを設定すると、URI に設定したすべての接続オプション（connectionTimeout、requestedChannelMax...）は使用されません。		ConnectionFactory
threadPoolSize (consumer)	コンシューマーは、スレッド数が固定されたスレッドプールエグゼキューターを使用します。この設定により、スレッドの数を設定できます。	10	int
autoDetectConnection Factory (advanced)	レジストリーから RabbitMQ 接続ファクトリーを検索するかどうか。有効で、接続ファクトリーのインスタンスを 1 つ見つかると、それが使用されます。明示的な接続ファクトリーは、優先されるコンポーネントまたはエンドポイントレベルに設定できます。	true	boolean
connectionTimeout (advanced)	接続のタイムアウト	60000	int
requestedChannelMax (advanced)	接続要求チャンネルの最大値（提供されているチャンネルの最大数）	0	int
requestedFrameMax (advanced)	接続に要求されたフレーム最大数（提供されているフレームの最大サイズ）	0	int

Name	説明	デフォルト	Type
requestedHeartbeat (advanced)	ハートビートが要求した接続（秒単位）	60	int
AutomaticRecovery Enabled （詳細）	接続の自動リカバリーを有効にします（接続シャットダウンがアプリケーションによって開始されない場合に自動リカバリーを実行する接続実装を使用）		ブール値
networkRecoveryInterval (advanced)	ネットワーク復旧間隔（ミリ秒単位）（ネットワーク障害からの復旧に使用される間隔）	5000	整数
topologyRecoveryEnabled (advanced)	接続トポロジーのリカバリーを有効にします（トポロジーのリカバリーが実行される場合）。		ブール値
prefetchEnabled (consumer)	RabbitMQConsumer 側でサービスの品質を有効にします。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。	false	boolean
prefetchSize (consumer)	サーバーが配信するコンテンツの最大量（octets で測定）（無制限の場合は 0）。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。		int
prefetchCount (consumer)	サーバーが配信するメッセージの最大数。無制限の場合は 0。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。		int
prefetchGlobal (consumer)	設定が各コンシューマーではなくチャネル全体に適用される必要がある場合、prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。	false	boolean
channelPoolMaxSize (producer)	プールの開いているチャンネルの最大数を取得します。	10	int
channelPoolMaxWait (producer)	プールからチャネルを待つ最大時間（ミリ秒単位）を設定します。	1000	Long
requestTimeout (advanced)	InOut エクスチェンジパターン（ミリ秒単位）を使用する場合の応答の待機のタイムアウトの設定	20000	Long
requestTimeoutChecker Interval (advanced)	InOut エクスチェンジの requestTimeoutCheckerInterval の設定	1000	Long

Name	説明	デフォルト	Type
transferException (advanced)	true であり、コンシューマー側で inOut エクスチェンジが失敗すると、応答で原因の例外が戻されます。	false	boolean
パブリッシャー Acknowledgement（プロデューサー）	true の場合、メッセージはパブリッシャーの確認応答がオンになります。	false	boolean
パブリッシャー AcknowledgementsTimeout (producer)	RabbitMQ サーバーからの基本的な.ack 応答を待つ時間（ミリ秒単位）。		Long
guaranteedDeliveries (producer)	true の場合、メッセージを配信できない場合(basic.return)に例外がスローされ、メッセージが mandatory とマークされます。この場合、PublisherAcknowledgement もアクティベートされます。「パブリッシャーの確認」も参照してください。メッセージを確認する場合は、を参照してください。	false	boolean
必須（プロデューサー）	このフラグは、メッセージがキューにルーティングできない場合にサーバーがどのように反応するかを指示します。このフラグが設定されている場合、サーバーは Return メソッドでルーティング不可能なメッセージを返します。このフラグがゼロの場合、サーバーはメッセージを警告なしでドロップします。ヘッダーが rabbitmq.MANDATORY が存在する場合は、このオプションが上書きされます。	false	boolean
即時（プロデューサー）	このフラグは、メッセージをキューコンシューマーにすぐにルーティングできない場合にサーバーがどのように反応するかを指示します。このフラグが設定されている場合、サーバーは Return メソッドで未配信メッセージを返します。このフラグがゼロの場合、サーバーはメッセージをキューに入れますが、消費される保証はありません。ヘッダーに rabbitmq.IMMEDIATE がある場合には、このオプションが上書きされます。	false	boolean
引数（詳細）	さまざまな RabbitMQ の概念を設定するための引数を指定します。Exchange: arg.exchange ごとに異なる接頭辞が必要です。queue: arg.queue。Binding: arg.binding.たとえば、メッセージスロット引数でキューを宣言するには、http://localhost:5672/exchange/queueeargs=arg.queue.x-message-ttl=60000します。		マップ

Name	説明	デフォルト	Type
clientProperties (advanced)	接続クライアントプロパティー（サーバーとネゴシエートで使用されるクライアント情報）		マップ
sslProtocol (security)	接続で SSL を有効にします。許可される値は true、TLS、および 'SSLv3 です。		文字列
trustManager (security)	SSL トラストマネージャーを設定します。このオプションを有効にするには SSL を有効にする必要があります。		TrustManager
autoAck (consumer)	メッセージが自動承認されるかどうか。	true	boolean
autoDelete (common)	true の場合、エクスチェンジが使用されなくなると、エクスチェンジが削除されます。	true	boolean
永続性 (common)	永続エクスチェンジを宣言する場合（エクスチェンジはサーバーの再起動後も存続します）。	true	boolean
排他 的(common)	排他キューには現在の接続によってのみアクセスでき、その接続が閉じられると削除されます。	false	boolean
passive (common)	パッシブキューは、RabbitMQ ですでに利用可能なキューによって異なります。	false	boolean
skipQueueDeclare (common)	true の場合、プロデューサーはキューを宣言およびバインドしません。これは、既存のルーティングキー経由でメッセージを送信するために使用できます。	false	boolean
skipQueueBind (common)	true の場合、キューは宣言後にエクスチェンジにバインドされません。	false	boolean
skipExchangeDeclare (common)	キューを宣言してエクスチェンジを宣言しない場合に使用できます。	false	boolean
宣言 （共通）	オプションが true の場合、Camel はエクスチェンジおよびキュー名を宣言し、それらをバインドします。オプションが false の場合、Camel はサーバーで交換およびキュー名を宣言しません。	true	boolean
deadLetterExchange (common)	デッドレターエクスチェンジの名前		文字列
deadLetterQueue (common)	デッドレターキューの名前		文字列

Name	説明	デフォルト	Type
deadLetterRoutingKey (common)	デッドレターエクスチェンジのルーティングキー		文字列
deadLetterExchangeType (common)	デッドレターエクスチェンジのタイプ	直接的な	文字列
allowNullHeaders (producer)	null 値をヘッダーに渡すことを許可	false	boolean
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

RabbitMQ エンドポイントは、URI 構文を使用して設定します。

```
rabbitmq:exchangeName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

256.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
exchangeName	***必須***。エクスチェンジ名は、生成されたメッセージを送信するエクスチェンジを決定します。コンシューマーの場合、エクスチェンジ名はキューがバインドするエクスチェンジを決定します。		文字列

256.2.2. クエリーパラメーター（61 パラメーター）：

Name	説明	デフォルト	Type
アドレス（共通）	このオプションを設定すると、camel-rabbitmq はオプションアドレスの設定に基づいて接続の作成を試みます。addresses の値は、server1:12345、server2:12345 などの文字列です。		address[]

Name	説明	デフォルト	Type
autoDelete (common)	true の場合、エクスチェンジが使用されなくなると、エクスチェンジが削除されます。	true	boolean
connectionFactory (common)	カスタムの RabbitMQ 接続ファクトリーを使用するには、以下を行います。このオプションを設定すると、URI に設定したすべての接続オプション（connectionTimeout、requestedChannelMax...）は使用されません。		ConnectionFactory
deadLetterExchange (common)	デッドレターエクスチェンジの名前		文字列
deadLetterExchangeType (common)	デッドレターエクスチェンジのタイプ	直接的な	文字列
deadLetterQueue (common)	デッドレターキューの名前		文字列
deadLetterRoutingKey (common)	デッドレターエクスチェンジのルーティングキー		文字列
宣言 （共通）	オプションが true の場合、Camel はエクスチェンジおよびキュー名を宣言し、それらをバインドします。オプションが false の場合、Camel はサーバーで交換およびキュー名を宣言しません。	true	boolean
永続性 (common)	永続エクスチェンジを宣言する場合（エクスチェンジはサーバーの再起動後も存続します）。	true	boolean
exchangeType (common)	direct または topic などのエクスチェンジタイプ。	直接的な	文字列
排他 的(common)	排他キューには現在の接続によってのみアクセスでき、その接続が閉じられると削除されます。	false	boolean
hostname (common)	実行中の rabbitmq インスタンスまたはクラスターのホスト名。		文字列
passive (common)	パッシブキューは、RabbitMQ ですでに利用可能なキューによって異なります。	false	boolean
portNumber (common)	実行中の rabbitmq インスタンスまたはクラスターを使用するホストのポート番号。デフォルト値は 5672 です。		int

Name	説明	デフォルト	Type
キュー （共通）	メッセージの受信元であるキュー		文字列
routingKey (common)	コンシューマーキューをエクスチェンジにバインドする際に使用するルーティングキー。プロデューサールーティングキーには、ヘッダー rabbitmq.ROUTING_KEY を設定します。		文字列
skipExchangeDeclare (common)	キューを宣言してエクスチェンジを宣言しない場合に使用できます。	false	boolean
skipQueueBind (common)	true の場合、キューは宣言後にエクスチェンジにバインドされません。	false	boolean
skipQueueDeclare (common)	true の場合、プロデューサーはキューを宣言およびバインドしません。これは、既存のルーティングキー経由でメッセージを送信するために使用できます。	false	boolean
vhost (common)	チャネルの vhost	/	文字列
autoAck (consumer)	メッセージが自動承認されるかどうか。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	ブローカーから消費する場合の同時コンシューマーの数（JMS コンポーネントと同じオプションと同様）。	1	int
prefetchCount (consumer)	サーバーが配信するメッセージの最大数。無制限の場合は 0。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。		int
prefetchEnabled (consumer)	RabbitMQConsumer 側でサービスの品質を有効にします。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。	false	boolean

Name	説明	デフォルト	Type
prefetchGlobal (consumer)	設定が各コンシューマーではなくチャネル全体に適用される必要がある場合、prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。	false	boolean
prefetchSize (consumer)	サーバーが配信するコンテンツの最大量（octets で測定）（無制限の場合は 0）。prefetchSize、prefetchCount、prefetchGlobal のオプションを同時に指定する必要があります。		int
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
threadPoolSize (consumer)	コンシューマーは、スレッド数が固定されたスレッドプールエグゼキューターを使用します。この設定により、スレッドの数を設定できます。	10	int
allowNullHeaders (producer)	null 値をヘッダーに渡すことを許可	false	boolean
bridgeEndpoint (producer)	bridgeEndpoint が true の場合、プロデューサーは rabbitmq.EXCHANGE_NAME および rabbitmq.ROUTING_KEY のメッセージヘッダーを無視します。	false	boolean
channelPoolMaxSize (producer)	プールの開いているチャンネルの最大数を取得します。	10	int
channelPoolMaxWait (producer)	プールからチャネルを待つ最大時間（ミリ秒単位）を設定します。	1000	Long
guaranteedDeliveries (producer)	true の場合、メッセージを配信できない場合(basic.return)に例外がスローされ、メッセージが mandatory とマークされます。この場合、PublisherAcknowledgement もアクティベートされます。「パブリッシャーの確認」も参照してください。メッセージを確認する場合は、を参照してください。	false	boolean

Name	説明	デフォルト	Type
即時 （プロデューサー）	このフラグは、メッセージをキューコンシューマーにすぐにルーティングできない場合にサーバーがどのように反応するかを指示します。このフラグが設定されている場合、サーバーは Return メソッドで未配信メッセージを返します。このフラグがゼロの場合、サーバーはメッセージをキューに入れますが、消費される保証はありません。ヘッダーに rabbitmq.IMMEDIATE がある場合には、このオプションが上書きされます。	false	boolean
必須 （プロデューサー）	このフラグは、メッセージがキューにルーティングできない場合にサーバーがどのように反応するかを指示します。このフラグが設定されている場合、サーバーは Return メソッドでルーティング不可能なメッセージを返します。このフラグがゼロの場合、サーバーはメッセージを警告なしでドロップします。ヘッダーが rabbitmq.MANDATORY が存在する場合は、このオプションが上書きされます。	false	boolean
publisherAcknowledgements (producer)	true の場合、メッセージはパブリッシャーの確認応答がオンになります。	false	boolean
publisherAcknowledgements Timeout (producer)	RabbitMQ サーバーからの基本的な.ack 応答を待つ時間（ミリ秒単位）。		Long
引数 （詳細）	さまざまな RabbitMQ の概念を設定するための引数を指定します。Exchange: arg.exchange ごとに異なる接頭辞が必要です。queue: arg.queue。Binding: arg.binding.たとえば、メッセージスロット引数でキューを宣言するには、http://localhost:5672/exchange/queueargs=arg.queue.x-message-ttl=60000します。		マップ
automaticRecoveryEnabled (advanced)	接続の自動リカバリーを有効にします（接続シャットダウンがアプリケーションによって開始されない場合に自動リカバリーを実行する接続実装を使用）		ブール値
bindingArgs (advanced)	declare=true の場合のキューバインディングパラメーターを設定するための **非推奨** のキー/値引数		マップ
clientProperties (advanced)	接続クライアントプロパティー（サーバーとネゴシエートで使用されるクライアント情報）		マップ
connectionTimeout (advanced)	接続のタイムアウト	60000	int

Name	説明	デフォルト	Type
exchangeArgs (advanced)	declare=true の場合にエクスチェンジパラメーターを設定するための **非推奨** のキー/値引数		マップ
exchangeArgsConfigurer (advanced)	**非推奨**: Channel.exchangeDeclare にエクスチェンジ引数を設定するように configurer を設定		ArgsConfigurer
networkRecoveryInterval (advanced)	ネットワーク復旧間隔（ミリ秒単位）（ネットワーク障害からの復旧に使用される間隔）	5000	整数
queueArgs (advanced)	declare=true の場合にキューパラメーターを設定するための **非推奨** のキー/値引数		マップ
queueArgsConfigurer (advanced)	**非推奨**: Channel.queueDeclare にキュー引数を設定するための設定		ArgsConfigurer
requestedChannelMax (advanced)	接続要求チャンネルの最大値（提供されているチャンネルの最大数）	0	int
requestedFrameMax (advanced)	接続に要求されたフレーム最大数（提供されているフレームの最大サイズ）	0	int
requestedHeartbeat (advanced)	ハートビートが要求した接続（秒単位）	60	int
requestTimeout (advanced)	InOut エクスチェンジパターン（ミリ秒単位）を使用する場合の応答の待機のタイムアウトの設定	20000	Long
requestTimeoutChecker Interval (advanced)	InOut エクスチェンジの requestTimeoutCheckerInterval の設定	1000	Long
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
topologyRecoveryEnabled (advanced)	接続トポロジーのリカバリーを有効にします（トポロジーのリカバリーが実行される場合）。		ブール値
transferException (advanced)	true であり、コンシューマー側で inOut エクスチェンジが失敗すると、応答で原因の例外が戻されます。	false	boolean

Name	説明	デフォルト	Type
パスワード（セキュリティー）	認証されたアクセスのパスワード	guest	文字列
sslProtocol (security)	接続で SSL を有効にします。許可される値は true、TLS、および 'SSLv3 です。		文字列
trustManager (security)	SSL トラストマネージャーを設定します。このオプションを有効にするには SSL を有効にする必要があります。		TrustManager
ユーザー名（セキュリティー）	認証されたアクセスの場合のユーザー名	guest	文字列

コネクションオプションの詳細は、http://www.rabbitmq.com/releases/rabbitmq-java-client/current-javadoc/com/rabbitmq/client/ConnectionFactory.html および AMQP 仕様を参照してください。

256.3. 接続ファクトリーの使用

RabbitMQ に接続するには、以下のようなログイン詳細で ConnectionFactory （JMS と同じ）を設定できます。

```
<bean id="rabbitConnectionFactory" class="com.rabbitmq.client.ConnectionFactory">
  <property name="host" value="localhost"/>
  <property name="port" value="5672"/>
  <property name="username" value="camel"/>
  <property name="password" value="bugsbunny"/>
</bean>

And then refer to the connection factory in the endpoint uri as shown below:

<camelContext>
  <route>
    <from uri="direct:rabbitMQEx2"/>
    <to uri="rabbitmq:ex2?connectionFactory=#rabbitConnectionFactory"/>
  </route>
</camelContext>
```

Camel 2.21 以降、 ConnectionFactory はデフォルトで自動検出されるため、

```
<camelContext>
  <route>
    <from uri="direct:rabbitMQEx2"/>
```

```
    <to uri="rabbitmq:ex2"/>
  </route>
</camelContext>
```

256.4. メッセージヘッダー

以下のヘッダーは、メッセージの消費時にエクスチェンジに設定されます。

プロパティー	値
rabbitmq.ROUTING_KEY	メッセージの受信に使用したルーティングキー、またはメッセージの生成時に使用されるルーティングキー。
rabbitmq.EXCHANGE_NAME	メッセージが受信されたエクスチェンジ
rabbitmq.DELIVERY_TAG	受信したメッセージの rabbitmq 配信タグ
rabbitmq.REDELIVERY_TAG	メッセージが再配信されるかどうか。
RabbitMQ.REQUEUE	Camel 2.14.2: これは、メッセージの拒否を制御するためにコンシューマーによって使用されます。コンシューマーがエクスチェンジの処理を完了し、エクスチェンジが失敗した場合、コンシューマーは RabbitMQ ブローカーからメッセージを拒否します。このヘッダーの値はこの動作を制御します。値が false（デフォルト）の場合、メッセージは破棄/デッドレターになります。値が true の場合、メッセージは再度キューに入れられます。

以下のヘッダーはプロデューサーによって使用されます。これらのエクスチェンジが camel のエクスチェンジに設定されていると、RabbitMQ メッセージに設定されます。

プロパティー	値
rabbitmq.ROUTING_KEY	メッセージの送信時に使用されるルーティングキー
rabbitmq.EXCHANGE_NAME	メッセージが受信されたエクスチェンジ
rabbitmq.EXCHANGE_OVERRIDE_NAME	Camel 2.21: プロデューサーに設定されたエンドポイントではなく、このエクスチェンジにメッセージを強制的に送信するために使用されます。
rabbitmq.CONTENT_TYPE	RabbitMQ メッセージに設定する contentType
RabbitMQ.PRIORITY	RabbitMQ メッセージに設定する優先度ヘッダー
rabbitmq.CORRELATIONID	RabbitMQ メッセージに設定する correlationId
rabbitmq.MESSAGE_ID	RabbitMQ メッセージに設定するメッセージ ID
rabbitmq.DELIVERY_MODE	メッセージが永続的であるべきかどうか。

プロパティー	値
rabbitmq.USERID	RabbitMQ メッセージに設定する userId
rabbitmq.CLUSTERID	RabbitMQ メッセージに設定する clusterId
rabbitmq.REPLY_TO	RabbitMQ メッセージに設定する replyTo
rabbitmq.CONTENT_ENCODING	RabbitMQ メッセージに設定する contentEncoding
rabbitmq.TYPE	RabbitMQ メッセージに設定するタイプ
RabbitMQ.EXPIRATION	RabbitMQ メッセージに設定する有効期限
rabbitmq.TIMESTAMP	RabbitMQ メッセージに設定するタイムスタンプ
rabbitmq.APP_ID	RabbitMQ メッセージに設定する appId

メッセージが受信されると、ヘッダーはコンシューマーによって設定されます。また、エクスチェンジの実行後、プロデューサーはダウンストリームプロセッサーのヘッダーも設定します。プロデューサーセットが上書きされる前のヘッダー。

256.5. メッセージボディー

コンポーネントは、本文の camel エクスチェンジを rabbit mq メッセージボディーとして使用します。オブジェクトの camel エクスチェンジはバイトアレイに変換する必要があります。それ以外の場合は、プロデューサーは、サポート対象外のボディータイプの例外をスローします。

256.6. サンプル

ルーティングキー B でエクスチェンジ A にバインドされているキューからメッセージを受信するには、以下を行います。

```
from("rabbitmq:A?routingKey=B")
```

自動確認が無効になっている単一のスレッドでキューからメッセージを受信するには、以下を行います。

```
from("rabbitmq:A?routingKey=B&threadPoolSize=1&autoAck=false")
```

C と呼ばれるエクスチェンジにメッセージを送信するには、以下を行います。

```
to("rabbitmq:C")
```

ヘッダーエクスチェンジおよびキューの宣言

```
from("rabbitmq:ex?exchangeType=headers&queue=q&bindingArgs=#bindArgs")
```

また、対応する Map<String, Object& gt; をレジストリーの「bindArgs」の ID に置き換えます。

spring でメソッドを宣言する例

```
@Bean(name="bindArgs")
public Map<String, Object> bindArgsBuilder() {
    return Collections.singletonMap("foo", "bar");
}
```

256.6.1. エクスチェンジ間でルーティングする問題（Camel 2.20.x 以前）

たとえば、以下の例のように、ある Rabbit のエクスチェンジから別のエクスチェンジにメッセージをルーティングする場合は、foo → bar を使用します。

```
from("rabbitmq:foo")
  .to("rabbitmq:bar")
```

その後、Camel はメッセージを自身（例： foo → foo）にルーティングすることに注意してください。なぜそこになのでしょうか？これは、メッセージを受信するコンシューマー（例： from）により、エクスチェンジの名前（例： foo ）を持つメッセージヘッダー rabbitmq.EXCHANGE_NAME が提供されるためです。また、Camel プロデューサーがメッセージを バー に送信すると、ヘッダー rabbitmq.EXCHANGE_NAME はこれを上書きし、代わりにメッセージを foo に送信します。

これを回避するには、以下のいずれかを実行する必要があります。

- ヘッダーを削除します。

```
from("rabbitmq:foo")
  .removeHeader("rabbitmq.EXCHANGE_NAME")
  .to("rabbitmq:bar")
```

- または、プロデューサーで bridgeEndpoint モードをオンにします。

```
from("rabbitmq:foo")
  .to("rabbitmq:bar?bridgeEndpoint=true")
```

Camel 2.21 以降では改善され、エクスチェンジ間の簡単にルーティングできるようになりました。ヘッダー rabbitmq.EXCHANGE_NAME は、宛先の交換を上書きするプロデューサーで使用されなくなりました。代わりに、新しいヘッダー rabbitmq.EXCHANGE_OVERRIDE_NAME を使用して、別のエクスチェンジに送信することができます。たとえば、エクスチェンジの検証に送信するには、以下を実行できます。

```
from("rabbitmq:foo")
  .setHeader("rabbitmq.EXCHANGE_OVERRIDE_NAME", constant("cheese"))
  .to("rabbitmq:bar")
```

第257章 リアクティブストリームコンポーネント

Camel バージョン 2.19 から利用可能

reactive-streams: コンポーネントを使用すると、リアクティブストリーム標準と互換性のあるリアクティブストリーム処理ライブラリーを持つメッセージを交換できます。http://www.reactive-streams.org/

コンポーネントはバックプレシャーをサポートしており、リアクティブストリーム技術互換性キット (TCK) を使用してテストされました。

Camel モジュールは、ユーザーが Camel ルート内で受信および送信 ストリームを定義できるようにするリアクティブ ストリームコンポーネントと、Camel エンドポイントを外部のリアクティブフレームワークに直接使用できるようにするダイレクトクライアント API を提供します。

Camel はリアクティブストリームと サブスクライバー の内部実装を 使用するため、特定のフレームワークには関係ありません。統合テストでは、Reactor Core 3、RxJava 2 のリアクティブフレームワークが使用されています。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-reactive-streams</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

257.1. URI 形式

```
reactive-streams://stream?[options]
```

stream は、Camel ルートを外部ストリーム処理システムにバインドするために使用される論理ストリーム名です。

257.2. オプション

Reactive Streams コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
internalEngineConfiguration (advanced)	Reactive Streams の内部エンジンを設定します。		ReactiveStreamsEngineConfiguration
backpressureStrategy (producer)	低速なサブスクライバーにイベントをプッシュするときに使用するバックプレシャーストラテジー。	BUFFER	ReactiveStreamsBackpressureStrategy
serviceType (advanced)	使用する基礎となるリアクティブストリーム実装のタイプを設定します。この実装はレジストリーから検索されるか、または ServiceLoader を使用します。デフォルトの実装は DefaultCamelReactiveStreamsService です。		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Reactive Streams エンドポイントは、URI 構文を使用して設定します。

```
reactive-streams:stream
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

257.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
stream	メッセージの交換にエンドポイントによって使用されるストリームチャネルの名前。		文字列

257.2.2. クエリーパラメーター（10 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
concurrentConsumers (consumer)	Camel ルートでエクスチェンジを処理するために使用されるスレッドの数。	1	int
exchangesRefillLowWatermark (consumer)	maxInflightExchanges のパーセンテージで、要求されたエクスチェンジの低基準をアクティブなサブスクリプションに設定します。アップストリームソースからの保留中のアイテムの数が基準値よりも低い場合、新しい項目をサブスクリプションに要求できます。0 に設定すると、サブスクライバーは maxInflightExchanges のバッチでアイテムを要求します。これは、以前のバッチのすべての項目が処理された後にのみ行います。1 に設定すると、エクスチェンジが処理されるたびにサブスクライバーが新しいアイテムを要求できます(chatty)。任意の中間値を使用できます。	0.25	double
forwardOnComplete (consumer)	onComplete イベントを Camel ルートにプッシュすべきかどうかを決定します。	false	boolean
forwardOnError (consumer)	onError イベントを Camel ルートにプッシュする必要があるかどうかを決定します。例外はメッセージボディーとして設定されます。	false	boolean
maxInflightExchanges (consumer)	Camel によって同時に処理されるエクスチェンジの最大数。このパラメーターは、ストリームのバックプレシャーを制御します。正の値以外の値を設定すると、バックプレッシャーが無効になります。	128	整数
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
backpressureStrategy (producer)	低速なサブスクライバーにイベントをプッシュするときに使用するバックプレシャーストラテジー。		ReactiveStreamsBackpressureStrategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

257.3. 用途

ライブラリーは、アプリケーションが Camel データと対話するために必要なすべての通信モードをサポートすることを目的としています。

- *Camel ルートからデータを 取得 （Camel からのみ必要）*
- *Camel ルートにデータを 送信 する（Camel 専用）*
- *Camel ルートへの変換の リクエスト （Camel へのイン送信）*
- *リアクティブ処理ステップ（Camel からのインアウト）を使用して Camel ルートからデータフローを処理*

257.4. CAMEL からデータを取得

Camel ルートからデータフローにサブスクライブするには、以下のスニペットのように、エクスチェンジを名前付きストリームにリダイレクトする必要があります。

```
from("timer:clock")
.setBody().header(Exchange.TIMER_COUNTER)
.to("reactive-streams:numbers");
```

ルートは XML DSL を使用して作成することもできます。

この例では、無制限の数字のストリームが、名前番号に関連付けられます。このストリームは、CamelReactiveStreams ユーティリティークラスを使用してアクセスできます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// Getting a stream of exchanges
Publisher<Exchange> exchanges = camel.fromStream("numbers");

// Getting a stream of Integers (using Camel standard conversion system)
Publisher<Integer> numbers = camel.fromStream("numbers", Integer.class);
```

このストリームは、リアクティブストリームと互換性のあるライブラリーと簡単に使用できます。以下は、RxJava 2 で使用する方法の例です（ただし、リアクティブフレームワークを使用してイベントを処理することもできます）。

```
Flowable.fromPublisher(integers)
    .doOnNext(System.out::println)
    .subscribe();
```

この例では、Camel によって生成されたすべての数字を System.out に出力します。

257.4.1. direct API を使用した Camel からデータを取得

（Camel DSL を全く使用せずに）リアクティブフレームワークの機能構成を使用して処理フロー全体を定義することを希望する場合は、Camel URI を使用してストリームを定義することもできます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// Get a stream from all the files in a directory
Publisher<String> files = camel.from("file:folder", String.class);

// Use the stream in RxJava2
Flowable.fromPublisher(files)
    .doOnNext(System.out::println)
    .subscribe();
```

257.5. CAMEL へのデータ送信

外部ライブラリーがイベントを Camel ルートにプッシュする必要がある場合は、Reactive Streams エンドポイントはコンシューマーとして設定する必要があります。

```
from("reactive-streams:elements")
.to("log:INFO");
```

要素 ストリームへのハンドルは、CamelReactiveStreams ユーティリティークラスから取得されま

す。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

Subscriber<String> elements = camel.streamSubscriber("elements", String.class);
```

サブスクライバーは、要素 ストリームから消費する Camel ルートにイベントをプッシュするために使用できます。

以下は、RxJava 2 で使用する方法の例です（ただし、リアクティブフレームワークを使用してイベントを公開することもできます）。

```
Flowable.interval(1, TimeUnit.SECONDS)
    .map(i -> "Item " + i)
    .subscribe(elements);
```

文字列項目は、この例では RxJava によって毎秒生成され、上記で定義した Camel ルートにプッシュされます。

257.5.1. direct API を使用した Camel へのデータ送信

この場合、direct API を使用してエンドポイント URI から Camel サブスクライバーを取得できます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// Send two strings to the "seda:queue" endpoint
Flowable.just("hello", "world")
    .subscribe(camel.subscriber("seda:queue", String.class));
```

257.6. CAMEL への変換のリクエスト

一部の Camel DSL で定義されるルートは、リアクティブストリームフレームワーク内で使用して、特定の変換を実行できます（例： 同じメカニズムを使用してデータを http エンドポイントに送信し、続行します）。

以下のスニペットは、RxJava 関数コードが Camel に読み込みファイルおよびマーシャリングファイルのタスクをリクエストする方法を示しています。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);
```

```
// Process files starting from their names
Flowable.just(new File("file1.txt"), new File("file2.txt"))
    .flatMap(file -> camel.toStream("readAndMarshal", String.class))
    // Camel output will be converted to String
    // other steps
    .subscribe();
```

これを機能させるには、以下のようなルートを Camel コンテキストで定義する必要があります。

```
from("reactive-streams:readAndMarshal")
.marshal() // ... other details
```

257.6.1. direct API を使用した Camel への変換のリクエスト

代替アプローチは、URI エンドポイントを直接リアクティブフローで使用することです。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// Process files starting from their names
Flowable.just(new File("file1.txt"), new File("file2.txt"))
    .flatMap(file -> camel.to("direct:process", String.class))
    // Camel output will be converted to String
    // other steps
    .subscribe();
```

toStream の代わりに to（） メソッドを使用する場合は、"reactive-streams:" エンドポイントを使用してルートを定義する必要はありません（hood では使用されません）。

この場合、Camel 変換は以下を行います。

```
from("direct:process")
.marshal() // ... other details
```

257.7. CAMEL データをリアクティブフレームワークに処理

リアクティブストリームエクスポーターはデータを一方向で交換できますが、Camel ルートは多くの場合（REST エンドポイントを定義するなど）、リクエストごとに応答が必要な一般的には インアウト 交換パターンを使用します。

このような場合、ユーザーはフローにリアクティブ処理ステップを追加して、Camel ルートを強化したり、リアクティブフレームワークを使用して変換全体を定義したりできます。

たとえば、以下のルートがあるとします。

```
from("timer:clock")
.setBody().header(Exchange.TIMER_COUNTER)
.to("direct:reactive")
.log("Continue with Camel route... n=${body}");
```

リアクティブ処理ステップは、「direct:reactive」エンドポイントに関連付けることができます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

camel.process("direct:reactive", Integer.class, items ->
    Flowable.fromPublisher(items) // RxJava2
        .map(n -> -n)); // make every number negative
```

Camel ルートのデータフローは外部のリアクティブフレームワークによって処理され、Camel 内で処理フローを継続します。

このメカニズムを使用して、完全にリアクティブな方法で In-Out エクスチェンジを定義することもできます。

```
CamelReactiveStreamsService camel = CamelReactiveStreams.get(context);

// requires a rest-capable Camel component
camel.process("rest:get:orders", exchange ->
                Flowable.fromPublisher(exchange)
                        .flatMap(ex -> allOrders())); // retrieve orders asynchronously
```

詳細は、Camel の例(camel-example-reactive-streams)を参照してください。

257.8. 高度なトピック

257.8.1. Backpressure(producer side)の制御

Camel エクスチェンジを外部サブスクライバーにルーティングする場合、バックプレッシャーはエクスチェンジの配信前にエクスチェンジをキャッシュする内部バッファーによって処理されます。サブスクライバーがエクスチェンジレートよりも遅い場合、バッファーが大きすぎる可能性があります。多くの状況では、これを回避する必要があります。

以下のルートを考慮してください。

```
from("jms:queue")
.to("reactive-streams:flow");
```

JMS キューには多数のメッセージが含まれ、フロー ストリームに関連するサブスクライバーが遅すぎると、メッセージは JMS からキューになり、バッファーに追加され、「out of memory」エラーが発生する可能性があります。このような問題を回避するには、ThrottlingInflightRoutePolicy をルートに設定できます。

```
ThrottlingInflightRoutePolicy policy = new ThrottlingInflightRoutePolicy();
policy.setMaxInflightExchanges(10);

from("jms:queue")
.routePolicy(policy)
.to("reactive-streams:flow");
```

このポリシーは、アクティブなエクスチェンジの最大数（バッファーの最大サイズなど）を制限することで、しきい値（例では10 ）よりも小さくなります。10 を超えるメッセージが進行中の場合、ルートは一時停止され、サブスクライバーがメッセージを処理するのを待機します。

このメカニズムにより、サブスクライバーはバックプレッシャーを介してルートの一時停止/再開を自動的に制御します。複数のサブスクライバーが同じストリームからアイテムを消費すると、最も遅くなります。

他の状況では、http コンシューマーを使用する場合などで、ルートは http サービスを利用不可にするので、デフォルト設定（ポリシーなし、バインドされていないバッファーなし）の使用が推奨されます。http サービスへの要求数を制限することにより、メモリーの問題の発生を防ぐ必要があります（例： スケールアウト）。

一定量のデータ損失が許可される場合、BUFFER 以外のバックプレシャーストラテジーを設定することは、高速ソースを処理するためのソリューションとなります。

```
from("direct:thermostat")
.to("reactive-streams:flow?backpressureStrategy=LATEST");
```

LATEST バックプレシャーストラテジーが使用される場合、ルートから受け取った最後のエクスチェンジのみがパブリッシャーによって保持され、古いデータは破棄されます（他のオプションも利用可能です）。

257.8.2. Backpressure（コンシューマー側）の制御

Camel がリアクティブストリームパブリッシャーからのアイテムを消費すると、インフライトエクスチェンジの最大数をエンドポイントオプションとして設定できます。

コンシューマーに関連付けられたサブスクライバーはパブリッシャーと対話し、ルート内のメッセージ数はしきい値を下回るようにします。

backpressure-aware ルートの例：

```
from("reactive-streams:numbers?maxInflightExchanges=10")
.to("direct:endpoint");
```

（リアクティブストリームバックプレシャーメカニズムを介して）Camel がソースパブリッシャーにリクエストするアイテムの数は、常に 10 未満です。メッセージは Camel 側の単一スレッドによって処理されます。

同時コンシューマー(threads)の数はエンドポイントオプション(concurrentConsumer)として設定することもできます。1 つのコンシューマー（デフォルト）を使用する場合、ソースストリームの項目の順序が維持されます。この値を増やすと、アイテムは複数のスレッドによって同時に処理されます（つまり、順番は保持されません）。

257.9. CAMEL REACTIVE STREAMS STARTER

spring-boot ユーザーはスターターモジュールを利用できます。スターターを使用する場合は、CamelReactiveStreamsService を直接 Spring コンポーネントに注入することができます。

スターターを使用するには、以下を Spring ブートの pom.xml ファイルに追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-reactive-streams-starter</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version --
>
</dependency>
```

257.10. 関連項目

- Configuring Camel (Camel の設定)

- *コンポーネント*
- *エンドポイント*
- *はじめに*

第258章 REACTOR コンポーネント

Camel バージョン 2.20 で利用可能

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-reactor</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第259章 REF コンポーネント

Camel バージョン 1.2 で利用可能

ref: コンポーネントはレジストリーにバインドされている既存のエンドポイントを検索するために使用されます。

259.1. URI 形式

```
ref:someName[?options]
```

someName はレジストリーのエンドポイントの名前です（通常は Spring レジストリーではなく）。Spring レジストリーを使用している場合、someName は Spring レジストリーのエンドポイントの Bean ID になります。

259.2. REF オプション

Ref コンポーネントにはオプションがありません。

Ref エンドポイントは、URI 構文を使用して設定します。

```
ref:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

259.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	レジストリーで検索するエンドポイントの名前が **必要です。**		文字列

259.2.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

259.3. ランタイム検索

このコンポーネントは、実行時に URI を計算できるレジストリーでエンドポイントの動的検出が必要な場合に使用できます。次に、以下のコードを使用してエンドポイントを検索できます。

```
// lookup the endpoint
String myEndpointRef = "bigspenderOrder";
Endpoint endpoint = context.getEndpoint("ref:" + myEndpointRef);

Producer producer = endpoint.createProducer();
Exchange exchange = producer.createExchange();
exchange.getIn().setBody(payloadToSend);
// send the exchange
producer.process(exchange);
```

また、以下のようなレジストリー内にエンドポイントの一覧を定義することができます。

```
<camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring">
    <endpoint id="normalOrder" uri="activemq:order.slow"/>
    <endpoint id="bigspenderOrder" uri="activemq:order.high"/>
```

```
</camelContext>
```

259.4. 例

以下の例では、URI の ref: を使用して spring ID の endpoint2 でエンドポイントを参照します。

当然ながら、代わりに ref 属性を使用しています。

```
<to ref="endpoint2"/>
```

これは、書くより一般的な方法です。

第260章 REST コンポーネント

Camel バージョン 2.14 から利用可能

rest コンポーネントでは、Rest DSL およびプラグインを使用して REST エンドポイント（コンシューマー）を REST トランスポートとして他の Camel コンポーネントに定義できます。

Camel 2.18 以降では、REST サービスをクライアント（プロデューサー）として使用して REST サービスを呼び出すこともできます。

260.1. URI 形式

```
rest://method:path[:uriTemplate]?[options]
```

260.2. URI オプション

REST コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
componentName (common)	REST トランスポートに使用する Camel Rest コンポーネント（restlet、spark-rest など）。明示的に構成されたコンポーネントがない場合、または Rest DSL と統合する Camel コンポーネントがある場合、または org.apache.camel.spi.RestConsumerFactory(consumer)または org.apache.camel.spi.RestProducerFactory(producer)がレジストリーに登録されている場合、Camel は検索を行います。いずれかのファイルが見つかると、それが使用されます。		文字列
apiDoc (producer)	使用する swagger api doc リソース。リソースはデフォルトでクラスパスからロードされ、JSon 形式である必要があります。		文字列
ホスト （プロデューサー）	使用する HTTP サービスのホストおよびポート（swagger スキーマのホストはオーバーライド）		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

REST エンドポイントは、URI 構文を使用して設定されます。

```
rest:method:path:uriTemplate
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

260.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
method	使用する ***必須*** の HTTP メソッド。		文字列
path	***必須*** のベースパス		文字列
uriTemplate	uri テンプレート		文字列

260.2.2. クエリーパラメーター（15 パラメーター）：

Name	説明	デフォルト	Type
componentName (common)	REST トランスポートに使用する Camel Rest コンポーネント（restlet、spark-rest など）。明示的にコンポーネントが設定されていない場合、Camel は Rest DSL と統合する Camel コンポーネントがある場合や、org.apache.camel.spi.RestConsumerFactory がレジストリーに登録されている場合は検索します。いずれかのファイルが見つかると、それが使用されます。		文字列
消費 （共通）	'text/xml' などのメディアタイプ、または 'application/json' などのメディアタイプは、この REST サービスが受け入れます。デフォルトでは、あらゆる種類のタイプを受け入れます。		文字列

Name	説明	デフォルト	Type
inType (common)	受信 POJO バインディングタイプを FQN クラス名として宣言するには、以下を実行します。		文字列
outType (common)	発信 POJO バインディングタイプを FQN クラス名として宣言するには、以下を実行します。		文字列
生成 （共通）	'text/xml' などのメディアタイプ、または 'application/json' などのメディアタイプは、この REST サービスを返します。		文字列
routeId (common)	この REST サービスにより作成されるルートの名前		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
Description (consumer)	この REST サービスを文書化するための人的説明		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
apiDoc (producer)	使用する swagger api doc リソース。リソースはデフォルトでクラスパスからロードされ、JSon 形式である必要があります。		文字列
bindingMode (producer)	プロデューサーのバインディングモードを設定します。'off' 以外のものに設定されている場合、プロデューサーは受信メッセージのボディーを inType から json または xml に、応答を json または xml から outType に変換しようとします。		RestBindingMode
ホスト （プロデューサー）	使用する HTTP サービスのホストおよびポート（swagger スキーマのホストはオーバーライド）		文字列

Name	説明	デフォルト	Type
queryParameters (producer)	呼び出しする HTTP サービスのクエリーパラメーター		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

260.3. サポートされる REST コンポーネント

以下のコンポーネントは、REST コンシューマー(Rest DSL)をサポートします。

- *camel-coap*
- *camel-netty-http*
- *camel-netty4-http*
- *camel-jetty*
- *camel-restlet*
- *camel-servlet*
- *camel-spark-rest*
- *camel-undertow*

以下のコンポーネントは、REST プロデューサーをサポートします。

- *camel-http*
- *camel-http4*
- *camel-netty4-http*
- *camel-jetty*
- *camel-restlet*
- *camel-undertow*

260.4. パスおよび URITEMPLATE 構文

path および uriTemplate オプションは、パラメーターのサポートを使用して REST コンテキストパスを定義する REST 構文で定義します。

TIP: uriTemplate が設定されていないと、path オプションは同じように機能します。パスのみを設定したり、両方のオプションを設定した場合も問題ありません。REST では、パスと uriTemplate の両方を設定するのが一般的な方法です。

以下は、パスのみを使用する Camel ルートです。

```
from("rest:get:hello")
  .transform().constant("Bye World");
```

以下のルートは、キー「me」を持つ Camel ヘッダーにマッピングされるパラメーターを使用します。

```
from("rest:get:hello/{me}")
  .transform().simple("Bye ${header.me}");
```

以下の例では、ベースパスを「hello」として設定し、uriTemplates を使用して 2 つの REST サービスを設定しています。

```
from("rest:get:hello:/{me}")
  .transform().simple("Hi ${header.me}");

from("rest:get:hello:/french/{me}")
  .transform().simple("Bonjour ${header.me}");
```

260.5. REST プロデューサーの例

rest コンポーネントを使用して、他の Camel コンポーネントと同様に REST サービスを呼び出すことができます。

たとえば、hello/{me} を使用して REST サービスを呼び出すには、以下を実行できます。

```
from("direct:start")
  .to("rest:get:hello/{me}");
```

次に、動的な値 {me} は、同じ名前を持つ Camel メッセージにマッピングされます。そのため、この REST サービスを呼び出すには、以下のように空のメッセージボディーとヘッダーを送信できます。

```
template.sendBodyAndHeader("direct:start", null, "me", "Donald Duck");
```

Rest プロデューサーは REST サービスのホスト名とポートを把握しておく必要があります。これは、以下のように host オプションを使用して設定できます。

```
from("direct:start")
  .to("rest:get:hello/{me}?host=myserver:8080/foo");
```

host オプションを使用する代わりに、以下のように restConfiguration でホストを設定できます。

```
restConfiguration().host("myserver:8080/foo");

from("direct:start")
  .to("rest:get:hello/{me}");
```

producerComponent を使用して、HTTP クライアントとして使用する Camel コンポーネントを選択できます。たとえば、http4 を使用できます。

```
restConfiguration().host("myserver:8080/foo").producerComponent("http4");
```

```
from("direct:start")
  .to("rest:get:hello/{me}");
```

260.6. REST プロデューサーバインディング

REST プロデューサーは、JSon または rest-dsl のように XML を使用したバインディングをサポートします。

たとえば、json バインディングモードを有効にして jetty を使用するには、rest 設定で設定できます。

```
restConfiguration().component("jetty").host("localhost").port(8080).bindingMode(RestBinding
Mode.json);

from("direct:start")
  .to("rest:post:user");
```

次に、REST プロデューサーを使用して REST サービスを呼び出すと、REST サービスを呼び出す前に POJO を json に自動的にバインドします。

```
UserPojo user = new UserPojo();
user.setId(123);
user.setName("Donald Duck");

template.sendBody("direct:start", user);
```

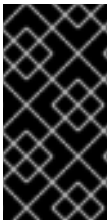
上記の例では、POJO インスタンスの UserPojo をメッセージボディーとして送信します。そして、REST設定で JSon バインディングをオンにしたので、REST サービスを呼び出す前に POJO は POJO から JSon にマーシャルされます。

ただし、レスポンスメッセージのバインディングも実行したい場合は（REST サービスが応答として返されるものなど）、JSon から POJO にアンマーシャリングする POJO のクラス名を指定するよう outType オプションを設定する必要があります。

たとえば、REST サービスが com.foo.MyResponsePojo にバインドされる JSon ペイロードを返す場合は、以下のように設定できます。

```
restConfiguration().component("jetty").host("localhost").port(8080).bindingMode(RestBinding
Mode.json);
```

```
from("direct:start")
  .to("rest:post:user?outType=com.foo.MyResponsePojo");
```



重要

REST サービスを呼び出す応答メッセージに対して POJO バインディングが発生した場合は、outType オプションを設定する必要があります。

260.7. その他の例

その他の例と、Rest DSL を使用してそれらを優れた RESTful 方法で定義する方法は、「Rest DSL」を参照してください。

Apache Camel ディストリビューションには camel-example-servlet-rest-tomcat のサンプルがあります。これは、SERVLET で Rest DSL を Apache Tomcat にデプロイするトランスポートとして、または同様の Web コンテナーにデプロイする方法を実証します。

260.8. 関連項目

- REST DSL
- SERVLET

第261章 REST SWAGGER コンポーネント

Camel バージョン 2.19 から利用可能

rest-swagger は、Swagger (Open API)仕様ドキュメントから REST プロデューサーを設定し、RestProducerFactory インターフェースを実装するコンポーネントに委譲します。現在既知の作業コンポーネントは以下の通りです。

- http
- http4
- netty4-http
- restlet
- jetty
- undertow

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rest-swagger</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

261.1. URI 形式

```
rest-swagger:[specificationPath#]operationId
```

operationId は Swagger 仕様の操作の ID で、spec Path は仕様へ のパスになります。specificationPath が指定されていない場合、デフォルトは swagger.json です。ルックアップメ

カニズムは Camels ResourceHelper を使用してリソースをロードします。つまり、CLASSPATH リソース(classpath:my-specification.json)、ファイル(file:/some/path.json)、web(http://api.example.com/swagger.json)、または Bean(ref:nameOfBean)を参照したり、Bean(bean:nameOfBean.methodName)のメソッドを使用して仕様リソースを取得し、Swagger の独自のリソースロードサポートが失敗します。

このコンポーネントは HTTP クライアントとして機能せず、上記の他のコンポーネントに委任されます。ルックアップメカニズムは、RestProducerFactory インターフェースを実装する単一のコンポーネントを検索し、それを使用します。CLASSPATH に複数のものが含まれる場合、プロパティー componentName は委譲するコンポーネントを示すように設定する必要があります。

ほとんどの設定は Swagger 仕様から取得されますが、コンポーネントまたはエンドポイントで設定を指定して上書きするオプションが存在します。通常、host または basePath は、これらが仕様とは異なる場合に上書きするだけで済みます。

注記

host パラメーターには、スキーム、ホスト名、ポート番号が含まれる絶対 URI が含まれる必要があります（例： https://api.example.com）。

componentName では、要求の実行に使用するコンポーネントを指定します。

コンポーネントまたはエンドポイントレベルで componentName を指定しない場合、CLASSPATH は適切な委譲を検索します。これを機能させるには、RestProducerFactory インターフェースを実装する CLASSPATH に 1 つのコンポーネントのみが存在する必要があります。

このコンポーネントのエンドポイント URI は lenient です。つまり、メッセージヘッダーに加えて、エンドポイントパラメーターとして REST 操作のパラメーターを指定できます。これらは、後続のすべての呼び出しに対して定数されるため、/api/7.1/users/{id} など、パスにあるすべての呼び出しの定数がないパラメーターにのみ、この機能を使用することが理にかなっています。

261.2. オプション

REST Swagger コンポーネントは、以下に示す 7 つのオプションをサポートします。

Name	説明	デフォルト	Type
basePath (producer)	API basePath（例： /v2）デフォルトは未設定です。設定しないと、Swagger 仕様に存在する値が上書きされます。		文字列
componentName (producer)	リクエストを実行する Camel コンポーネントの名前。compnent は Camel レジストリーに存在し、RestProducerFactory サービスプロバイダーインターフェースを実装する必要があります。設定されていない場合は、CLASSPATH が RestProducerFactory SPI を実装する単一のコンポーネントを検索します。エンドポイント設定で上書きできます。		文字列
消費 （プロデューサー）	このコンポーネントが消費できるペイロードタイプ。application/json や複数のタイプの application/json、application/xml、q=0.5 など、RFC7231 に従って 1 つのタイプを使用できます。これは Accept HTTP ヘッダーの値と同じです。設定すると、Swagger 仕様で見つかった値が上書きされます。エンドポイント設定で上書き可能		文字列
ホスト （プロデューサー）	HTTP 要求を https://hostname:port の形式で転送するスキームホスト名およびポート。エンドポイント、コンポーネント、または Camel Context の correspoding REST 設定で設定できます。このコンポーネントに複数の名前を付けた場合（例： petstore）、REST 設定が最初に参照され、次に rest-swagger およびグローバル設定が最後に確認されます。設定すると、Swagger 仕様で見つかった値が上書きされる場合、RestConfiguration になります。エンドポイント設定で上書きできます。		文字列
生成 （プロデューサー）	このコンポーネントによって生成されるペイロードタイプ。たとえば、RFC7231 に準拠する application/json などです。これは Content-Type HTTP ヘッダーの値と同じです。設定すると、Swagger 仕様に存在する値が上書きされます。エンドポイント設定で上書きできます。		文字列

Name	説明	デフォルト	Type
specificationUri (producer)	Swagger 仕様ファイルへのパス。スキームのホスト、ベースパスはこの仕様から取得されますが、コンポーネントまたはエンドポイントレベルのプロパティーで上書きできます。指定のない場合、コンポーネントは swagger.json リソースを読み込もうとします。このコンポーネントのコンポーネントおよびエンドポイントで定義されるホストには、スキーム、ホスト名、およびオプションで URI 構文のポートを含める必要があります（https://api.example.com:8080など）。エンドポイント設定で上書きできます。	swagger.json	URI
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

REST Swagger エンドポイントは、URI 構文を使用して設定します。

```
rest-swagger:specificationUri#operationId
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

261.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
specificationUri	Swagger 仕様ファイルへのパス。スキームのホスト、ベースパスはこの仕様から取得されますが、コンポーネントまたはエンドポイントレベルのプロパティーで上書きできます。指定のない場合、コンポーネントは swagger.json リソースを読み込もうとします。このコンポーネントのコンポーネントおよびエンドポイントで定義されるホストには、スキーム、ホスト名、およびオプションで URI 構文のポートを含める必要があります（https://api.example.com:8080など）。コンポーネントの設定を上書きします。	swagger.json	URI
operationId	Swagger 仕様からの操作に **必要な** ID。		文字列

261.2.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
basePath (producer)	API basePath（例： /v2）デフォルトは未設定です。設定しないと、Swagger 仕様およびコンポーネント設定の値が上書きされます。		文字列
componentName (producer)	リクエストを実行する Camel コンポーネントの名前。compnent は Camel レジストリーに存在し、RestProducerFactory サービスプロバイダーインターフェースを実装する必要があります。設定されていない場合は、CLASSPATH が RestProducerFactory SPI を実装する単一のコンポーネントを検索します。コンポーネントの設定を上書きします。		文字列
消費 （プロデューサー）	このコンポーネントが消費できるペイロードタイプ。application/json や複数のタイプの application/json、application/xml、q=0.5 など、RFC7231 に従って 1 つのタイプを使用できます。これは Accept HTTP ヘッダーの値と同じです。設定すると、Swagger 仕様で見つかった値や、コンポーネントの設定に含まれる値が上書きされます。		文字列
ホスト （プロデューサー）	HTTP 要求を https://hostname:port の形式で転送するスキームホスト名およびポート。エンドポイント、コンポーネント、または Camel Context の correspoding REST 設定で設定できます。このコンポーネントに複数の名前を付けた場合（例： petstore）、REST 設定が最初に参照され、次に rest-swagger およびグローバル設定が最後に確認されます。設定すると、Swagger 仕様で見つかった値が上書きされる場合、RestConfiguration になります。その他のすべての設定を上書きします。		文字列
生成 （プロデューサー）	このコンポーネントによって生成されるペイロードタイプ。たとえば、RFC7231 に準拠する application/json などです。これは Content-Type HTTP ヘッダーの値と同じです。設定すると、Swagger 仕様に存在する値が上書きされます。その他のすべての設定を上書きします。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

261.3. 例： PETSTORE

examples ディレクトリーの camel-example-rest-swagger プロジェクトの例をチェックアウトします。

たとえば、PetStore が提供する REST API は単に仕様 URI と、Swagger 仕様から必要な操作 ID を参照する場合や、仕様をダウンロードして、自動的に使用されるように CLASSPATH の swagger.json として保存します。Undertow コンポーネントを使用してすべてのリクエストを実行し、Spring Boot に対して Camels excelent サポートを実行してみましょう。

以下は、Maven POM ファイルで定義される依存関係です。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-undertow-starter</artifactId>
</dependency>

<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rest-swagger-starter</artifactId>
</dependency>
```

Undertow コンポーネントと RestSwaggerComponent を定義して開始します。

```
@Bean
public Component petstore(CamelContext camelContext, UndertowComponent undertow) {
    RestSwaggerComponent petstore = new RestSwaggerComponent(camelContext);
    petstore.setSpecificationUri("http://petstore.swagger.io/v2/swagger.json");
    petstore.setDelegate(undertow);

    return petstore;
}
```

注記

Spring Boot の Camel のサポートは UndertowComponent Spring Bean を自動作成し、接頭辞 camel.component.undertow. を使用して application.properties （または application.yml）を使用して設定できます。ここで petstore コンポーネントを定義し、Camel コンテキストに名前付きコンポーネントを定義して、PetStore REST API との対話に使用できます。これが使用する唯一の rest-swagger コンポーネントである場合（ application.propertiesを使用して）同じ方法で設定できます。

これで、ProducerTemplate を使用して PetStore REST メソッドを呼び出すだけです。

```
@Autowired
ProducerTemplate template;
```

```
String getPetJsonById(int petId) {
    return template.requestBodyAndHeaders("petstore:getPetById", null, "petId", petId);
}
```

第262章 RESTLET コンポーネント

Camel バージョン 2.0 で利用可能

Restlet コンポーネントは、RESTful リソースを消費および生成するための Restlet ベースのエンドポイントを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-restlet</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

262.1. URI 形式

```
restlet:restletUrl[?options]
```

restletUrl の形式：

```
protocol://hostname[:port][/resourcePattern]
```

restlet は、プロトコルやアプリケーションの懸念の分離をプロモートします。Restlet Engine の参照実装は、複数のプロトコルをサポートします。ただし、HTTP プロトコルのみをテストしました。デフォルトのポートはポート 80 です。現在、プロトコルに基づいてデフォルトのポートを自動的に切り替えることはありません。

URI にクエリーオプションを追加するには、?option=value&option=value&…

INFO: It seems Restlet is case sensitive in understanding headers.たとえば、コンテンツタイプを使用するには Content-Type を使用し、ロケーションに Location などを使用します。

警告

Camel 2.14.0 および 2.14.1 の camel-restlet でパフォーマンスに関するレポートを受けました。996 の問題の問題でこれを Restlet チームに報告し ました。問題に対応するには、Camel 2.14.2 以降から、エンドポイント URI で Sync=true をオプションとして設定するか、または RestletComponent にこれをグローバルオプションとして設定し、すべてのエンドポイントがこのオプションを継承します。

262.2. オプション

Restlet コンポーネントは、以下に記載されている 22 オプションをサポートします。

Name	説明	デフォルト	Type
controllerDaemon (consumer)	コントローラースレッドがデーモンである必要があるかどうかを示します（JVM の終了をブロックしません）。		ブール値
controllerSleepTimeMs (consumer)	コントローラースレッドが各制御間でスリープ状態になる時間。		整数
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
inboundBufferSize (consumer)	メッセージの読み取り時のバッファーサイズ		整数
maxConnectionsPerHost (common)	ホスト（IP アドレス）ごとの同時接続の最大数。		整数
maxThreads （コンシューマー）	リクエストに対応する最大スレッド。		整数
lowThreads (consumer)	コネクターがオーバーロードとみなされる場合を判別するワーカースレッドの数。		整数
maxTotalConnections (common)	合計同時接続の最大数。		整数

Name	説明	デフォルト	Type
minThreads （コンシューマー）	要求処理を待機している最小スレッド。		整数
outboundBufferSize (consumer)	メッセージの書き込み時のバッファーサイズ		整数
persistingConnections (consumer)	呼び出し後に接続が有効な状態であるかどうかを示します。		ブール値
pipeliningConnections (consumer)	パイプライン接続がサポートされるかどうかを示します。		ブール値
threadMaxIdleTimeMs (consumer)	アイドルスレッドが操作を取得するまで待機する時間。		整数
useForwardedForHeader (consumer)	一般的なプロキシーおよびキャッシュでサポートされる X-Forwarded-For ヘッダーを検索し、これを使用して Request.getClientAddresses（）メソッドの結果を設定します。この情報は、ローカルネットワーク内の中間コンポーネントにのみ安全です。他のアドレスはファクヘッダーを設定して簡単に変更できます。また、深刻なセキュリティーチェックについては信頼できません。		ブール値
reuseAddress (consumer)	SO_REUSEADDR ソケットオプションを有効化/無効化します。詳細は、java.io.ServerSocketreuseAddress プロパティーを参照してください。		ブール値
maxQueued (consumer)	サービスに使用できるワーカースレッドがない場合にキューに配置することができる呼び出しの最大数。値が '0' の場合、キューが使用されず、ワーカースレッドがすぐに利用できない場合に呼び出しが拒否されます。値が「-1」の場合には、バインドされていないキューが使用され、呼び出しは拒否されません。		整数

Name	説明	デフォルト	Type
disableStreamCache (consumer)	Restlet からの raw 入力ストリームがキャッシュされているかどうかを決定します（Camel はストリームをメモリー内/オーバーフロー内のファイル、ストリームキャッシング）キャッシュに読み取ります。デフォルトでは、Camel は Restlet 入力ストリームをキャッシュし、Camel がストリームからすべてのデータを取得できるようにするために複数回読み取りをサポートします。ただし、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。たとえば、ファイルまたは他の永続ストアに直接ストリーミングする場合などに、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。DefaultRestletBinding は、ストリームの読み取りを複数回サポートするために、このオプションが false の場合は、リクエスト入力ストリームをストリームキャッシュにコピーし、メッセージボディーに配置します。	false	boolean
ポート （コンシューマー）	restlet コンシューマールートのポート番号を設定します。これにより、これらのコンシューマーに同じポートを再利用するように一度設定できます。		int
同期 （プロデューサー）	プロデューサーに同期 Restlet Client を使用するかどうか。このオプションを true に設定すると、Restlet 同期クライアントのパフォーマンスが向上するため、パフォーマンスが向上します。		ブール値
enabledConverters (advanced)	完全なクラス名または単純なクラス名として有効にするコンバーターの一覧。自動的に登録されるコンバーターはすべて、空または null の場合有効になります。		リスト
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Restlet エンドポイントは URI 構文を使用して設定します。

```
restlet:protocol:host:port/uriPattern
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

262.2.1. パスパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
protocol	http または https を使用するプロトコルが **必要** です。		文字列
host	***必須***: restlet サービスのホスト名		文字列
port	***必須***: restlet サービスのポート番号	80	int
uriPattern	/customer/id などのリソースパターン		文字列

262.2.2. クエリーパラメーター（18 パラメーター）：

Name	説明	デフォルト	Type
restletMethod (common)	プロデューサーエンドポイントで、使用するリクエストメソッドを指定します。コンシューマーエンドポイントでは、エンドポイントは restletMethod リクエストのみを消費するように指定します。	GET	メソッド
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
restletMethods (consumer)	restlet コンシューマーエンドポイントによってサービスされるように、コンマで区切られた 1 つ以上のメソッド（restletMethods=post,put など）を指定します。restletMethod と restletMethods オプションの両方を指定すると、restletMethod 設定は無視されます。可能なメソッドは ALL、CONNECT、DELETE、GET、HEAD、OPTIONS、PATCH、POST、PUT、TRACE です。		文字列

Name	説明	デフォルト	Type
disableStreamCache (consumer)	Restlet からの raw 入力ストリームがキャッシュされているかどうかを決定します（Camel はストリームをメモリー内/オーバーフロー内のファイル、ストリームキャッシング）キャッシュに読み取ります。デフォルトでは、Camel は Restlet 入力ストリームをキャッシュし、Camel がストリームからすべてのデータを取得できるようにするために複数回読み取りをサポートします。ただし、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。たとえば、ファイルまたは他の永続ストアに直接ストリーミングする場合などに、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。DefaultRestletBinding は、ストリームの読み取りを複数回サポートするために、このオプションが false の場合は、リクエスト入力ストリームをストリームキャッシュにコピーし、メッセージボディーに配置します。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
restletUriPatterns (consumer)	Camel レジストリーで List を参照するため表記を使用して、RESTlet コンシューマーエンドポイントによって提供される 1 つ以上の URI テンプレートを指定します。**URI パターンがエンドポイント URI に定義されている場合、エンドポイントに定義された URI パターンと restletUriPatterns オプションの両方が適用されます。**		リスト
connectTimeout (producer)	接続がタイムアウトすると、クライアントは接続を切断します。無制限の待機時間は 0 になります。	30000	int
cookieHandler (producer)	HTTP セッションを維持するためのクッキーハンドラーの設定		CookieHandler
socketTimeout (producer)	クライアントソケットの受信タイムアウト。無制限の待機時間は 0 です。	30000	int

Name	説明	デフォルト	Type
throwExceptionOnFailure (producer)	プロデューサーの失敗で例外をスローするかどうか。このオプションが false の場合、http ステータスコードはメッセージヘッダーとして設定され、error の値がある場合にチェックできます。	true	boolean
autoCloseStream (producer)	restlet プロデューサーを使用して REST サービスを呼び出す応答としてストリーム表現を自動的に閉じるかどうか。応答がストリーミングされ、streamRepresentation オプションが有効になっている場合は、ストリーミング応答から InputStream を自動的に閉じ、Camel Exchange がルーティングされたときに入力ストリームが閉じられるようにすることができます。ただし、Camel ルート外でストリームを読み取る必要がある場合は、ストリームを自動閉じる必要がない場合があります。	false	boolean
streamRepresentation (producer)	restlet プロデューサーを使用して REST サービスを呼び出す応答としてストリーム表現をサポートするかどうか。応答がストリーミングされる場合、このオプションを有効にして java.io.InputStream を Camel Message ボディーのメッセージボディーとして使用できます。このオプションを使用すると、autoCloseStream オプションを有効にし、Camel Exchange がルーティングされたときに入力ストリームが閉じられるようにすることができます。ただし、Camel ルート外でストリームを読み取る必要がある場合は、ストリームを自動閉じる必要がない場合があります。	false	boolean
headerFilterStrategy (advanced)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
restletBinding (advanced)	カスタム RestletBinding を使用して Restlet と Camel メッセージの間でバインドする場合。		RestletBinding
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
restletRealm (security)	restlet のセキュリティーレルムをマップとして設定します。		マップ
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定します。		SSLContextParameters

262.3. メッセージヘッダー

Name	タイプ	説明
Content-Type	**文字列**	アプリケーション/プロセッサーによって OUT メッセージに設定できるコンテンツタイプを指定します。値は、レスポンスメッセージの **コンテンツタイプ** です。このヘッダーが設定されていない場合、コンテンツタイプは OUT メッセージボディーのオブジェクトタイプに基づいています。Camel 2.3 以降では、Content-Type ヘッダーが Camel IN メッセージに指定される場合、ヘッダーの値は Restlet リクエストメッセージのコンテンツタイプを決定します。 それ以外の場合、デフォルトは「application/x-www-form-urlencoded」になります。2.3 よりも前のリリースでは、要求コンテンツタイプのデフォルトを変更することはできません。
CamelAcceptContentType	**文字列**	**Camel 2.9.3 以降 2.10.0:** HTTP Accept リクエストヘッダー。
CamelHttpMethod	**文字列**	HTTP 要求メソッド。これは IN メッセージヘッダーで設定されます。
CamelHttpQuery	**文字列**	リクエスト URI のクエリー文字列。これは、restlet コンポーネントがリクエストを受信する際に **DefaultRestletBinding** によって IN メッセージに設定されます。
CamelHttpResponseCode	**文字列** または **Integer**	応答コードは、アプリケーション/プロセッサーにより OUT メッセージに設定できます。値は、レスポンスメッセージのレスポンスコードです。このヘッダーが設定されていない場合、応答コードは restlet ランタイムエンジンによって設定されます。
CamelHttpUri	**文字列**	HTTP 要求 URI。これは IN メッセージヘッダーで設定されます。
CamelRestletLogin	**文字列**	Basic 認証のログイン名。これはアプリケーションによって IN メッセージに設定され、Camel による restlet リクエストヘッダーの前にフィルターされます。
CamelRestletPassword	**文字列**	Basic 認証のパスワード名。これはアプリケーションによって IN メッセージに設定され、Camel による restlet リクエストヘッダーの前にフィルターされます。
CamelRestletRequest	**要求**	**camel** 2.8: すべての要求詳細を保持する **org.restlet.Request** オブジェクト。

Name	タイプ	説明
CamelRestletResponse	**Response**	camel 2.8: **org.restlet.Response** オブジェクト。これを使用して、Restlet からの API を使用して応答を作成できます。以下の例を参照してください。
org.restlet.*		Camel IN ヘッダーに伝播される Restlet メッセージの属性。
cache-control	**文字列** または **リスト<CacheDirective>**	**Camel 2.11:** ユーザーは cache-control を String 値または Camel メッセージヘッダーから CacheDirective of Restlet の一覧で設定できます。

262.4. メッセージボディー

Camel は外部サーバーからの restlet 応答を OUT ボディーに保存します。IN メッセージからのヘッダーはすべて OUT メッセージにコピーされ、ルーティング時にヘッダーが保持されます。

262.5. サンプル

262.5.1. 認証での restlet エンドポイント

以下のルートは、http://localhost:8080 の POST 要求をリッスンする restlet コンシューマーエンドポイントを開始します。プロセッサーは、リクエストの本文と id ヘッダーの値を出力する応答を作成します。

URI クエリーの restletRealm 設定は、レジストリーでレルムマップを検索するために使用されます。このオプションを指定すると、restlet コンシューマーは情報を使用してユーザーログインを認証します。認証された 要求のみがリソースにアクセスできます。この例では、レジストリーとして機能する Spring アプリケーションコンテキストを作成します。Realm Map の Bean ID は **restletRealmRef** *と一致する必要があります。*

以下の例は、http://localhost:8080 （restlet コンシューマーエンドポイント）のサーバーにリクエストを送信する direct エンドポイントを開始します。

これはすべて必要です。リクエストを送信し、restlet コンポーネントを試す準備が整いました。

サンプルクライアントは、以下のヘッダーを使用して direct:start-auth エンドポイントにリクエストを送信します。

- CamelRestletLogin （Camel によって内部で使用される）
- CamelRestletPassword （Camel によって内部で使用される）
- id （アプリケーションヘッダー）

注記

org.apache.camel.restlet.auth.login および org.apache.camel.restlet.auth.password は Restlet ヘッダーとして伝播されません。

サンプルクライアントは以下のような応答を取得します。

```
received [<order foo='1'/>] as an order id = 89531
```

262.5.2. 複数のメソッドおよび URI テンプレートを提供する単一の restlet エンドポイント（非推奨）

この機能は 非推奨 にされており、使用しないようにしてください。

restletMethods オプションを使用して、複数の HTTP メソッドにサービスを提供する 1 つのルートを作成できます。このスニペットは、ヘッダーからリクエストメソッドを取得する方法も示しています。

複数のメソッドを提供するだけでなく、次のスニペットでは、restletUriPatterns オプションを使用して複数の URI テンプレートをサポートするエンドポイントを作成する方法を示します。リクエスト URI は IN メッセージのヘッダーでも利用可能です。URI パターンがエンドポイント URI に定義されている場合（このサンプルではない場合）、エンドポイントに定義された URI パターンと restletUriPatterns オプションの両方が適用されます。

restletUriPatterns=#uriTemplates オプションは、Spring XML 設定で定義され た List<String > Bean を参照します。

```
<util:list id="uriTemplates">
    <value>/users/{username}</value>
    <value>/atom/collection/{id}/component/{cid}</value>
</util:list>
```

262.5.3. Restlet API を使用した応答の設定

Camel 2.8 から利用可能

org.restlet.Response API を使用して応答を設定する必要がある場合があります。これにより、Restlet API に完全にアクセスでき、レスポンスを詳細に制御できます。インラインの Camel プロセッサーから応答を生成する以下のルートスニペットを参照してください。

Restlet Response API を使用した応答の生成

262.5.4. コンポーネントでの最大スレッドの設定

最大スレッドオプションを設定するには、以下のようにコンポーネントでこれを実行する必要があります。

```
<bean id="restlet" class="org.apache.camel.component.restlet.RestletComponent">
  <property name="maxThreads" value="100"/>
</bean>
```

262.5.5. webapp 内での Restlet サーブレットの使用

Camel 2.8 で利用可能

では、サーブレットコンテナー内で Restlet アプリケーションを設定し、サブクラスされた SpringServerServlet を使用すると、Restlet コンポーネントを注入して Camel 内の設定を可能にする 3 つの方法 があります。

サーブレットコンテナー内の Restlet サーブレットを使用すると、URI の相対パスでルートを設定できます（ハードコーディングされた絶対 URI の制限を削除）。また、ホストサーブレットコンテナーが受信要求を処理するようにするには（新しいポートで別のサーバープロセスを生成する場合を除く）。

設定するには、以下を camel-context.xml に追加します。

```
<camelContext>
  <route id="RS_RestletDemo">
```

```
    <from uri="restlet:/demo/{id}" />
    <transform>
      <simple>Request type : ${header.CamelHttpMethod} and ID : ${header.id}</simple>
    </transform>
  </route>
</camelContext>

<bean id="RestletComponent" class="org.restlet.Component" />

<bean id="RestletComponentService"
class="org.apache.camel.component.restlet.RestletComponent">
  <constructor-arg index="0">
    <ref bean="RestletComponent" />
  </constructor-arg>
</bean>
```

そして、これを web.xml に追加します。

```
<!-- Restlet Servlet -->
<servlet>
  <servlet-name>RestletServlet</servlet-name>
  <servlet-class>org.restlet.ext.spring.SpringServerServlet</servlet-class>
  <init-param>
    <param-name>org.restlet.component</param-name>
    <param-value>RestletComponent</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>RestletServlet</servlet-name>
  <url-pattern>/rs/*</url-pattern>
</servlet-mapping>
```

その後、ここでデプロイされたルートには http://localhost:8080/mywebapp/rs/demo/1234 からアクセスできます。

localhost:8080 はサーブレットコンテナーのサーバーおよびポートです。
mywebapp はデプロイ済み webapp
の名前です。その後、ブラウザーは以下の内容を表示します。

```
"Request type : GET and ID : 1234"
```

Maven pom.xml ファイルで実行できる restlet に、依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.restlet.jee</groupId>
  <artifactId>org.restlet.ext.spring</artifactId>
```

```
  <version>${restlet-version}</version>
</dependency>
```

この場合、restlet Maven リポジトリーにも依存関係を追加する必要があります。

```
<repository>
  <id>maven-restlet</id>
  <name>Public online Restlet repository</name>
  <url>http://maven.restlet.org</url>
</repository>
```

第263章 RIBBON コンポーネント

Camel バージョン 2.18 から利用可能

ribbon コンポーネントは、クライアント側の負荷分散に Netflix Ribbon を使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ribbon</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

このコンポーネントは、ServiceCall EIP を使用する際にクライアント側で負荷分散機能を適用するのに役立ちます。

263.1. 設定

- プログラマティック

```
RibbonConfiguration configuration = new RibbonConfiguration();
configuration.addProperties("ServerListRefreshInterval", "250");

RibbonLoadBalancer loadBalancer = new RibbonLoadBalancer(configuration);

from("direct:start")
   .serviceCall()
      .name("myService")
      .loadBalancer(loadBalancer)
      .consulServiceDiscovery()
      .end()
   .to("mock:result");
```

- Spring Boot

application.properties

```
camel.cloud.ribbon.properties[ServerListRefreshInterval] = 250
```

routes

```
from("direct:start")
   .serviceCall()
      .name("myService")
      .ribbonLoadBalancer()
      .consulServiceDiscovery()
      .end()
   .to("mock:result");
```

- XML

```
<route>
   <from uri="direct:start"/>
   <serviceCall name="myService">
      <!-- enable ribbon load balancer -->
      <ribbonLoadBalancer>
         <properties key="ServerListRefreshInterval" value="250"/>
      </ribbonLoadBalancer>
   </serviceCall>
</route>
```

263.2. 関連項目

- ServiceCall EIP

第264章 RMI コンポーネント

Camel バージョン 1.0 で利用可能

rmi: コンポーネントは Exchange を RMI プロトコル(JRMP)にバインドします。

このバインディングは RMI を使用するだけなので、通常の RMI ルールは、呼び出すことのできるメソッドに適用されます。このコンポーネントは、Remote インターフェースを拡張するインターフェースからメソッド呼び出しを行う Exchange のみをサポートしています。メソッド内のすべてのパラメーターは、336 または Remote オブジェクトのいずれ か でなければなりません。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rmi</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

264.1. URI 形式

```
rmi://rmi-regisitry-host:rmi-registry-port/registry-path[?options]
```

以下に例を示します。

```
rmi://localhost:1099/path/to/service
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

264.2. オプション

RMI コンポーネントにはオプションがありません。

RMI エンドポイントは、URI 構文を使用して設定します。

```
rmi:hostname:port/name
```

-

以下の path パラメーターおよびクエリーパラメーターを使用します。

264.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
hostname	RMI サーバーのホスト名	localhost	文字列
name	RMI サーバーにバインドする際に使用する ***必須*** 名		文字列
port	RMI サーバーのポート番号	1099	int

264.2.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
メソッド（共通）	呼び出すメソッドの名前を設定できます。		文字列
remoteInterfaces (common)	リモートインターフェースに固有のもの。		リスト
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

264.3. 使用

RMI レジストリーに登録されている既存の RMI サービスを呼び出すには、以下のようなルートを作成します。

```
from("pojo:foo").to("rmi://localhost:1099/foo");
```

RMI レジストリーで既存の camel プロセッサーまたはサービスをバインドするには、以下のように RMI エンドポイントを定義します。

```
RmiEndpoint endpoint= (RmiEndpoint) endpoint("rmi://localhost:1099/bar");
endpoint.setRemoteInterfaces(ISay.class);
from(endpoint).to("pojo:bar");
```

RMI コンシューマーエンドポイントをバインドする場合は、公開される リモート インターフェースを指定する必要があります。

XML DSL では、Camel 2.7 以降では以下のように実行できます。

```
<camel:route>
    <from uri="rmi://localhost:37541/helloServiceBean?
remoteInterfaces=org.apache.camel.example.osgi.HelloService"/>
    <to uri="bean:helloServiceBean"/>
</camel:route>
```

264.4. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*

- *エンドポイント*
- *はじめに*

第265章 ROUTEBOX コンポーネント（非推奨）

Camel バージョン 2.6 で利用可能

変更に関する Routebox の件名

routebox コンポーネントを使用すると、カプセル化機能やストラテジーベースの間接化サービスを、自動作成またはユーザーが挿入された Camel コンテキストでホストされる Camel ルートのコレクションに提供する特殊なエンドポイントを作成できます。

Routebox エンドポイントは、Camel ルートで直接呼び出すことができる Camel エンドポイントです。routebox エンドポイントは、以下の主要な機能を実行します。

- カプセル化 - 内側の Camel コンテキストに保存されている Camel ルートのコレクションをホストするブラックボックスとして機能します。内部コンテキストは routebox コンポーネントで完全に制御され、JVM バインド です。
- ストラテジーベースの間接化： ユーザー定義の内部ルーティングストラテジーまたはディスパッチマップに基づいて、Camel ルートと Camel ルートに送信された直接ペイロード。
- Exchange propagation: routebox エンドポイントによって変更されたエクスチェンジを camel ルートの次のセグメントに転送します。

routebox コンポーネントは、コンシューマーおよびプロデューサーエンドポイントの両方をサポートします。

プロデューサーエンドポイントは 2 つのフレーバーです。

- 外部ルートトレイコンシューマーエンドポイントへの受信リクエストを送受信するプロデューサー
- 内部埋め込み Camel コンテキストでルートを直接呼び出すプロデューサー。これにより、リクエストを外部コンシューマーに送信しません。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-routebox</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

265.1. CAMEL ROUTEBOX エンドポイントが必要

routebox コンポーネントは、複雑な環境での統合を容易にするように設計されています。

- *ルートに関する大規模なコレクション*
- *さまざまな方法でインテグレーションを必要とする各種のエンドポイントテクノロジーを含む*

このような環境では、Camel ルート間でレイヤーを効果的に整理することで、統合ソリューションを作り出す必要があることがよくあります。

- *粒度の細かいレベルルートまたは高レベルのルート： 統合フォーカスエリアを表す Routebox エンドポイントとして公開される内部または下位レベルのルートの集約されたコレクション。たとえば、以下のようになります。*

フォーカスエリア	粒度の細かいルートの例
department Focus	HR ルート、営業ルート等
supply chain & B2B Focus	配送経路、Fulfillment ルート、サードパーティーサービスなど
テクノロジー集約	データベースルート、JMS ルート、スケジュールされているバッチルートなど

- *粒度の細かいルート： 単数および特定のビジネスまたは統合パターンを実行するルート。*

粒度の細かいルートの Routebox エンドポイントに送信される要求は、特定の統合目的の一時的なルートに要求を委譲し、最終的な内部結果を収集し、結合されたルートで次のステップに進むことができます。

265.2. URI 形式

```
routebox:routeboxname[?options]
```

URI にクエリーオプションを追加するには、 ?option=value&option=value&…

265.3. オプション

RouteBox コンポーネントにはオプションがありません。

RouteBox エンドポイントは URI 構文を使用して設定されます。

```
routebox:routeboxName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

265.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
routeboxName	routebox に **必要な** 論理名（キュー名など）		文字列

265.3.2. クエリーパラメーター（17 パラメーター）：

Name	説明	デフォルト	Type
dispatchMap (common)	HashMap タイプのオブジェクト値と一致する Camel Registry のキーを表す文字列。HashMap キーには、エクスチェンジヘッダー ROUTE_DISPATCH_KEY に設定された値と照合できる文字列が含まれている必要があります。HashMap の値には、要求の転送先となる内部ルートコンシューマー URI が含まれている必要があります。		マップ

Name	説明	デフォルト	Type
dispatchStrategy (common)	デフォルトではなくカスタムディスパッチを使用できるようにカスタムの RouteboxDispatchStrategy を使用するには、以下を実行します。		RouteboxDispatchStrategy
forkContext (common)	同じ CamelContext を再利用せずに、新しい内部 CamelContext をフォークして作成するかどうか。	true	boolean
innerProtocol (common)	Routebox コンポーネントによって内部で使用されるプロトコル。Direct または SEDA を指定できます。Routebox コンポーネントは、現在 JVM バインドであるプロトコルを提供します。	直接的な	文字列
queueSize (common)	リクエストを受信する固定サイズキューを作成します。		int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollInterval (consumer)	seda からのポーリング時に使用されるタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	Long
threads （コンシューマー）	リクエストを受信するために routebox によって使用されるスレッドの数。	20	int
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
connectionTimeout (producer)	メッセージの送信時にプロデューサーによって使用されるタイムアウト（ミリ秒単位）。	20000	Long

Name	説明	デフォルト	Type
sendToConsumer (producer)	プロデューサーエンドポイントが外部ルートトレイコンシューマーにリクエストを送信するかどうかを指定します。設定が false の場合、プロデューサーは埋め込みのコンテキストを作成し、内部にリクエストを処理します。	true	boolean
innerContext (advanced)	org.apache.camel.CamelContext タイプのオブジェクト値と一致する Camel Registry のキーを表す文字列。ユーザーが CamelContext を提供しない場合、内部ルートをデプロイするために CamelContext が自動的に作成されます。		CamelContext
innerProducerTemplate (advanced)	内部に組み込まれた CamelContext が使用する ProducerTemplate		ProducerTemplate
innerRegistry (advanced)	内部埋め込み CamelContext にカスタムレジストリーを使用します。		レジストリー
routeBuilders (advanced)	List タイプのオブジェクト値と一致する Camel Registry のキーを表す文字列。ユーザーが内部ルートと事前の innerContext を提供しない場合、routeBuilders オプションは内部ルートを含む RouteBuilder の空ではないリストとして指定する必要があります。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

265.4. ROUTEBOX からのメッセージの送受信

リクエストを送信する前に、以下のように必要な URI パラメーターをレジストリーにロードして routebox を適切に設定する必要があります。Spring の場合、必要な Bean が正しく宣言されている場合、レジストリーは Camel によって自動的に設定されます。

265.4.1. ステップ 1: 内部のルートの詳細をレジストリーに読み込む

```
@Override
protected JndiRegistry createRegistry() throws Exception {
    JndiRegistry registry = new JndiRegistry(createJndiContext());

    // Wire the routeDefinitions & dispatchStrategy to the outer camelContext where the
routebox is declared
```

```
    List<RouteBuilder> routes = new ArrayList<RouteBuilder>();
    routes.add(new SimpleRouteBuilder());
    registry.bind("registry", createInnerRegistry());
    registry.bind("routes", routes);

    // Wire a dispatch map to registry
    HashMap<String, String> map = new HashMap<String, String>();
    map.put("addToCatalog", "seda:addToCatalog");
    map.put("findBook", "seda:findBook");
    registry.bind("map", map);

    // Alternatively wiring a dispatch strategy to the registry
    registry.bind("strategy", new SimpleRouteDispatchStrategy());

    return registry;
}

private JndiRegistry createInnerRegistry() throws Exception {
    JndiRegistry innerRegistry = new JndiRegistry(createJndiContext());
    BookCatalog catalogBean = new BookCatalog();
    innerRegistry.bind("library", catalogBean);

    return innerRegistry;
}
...
CamelContext context = new DefaultCamelContext(createRegistry());
```

265.4.2. ステップ 2: Dispatch マップの代わりに Dispatch ストラテジーを使用したオプションで

ディスパッチストラテジーを使用するには、以下の例のように org.apache.camel.component.routebox.strategy.RouteboxDispatchStrategy インターフェースを実装します。

```
public class SimpleRouteDispatchStrategy implements RouteboxDispatchStrategy {

    /* (non-Javadoc)
     * @see
org.apache.camel.component.routebox.strategy.RouteboxDispatchStrategy#selectDestination
Uri(java.util.List, org.apache.camel.Exchange)
     */
    public URI selectDestinationUri(List<URI> activeDestinations,
            Exchange exchange) {
        URI dispatchDestination = null;

        String operation = exchange.getIn().getHeader("ROUTE_DISPATCH_KEY", String.class);
        for (URI destination : activeDestinations) {
            if (destination.toASCIIString().equalsIgnoreCase("seda:" + operation)) {
                dispatchDestination = destination;
                break;
            }
        }
```

```
        return dispatchDestination;
    }
}
```

265.4.3. ステップ 2: routebox コンシューマーの起動

ルートコンシューマーの作成時には、routeboxUri の # エントリーが、CamelContext レジストリーの作成された内部レジストリー、routebuilder 一覧、および dispatchStrategy/dispatchMap に一致することに注意してください。すべての routebuilder および関連するルートは、内部コンテキストで作成された routebox で起動されることに注意してください。

```
private String routeboxUri = "routebox:multipleRoutes?
innerRegistry=#registry&routeBuilders=#routes&dispatchMap=#map";

public void testRouteboxRequests() throws Exception {
    CamelContext context = createCamelContext();
    template = new DefaultProducerTemplate(context);
    template.start();

    context.addRoutes(new RouteBuilder() {
        public void configure() {
            from(routeboxUri)
                .to("log:Routes operation performed?showAll=true");
        }
    });
    context.start();

    // Now use the ProducerTemplate to send the request to the routebox
    template.requestBodyAndHeader(routeboxUri, book, "ROUTE_DISPATCH_KEY",
"addToCatalog");
}
```

265.4.4. ステップ 3: routebox プロデューサーの使用

要求を routebox に送信する場合、プロデューサーは内部ルートエンドポイント URI を把握する必要はありません。以下に示すように、ディスパッチストラテジーまたは dispatchMap で Routebox URI エンドポイントを呼び出すだけです。

要求が正しい内部ルートに送信できるように、ディスパッチマップのキーと一致するキーを使用して ROUTE_DISPATCH_KEY （Dipatch Strategy でオプション）と呼ばれる特別な交換ヘッダーを設定する必要があります。

```
from("direct:sendToStrategyBasedRoutebox")
    .to("routebox:multipleRoutes?
innerRegistry=#registry&routeBuilders=#routes&dispatchStrategy=#strategy")
    .to("log:Routes operation performed?showAll=true");

from ("direct:sendToMapBasedRoutebox")
```

```
    .setHeader("ROUTE_DISPATCH_KEY", constant("addToCatalog"))
    .to("routebox:multipleRoutes?
innerRegistry=#registry&routeBuilders=#routes&dispatchMap=#map")
    .to("log:Routes operation performed?showAll=true");
```

第266章 RSS コンポーネント

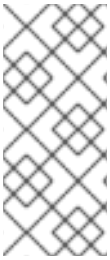
Camel バージョン 2.0 で利用可能

rss: コンポーネントは RSS フィードのポーリングに使用されます。Camel はデフォルトで 60 秒ごとにフィードをポーリングします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rss</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

注記： コンポーネントは現在、ポーリング（時間がかかる）フィードのみをサポートしています。



注記

camel-rss は内部的に、ServiceMix でホストされる ROME のパッチが適用されたバージョン の ROME を使用して、一部の OSGi クラスローディングの問題 を解決します。

266.1. URI 形式

```
rss:rssUri
```

rssUri はポーリングする RSS フィードの URI に置き換えます。

URI にクエリーオプションを追加するには、?option=value&option=value&...

266.2. オプション

RSS コンポーネントにはオプションがありません。

RSS エンドポイントは、URI 構文を使用して設定します。

```
rss:feedUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

266.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
feedUri	ポーリングに **必要な** フィードの URI。		文字列

266.2.2. クエリーパラメーター（27 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
feedHeader (consumer)	フィードオブジェクトをヘッダーとして追加するかどうかを設定します。	true	boolean
Filter （コンシューマー）	エントリーのフィルタリングを使用するかどうかを設定します。	true	boolean
lastUpdate (consumer)	atom フィードからのエントリーのフィルタリングに使用するタイムスタンプを設定します。このオプションは splitEntries と併用されます。		Date
パスワード （コンシューマー）	HTTP フィードからのポーリング時に使用するパスワードを設定します。		文字列
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

Name	説明	デフォルト	Type
sortEntries (consumer)	発行日でエントリーをソートするかどうかを設定します。splitEntries = true の場合にのみ機能します。	false	boolean
splitEntries (consumer)	エントリーを個別に送信するかどうか、またはフィード全体を 1 つのメッセージとして送信すべきかどうかを設定します。	true	boolean
throttleEntries (consumer)	1 つのフィードポーリングで特定されるすべてのエントリーを即座に配信するかどうかを設定します。true の場合、consumer.delay ごとに 1 つのエントリーのみを処理します。splitEntries = true の場合にのみ該当します。	true	boolean
ユーザー名 （コンシューマー）	HTTP フィードからのポーリング時に使用するユーザー名を設定します。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int

Name	説明	デフォルト	Type
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit

Name	説明	デフォルト	Type
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

266.3. データ型の交換

Camel は、ROME SyndFeed を使用してエクスチェンジの In ボディーを初期化します。splitEntries フラグの値に応じて、Camel は SyndEntry が 1 つの SyndFeed または SyndEntrys の java.util.List を返します。

オプション	値	動作
splitEntries	**true**	現在のフィードからの単一エントリーがエクスチェンジに設定されます。
splitEntries	**false**	現在のフィードのエントリー一覧全体がエクスチェンジに設定されます。

266.4. メッセージヘッダー

ヘッダー	説明
CamelRssFeed	**SyncFeed** オブジェクト全体。

266.5. RSS データフォーマット

RSS コンポーネントには RSS データ形式が含まれており、String(XML)と ROME RSS モデルオブジェクト間の変換に使用できます。

- ***marshal = from ROME SyndFeed to XML String***
- ***unmarshal = from XML String から ROME SyndFeed***

RSS データフォーマットを使用するルートは、from("rss:file:src/test/data/rss20.xml?splitEntries=false&consumer.delay=1000").marshal（）.rss（）.to("mock:marshal");

この機能の目的は、Camel の組み込み式を使用して RSS メッセージの操作を可能にすることです。以下に示すように、XPath 式を使用して RSS メッセージをフィルタリングすることができます。以下の例では、タイトルに Camel のあるエントリーがフィルターを通過します。

```
`from("rss:file:src/test/data/rss20.xml?splitEntries=true&consumer.delay=100").marshal().rss().filter().xpath("//item/title[contains(.,'Camel')]").to("mock:result");`
```

ヒント

RSS フィードの URL がクエリー パラメーター を使用する場合、このコンポーネントはクエリーパラメーターを解決します。たとえば、フィードが alt=rss を使用する場合、以下の例は解決されます：from("rss:http://someserver.com/feeds/posts/default?alt=rss&splitEntries=false&consumer.delay=1000").to("bean:rss");

266.6. エントリーのフィルタリング

上記のデータフォーマットセクションに示されるように、XPath を使用してエントリーをフィルタリングできます。Camel の Bean インテグレーションを利用して、独自の条件を実装することもできます。たとえば、上記の XPath の例と同等のフィルターは以下のようになります。

from("rss:file:src/test/data/rss20.xml?splitEntries=true&consumer.delay=100").filter（）.method("myFilterBean", "titleContainsCamel").to("mock:result");

このカスタム Bean は次のようになります。

```
public static class FilterBean {
    public boolean titleContainsCamel(@Body SyndFeed feed) {
        SyndEntry firstEntry = (SyndEntry) feed.getEntries().get(0);
return firstEntry.getTitle().contains("Camel");
```

266.7. 関連項目

- Configuring Camel (Camel の設定)

- *コンポーネント*
- *エンドポイント*
- *はじめに*
- *Atom*

第267章 RSS DATAFORMAT

Camel バージョン 2.1 で利用可能

RSS コンポーネントには RSS データ形式が含まれており、String(XML)と ROME RSS モデルオブジェクト間の変換に使用できます。

- *marshal = from ROME SyndFeed to XML String*
- *unmarshal = from XML String から ROME SyndFeed*

以下を使用するルートは以下のようになります。

この機能の目的は、RSS メッセージの操作に Camel の非常に組み込み式を使用できることです。以下に示すように、XPath 式を使用して RSS メッセージをフィルタリングすることができます。

ヒント

RSS フィードの URL がクエリー パラメーターを使用する場合、このコンポーネントはそれらも理解します。たとえば、フィードが alt=rss を使用する場合、たとえば("rss: http://someserver.com/feeds/posts/default? alt=rss&splitEntries=false&consumer.delay=1000").to("bean:rss"); から実行 できます。

267.1. オプション

RSS データフォーマットは、以下に示す 1 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

第268章 SALESFORCE コンポーネント

Camel バージョン 2.12 から利用可能

このコンポーネントは、Java DTO を使用して Salesforce と通信するためにプロデューサーおよびコンシューマーエンドポイントをサポートします。
これらの DTO を生成するコンパニオンの maven プラグイン Camel Salesforce プラグインがあります（詳細はこちらを参照してください）。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-salesforce</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

注記

コンポーネントに貢献する開発者は、統合テストを実行するための環境の開始および設定方法に関する説明は、README.md ファイルを確認するように指示されます。

268.1. SALESFORCE への認証

コンポーネントは、3 つの OAuth 認証フローをサポートします。

- OAuth 2.0 Username-Password Flow
- OAuth 2.0 の更新トークンフロー
- OAuth 2.0 JWT Bearer Token Flow

フローごとに異なるプロパティーセットに、以下を設定する必要があります。

表268.1 各認証フローに設定するプロパティー

プロパティー	Salesforce での検索場所	フロー
clientId	接続されたアプリケーション、コンシューマーキー	すべてのフロー
clientSecret	接続されたアプリケーション、コンシューマーシークレット	ユーザー名-パスワード、トークンの更新
userName	Salesforce ユーザーのユーザー名	Username-Password、JWT Bearer トークン
password	Salesforce ユーザーパスワード	Username-Password
refreshToken	OAuth フローコールバックから	トークンの更新
keystore	接続されたアプリケーション、デジタル署名	JWT ベアラートークン

コンポーネントは自動的に設定しようとしているフローを判別し、曖昧さをなくし、authenticationType プロパティーを設定します。

注記

Username-Password Flow を実稼働環境で使用することは推奨されません。

注記

JWT Bearer Token Flow で使用される証明書は自己署名証明書になります。証明書と秘密鍵を保持する KeyStore には単一の certificate-private key エントリーのみが含まれている必要があります。

268.2. URI 形式

コンシューマーとして使用し、ストリーミングイベントを受信する場合、URI スキームは以下のようになります。

```
salesforce:topic?options
```

プロデューサーとして使用し、Salesforce RSET API を呼び出す場合、URI スキームは以下のようになります。

```
salesforce:operationName?options
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

268.3. SALESFORCE ヘッダーを渡して SALESFORCE 応答ヘッダーの取得

Camel 2.21 では、インバウンドメッセージ ヘッダーを介して Salesforce ヘッダーを渡すサポートがあり、Camel メッセージ上で Sforce または x-sfdc で始まるヘッダー名はリクエストで渡されます。また、Sforce で始まるレスポンスヘッダーは outboud メッセージヘッダーに存在します。

たとえば、以下を指定する API 制限を取得するには、以下を実行します。

```
// in your Camel route set the header before Salesforce endpoint
//...
  .setHeader("Sforce-Limit-Info", constant("api-usage"))
  .to("salesforce:getGlobalObjects")
  .to(myProcessor);

// myProcessor will receive `Sforce-Limit-Info` header on the outbound
// message
class MyProcessor implements Processor {
    public void process(Exchange exchange) throws Exception {
        Message in = exchange.getIn();
        String apiLimits = in.getHeader("Sforce-Limit-Info", String.class);
    }
}
```

268.4. サポートされる SALESFORCE API

コンポーネントは、以下の Salesforce API をサポートします。

プロデューサーエンドポイントは以下の API を使用できます。API の多くは一度に 1 つのレコードを処理します。Query API は複数のレコードを取得できます。

268.4.1. REST API

operationName には以下を使用できます。

- getVersions - サポートされる Salesforce REST API バージョンを取得します。

- *getResources: 利用可能な Salesforce REST リソースエンドポイントを取得します。*
- *getGlobalObjects: 利用可能なすべての SObject タイプのメタデータを取得します。*
- *getBasicInfo: 特定の SObject タイプの基本メタデータを取得します。*
- *getDescription: 特定の SObject タイプの包括的なメタデータを取得します。*
- *getSObject - Salesforce ID を使用して SObject を取得します。*
- *createSObject - SObject を作成します。*
- *UpdateSObject: Id を使用して SObject を更新します。*
- *deleteSObject: Id を使用して SObject を削除します。*
- *getSObjectWithId: 外部（ユーザー定義）id フィールドを使用して SObject を取得します。*
- *upsertSObject: 外部 ID を使用して SObject を更新または挿入します。*
- *deleteSObjectWithId: 外部 ID を使用して SObject を削除します。*
- *Query - Salesforce SOQL クエリーを実行します。*
- *queryMore: 'query' API から返される結果リンクを使用して、より多くの結果を取得します（多数の結果の場合）。*
- *queryAll: SOQL クエリーを実行します。マージまたは削除が原因で、削除された結果を返します。アーカイブされた Task および Event レコードに関する情報も返します。*

- *検索 - Salesforce SOSL クエリーを実行します。*
- *制限： 組織 API の使用制限の取得*
- *recent: 最新のアイテムの取得*
- *Approval: 承認プロセスのレコードまたはレコード（バッチ）を送信します。*
- *Approvals: すべての承認プロセスの一覧を取得します。*
- *複合： 最大 25 個の REST 要求を送信し、個別の応答を受け取る可能性がある*
- *composite-tree: 1 つの行内に親子関係（最大 5 レベル）を持つ最大 200 レコードを作成します。*
- *composite-batch: バッチで要求の構成を送信する*
- *getBlobField: 個々のレコードから指定された blob フィールドを取得します。*
- *apexCall: ユーザー定義の APEX REST API 呼び出しを実行します。*

たとえば、以下のプロデューサーエンドポイントは upsertSObject API を使用し、sObjectIdName パラメーターで 'Name' を外部 id フィールドとして指定します。リクエストメッセージのボディーは、maven プラグインを使用して生成された SObject DTO である必要があります。応答メッセージは、既存のレコードが更新されている場合、または新規レコードの ID を持つ CreateSObjectResult または新規オブジェクトの作成時のエラーリストのいずれかで null になります。

```
...to("salesforce:upsertSObject?sObjectIdName=Name")...
```

268.4.2. REST Bulk API

プロデューサーエンドポイントは以下の API を使用できます。すべてのジョブデータ形式（例：xml、csv、zip/xml、および zip/csv）がサポートされます。
リクエストと応答は、ルートによってマーシャリング/アンマーシャリングする必要があります。通常、要求は CSV ファイル
などの一部のストリームソースとなり、応答を要求に関連付けるファイルに保存することもできます。

operationName には以下を使用できます。

- *createJob - Salesforce の一括ジョブの作成*
- *getJob: Salesforce ID を使用してジョブを取得します。*
- *closeJob: ジョブを閉じる*
- *abortJob: ジョブの中止*
- *createBatch: 一括ジョブ内でバッチを送信します。*
- *getBatch: Id を使用してバッチを取得します。*
- *getAllBatches: Bulk Job Id のバッチをすべて取得します。*
- *getRequest - Batch のリクエストデータ(XML/CSV)を取得します。*
- *getResults: 完了時にバッチの結果を取得します。*
- *createBatchQuery - SOQL クエリーからバッチを作成します。*
- *getQueryResultIds: Batch Query の Result Ids の一覧を取得します。*

- *getQueryResult: 結果 ID の結果を取得します。*

- *getRecentReports: GET リクエストを Report List リソースに送信して最近確認したレポートを最大 200 取得します。*

- *getReportDescription: レポートのレポート、レポートタイプ、および関連メタデータを表形式または要約またはマトリックスの形式のいずれかで取得します。*

- *executeSyncReport: フィルターを変更せずに同期的にレポートを実行し、最新のサマリーデータを返します。*

- *executeAsyncReport: フィルターと非同期的にレポートのインスタンスを実行し、詳細の有無に関わらずサマリーデータを返します。*

- *getReportInstances: 非同期実行を要求したレポートのインスタンス一覧を返します。一覧の各項目は、レポートの別のインスタンスとして扱われます。*

- *getReportResults: レポートの実行結果が含まれます。*

たとえば、以下のプロデューサーエンドポイントは createBatch API を使用してジョブバッチを作成します。in メッセージには、InputStream（通常は、ファイルから UTF-8 CSV または XML コンテンツなど）に変換できるボディーと、ジョブコンテンツタイプのヘッダーフィールド 'jobId' と、XML、CSV、ZIP_XML または ZIP_CSV を指定できます。put メッセージのボディーには成功した場合は BatchInfo が含まれるか、エラー発生時に SalesforceException をスローします。

```
...to("salesforce:createBatchJob").
```

268.4.3. REST Streaming API

コンシューマーエンドポイントは、以下の sytax を使用してエンドポイントをストリーミングし、作成/更新で Salesforce の通知を受け取ることができます。

トピックの作成およびサブスクライブ

```
from("salesforce:CamelTestTopic?
notifyForFields=ALL&notifyForOperations=ALL&sObjectName=Merchandise__c&updateTopic
=true&sObjectQuery=SELECT Id, Name FROM Merchandise__c")...
```

既存のトピックをサブスクライブするには、以下を実行します。

```
from("salesforce:CamelTestTopic&sObjectName=Merchandise__c")...
```

268.5. 例

268.5.1. ContentWorkspace へのドキュメントのアップロード

Processor インスタンスを使用して Java で ContentVersion を作成します。

```
public class ContentProcessor implements Processor {
    public void process(Exchange exchange) throws Exception {
        Message message = exchange.getIn();

        ContentVersion cv = new ContentVersion();
        ContentWorkspace cw = getWorkspace(exchange);
        cv.setFirstPublishLocationId(cw.getId());
        cv.setTitle("test document");
        cv.setPathOnClient("test_doc.html");
        byte[] document = message.getBody(byte[].class);
        ObjectMapper mapper = new ObjectMapper();
        String enc = mapper.convertValue(document, String.class);
        cv.setVersionDataUrl(enc);
        message.setBody(cv);
    }

    protected ContentWorkspace getWorkSpace(Exchange exchange) {
        // Look up the content workspace somehow, maybe use enrich() to add it to a
        // header that can be extracted here
        ....
    }
}
```

プロセッサーからの出力を Salesforce コンポーネントに出力します。

```
from("file:///home/camel/library")
    .to(new ContentProcessor())     // convert bytes from the file into a ContentVersion SObject
                                    // for the salesforce component
    .to("salesforce:createSObject");
```

268.6. SALESFORCE LIMITS API の使用

salesforce:limits 操作を使用すると、Salesforce から API 制限を取得し、受信したそのデータに対して処理できます。salesforce:limits 操作の結果は org.apache.camel.component.salesforce.api.dto.Limits クラスにマップされ、カスタムプロセッサーまたは式で使用できます。

たとえば、Salesforce の API の使用状況を制限し、日次 API 要求が他のルート用に 10% のままになるようにする必要があるとします。出力メッセージのボディーには、クエリーの実行時に選択できる Content Based Router および Content Based Router and Spring Expression Language(SpEL) と共に使用できる org.apache.camel.component.salesforce.api.dto.Limits オブジェクトのインスタンスが含まれます。

body . dailyApiRequests.remaining に保持される整数値で 1. 0 を乗算したことで、式が浮動小数十の浮動小数学的に評価され、ゼロ（一部の API 制限が消費される）または 1 （API の制限が消費されていない）のいずれかの結果になります。

```
from("direct:querySalesforce")
    .to("salesforce:limits")
    .choice()
    .when(spel("#{1.0 * body.dailyApiRequests.remaining / body.dailyApiRequests.max < 0.1}"))
        .to("salesforce:query?...")
    .otherwise()
        .setBody(constant("Used up Salesforce API limits, leaving 10% for critical routes"))
    .endChoice()
```

268.7. 承認の使用

すべてのプロパティーには、承認が付けられた Salesforce REST API と同じ名前が付けられます。承認プロパティーは、Endpoint の approval.PropertyName を設定して設定できます。これらは、body または header のいずれかに存在しないプロパティーが Endpoint 設定から取得することをテンプレートに使用 します。または、レジストリーの Bean への参照に approval プロパティーを割り当てることにより、Endpoint で承認テンプレートを設定できます。

受信メッセージヘッダーで同じ approval.PropertyName を使用してヘッダー値を指定することもできます。

また、最後に ボディー には AprovalRequest または Iterable of ApprovalRequest オブジェクトを 1 つ追加してバッチとして処理できます。

覚えておくべき重要なことは、以下の 3 つのメカニズムで指定された値の優先度です。

1. ボディーの値は、他の値よりも先に優先されます。

2. メッセージヘッダーの値は、テンプレート値の前に優先されます。

3. ヘッダーまたはボディーに他の値が指定されていない場合に、テンプレートの値が設定されます。

たとえば、ヘッダーの値を使用して承認のために 1 つのレコードを送信するには、以下を使用します。

ルートを指定します。

```
from("direct:example1")//
        .setHeader("approval.ContextId", simple("${body['contextId']}"))
        .setHeader("approval.NextApproverIds", simple("${body['nextApproverIds']}"))
        .to("salesforce:approval?"//
           + "approval.actionType=Submit"//
           + "&approval.comments=this is a test"//
           + "&approval.processDefinitionNameOrId=Test_Account_Process"//
           + "&approval.skipEntryCriteria=true");
```

以下を使用して承認のレコードを送信できます。

```
final Map<String, String> body = new HashMap<>();
body.put("contextId", accountIds.iterator().next());
body.put("nextApproverIds", userId);

final ApprovalResult result = template.requestBody("direct:example1", body,
ApprovalResult.class);
```

268.8. SALESFORCE RECENT ITEMS API の使用

最新のアイテムを取得するには、salesforce:recent 操作を使用します。この操作は、Id、Name、および Attributes （ type および url プロパティーを持つ）が含まれる org.apache.camel.component.salesforce.api.dto.RecentItemオブジェクト(List <RecentItem>)の java.util.List を返します。limit パラメーターを返す最大レコード数に設定すると、返されるアイテム数を制限できます。以下に例を示します。

```
from("direct:fetchRecentItems")
    to("salesforce:recent")
        .split().body()
```

```
        .log("${body.name} at ${body.attributes.url}");
```

268.9. 承認の使用

すべてのプロパティーには、承認が付けられた Salesforce REST API と同じ名前が付けられます。承認プロパティーは、Endpoint の approval.PropertyName を設定して設定できます。これらは、body または header のいずれかに存在しないプロパティーが Endpoint 設定から取得することをテンプレートに使用 します。または、レジストリーの Bean への参照に approval プロパティーを割り当てることにより、Endpoint で承認テンプレートを設定できます。

受信メッセージヘッダーで同じ approval.PropertyName を使用してヘッダー値を指定することもできます。

また、最後に ボディー には AprovalRequest または Iterable of ApprovalRequest オブジェクトを 1 つ追加してバッチとして処理できます。

覚えておくべき重要なことは、以下の 3 つのメカニズムで指定された値の優先度です。

1. ボディーの値は、他の値よりも先に優先されます。
2. メッセージヘッダーの値は、テンプレート値の前に優先されます。
3. ヘッダーまたはボディーに他の値が指定されていない場合に、テンプレートの値が設定されます。

たとえば、ヘッダーの値を使用して承認のために 1 つのレコードを送信するには、以下を使用します。

ルートを指定します。

```
from("direct:example1")//
        .setHeader("approval.ContextId", simple("${body['contextId']}"))
        .setHeader("approval.NextApproverIds", simple("${body['nextApproverIds']}"))
        .to("salesforce:approval?"//
           + "approvalActionType=Submit"//
```

```
        + "&approvalComments=this is a test"//
        + "&approvalProcessDefinitionNameOrId=Test_Account_Process"//
        + "&approvalSkipEntryCriteria=true");
```

以下を使用して承認のレコードを送信できます。

```
final Map<String, String> body = new HashMap<>();
body.put("contextId", accountIds.iterator().next());
body.put("nextApproverIds", userId);

final ApprovalResult result = template.requestBody("direct:example1", body,
ApprovalResult.class);
```

268.10. SALESFORCE COMPOSITE API を使用した SOBJECT ツリーの送信

親子関係を含む最大 200 レコードを作成するには、 salesforce:composite-tree 操作を使用します。これには、入力メッセージに org.apache.camel.component.salesforce.api.dto.composite.SObjectTree のインスタンスが必要で、出力メッセージで同じオブジェクトツリーを返します。ツリー内の org.apache.camel.component.salesforce.api.dto.AbstractSObjectBase インスタンスは、識別子の値（Id プロパティー）または対応する org.apache.camel.component.salesforce.api.dto.composite.SObjectNode の エラー で更新されます。

一部のレコード操作は成功し、一部はエラーを手動で確認する必要がある点に注意してください。

この機能を使用する最も簡単な方法は、camel-salesforce-maven-plugin によって生成された DTO を使用することですが、データベースからのインスタンスのプライマリーキーなど、ツリー内の各オブジェクトを識別する参照をカスタマイズするオプションもあります。

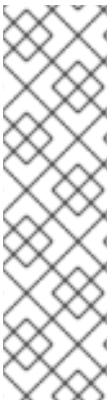
例を見てみましょう。

```
Account account = ...
Contact president = ...
Contact marketing = ...

Account anotherAccount = ...
Contact sales = ...
Asset someAsset = ...

// build the tree
SObjectTree request = new SObjectTree();
request.addObject(account).addChildren(president, marketing);
request.addObject(anotherAccount).addChild(sales).addChild(someAsset);
```

```
final SObjectTree response = template.requestBody("salesforce:composite-tree", tree,
SObjectTree.class);
final Map<Boolean, List<SObjectNode>> result = response.allNodes()
                                        .collect(Collectors.groupingBy(SObjectNode::hasErrors));

final List<SObjectNode> withErrors = result.get(true);
final List<SObjectNode> succeeded = result.get(false);

final String firstId = succeeded.get(0).getId();
```

268.11. SALESFORCE COMPOSITE API を使用したバッチでの複数のリクエストの送信

Composite API バッチ操作(composite-batch)を使用すると、バッチで複数のリクエストを累積してから 1 回のリクエストを送信して送信し、複数の個別のリクエストのラウンドトリップコストを節約できます。その後、各応答は保持された順序を持つ応答のリストで受信され、n 番目のリクエスト応答は応答の n 番目の位置に置かれます。



注記

結果は API から API に異なるため、リクエストの結果は java.lang.Object として指定されます。ほとんどの場合、結果は文字列キーおよび値や他の java.util.Map を値として持つ java.util.Map になります。JSON 形式で行われるリクエストはタイプ情報を保持します（つまり、文字列の値と数字の値が分かっているなど）、一般的にはタイプフレンドリーになります。応答は XML と JSON ごとに異なることに注意してください。これは、Salesforce API からの応答が異なるためです。したがって、レスポンス処理コードを変更せずに形式を切り替える場合には注意してください。

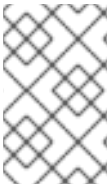
例を見てみましょう。

```
final String acountId = ...
final SObjectBatch batch = new SObjectBatch("38.0");

final Account updates = new Account();
updates.setName("NewName");
batch.addUpdate("Account", accountId, updates);

final Account newAccount = new Account();
newAccount.setName("Account created from Composite batch API");
batch.addCreate(newAccount);

batch.addGet("Account", accountId, "Name", "BillingPostalCode");

batch.addDelete("Account", accountId);

final SObjectBatchResponse response = template.requestBody("salesforce:composite-batch?
format=JSON", batch, SObjectBatchResponse.class);

boolean hasErrors = response.hasErrors(); // if any of the requests has resulted in either 4xx
```

```
or 5xx HTTP status
final List<SObjectBatchResult> results = response.getResults(); // results of three operations sent in batch

final SObjectBatchResult updateResult = results.get(0); // update result
final int updateStatus = updateResult.getStatusCode(); // probably 204
final Object updateResultData = updateResult.getResult(); // probably null

final SObjectBatchResult createResult = results.get(1); // create result
@SuppressWarnings("unchecked")
final Map<String, Object> createData = (Map<String, Object>) createResult.getResult();
final String newAccountId = createData.get("id"); // id of the new account, this is for JSON, for XML it would be createData.get("Result").get("id")

final SObjectBatchResult retrieveResult = results.get(2); // retrieve result
@SuppressWarnings("unchecked")
final Map<String, Object> retrieveData = (Map<String, Object>) retrieveResult.getResult();
final String accountName = retrieveData.get("Name"); // Name of the retrieved account, this is for JSON, for XML it would be createData.get("Account").get("Name")
final String accountBillingPostalCode = retrieveData.get("BillingPostalCode"); // Name of the retrieved account, this is for JSON, for XML it would be createData.get("Account").get("BillingPostalCode")

final SObjectBatchResult deleteResult = results.get(3); // delete result
final int updateStatus = deleteResult.getStatusCode(); // probably 204
final Object updateResultData = deleteResult.getResult(); // probably null
```

268.12. SALESFORCE COMPOSITE API を使用した複数のチェーンされたリクエストの送信

複合 操作では、前のリクエストで生成されたインスタンス識別子を後続のリクエストで使用することができます。個々のリクエストと応答は、提供された 参照 とリンクされます。



注記

複合 API は JSON ペイロードのみをサポートします。

注記

バッチ API の場合、結果は API から API に異なるため、リクエストの結果は java.lang.Object として指定されます。ほとんどの場合、結果は文字列キーおよび値や他の java.util.Map を値として持つ java.util.Map になります。JSON 形式で行われるリクエストはタイプ情報を保持します（つまり、文字列の値と数字の値が分かっているなど）、一般的にはタイプフレンドリーになります。

例を見てみましょう。

```
SObjectComposite composite = new SObjectComposite("38.0", true);

// first insert operation via an external id
final Account updateAccount = new TestAccount();
updateAccount.setName("Salesforce");
updateAccount.setBillingStreet("Landmark @ 1 Market Street");
updateAccount.setBillingCity("San Francisco");
updateAccount.setBillingState("California");
updateAccount.setIndustry(Account_IndustryEnum.TECHNOLOGY);
composite.addUpdate("Account", "001xx000003DIpcAAG", updateAccount,
"UpdatedAccount");

final Contact newContact = new TestContact();
newContact.setLastName("John Doe");
newContact.setPhone("1234567890");
composite.addCreate(newContact, "NewContact");

final AccountContactJunction__c junction = new AccountContactJunction__c();
junction.setAccount__c("001xx000003DIpcAAG");
junction.setContactId__c("@{NewContact.id}");
composite.addCreate(junction, "JunctionRecord");

final SObjectCompositeResponse response = template.requestBody("salesforce:composite?
format=JSON", composite, SObjectCompositeResponse.class);
final List<SObjectCompositeResult> results = response.getCompositeResponse();

final SObjectCompositeResult accountUpdateResult = results.stream().filter(r ->
"UpdatedAccount".equals(r.getReferenceId())).findFirst().get()
final int statusCode = accountUpdateResult.getHttpStatusCode(); // should be 200
final Map<String, ?> accountUpdateBody = accountUpdateResult.getBody();

final SObjectCompositeResult contactCreationResult = results.stream().filter(r ->
"JunctionRecord".equals(r.getReferenceId())).findFirst().get()
```

268.13. CAMEL SALESFORCE MAVEN プラグイン

この Maven プラグインは Camel Salesforce の DTO を生成します。

268.14. オプション

Salesforce コンポーネントは、以下に示す 29 個のオプションをサポートします。

Name	説明	デフォルト	Type

Name	説明	デフォルト	Type
authenticationType (security)	使用する明示的な認証方法、USERNAME_PASSWORD、REFRESH_TOKEN、または JWT のいずれか。Salesforce コンポーネントは、プロパティーセットから使用する認証方法を自動決定できます。このプロパティーを設定して曖昧さを排除できます。		AuthenticationType
loginConfig (security)	1 つのネストされた Bean のすべての認証設定。そこに設定されたすべてのプロパティーはコンポーネントにも直接設定できます。		SalesforceLoginConfig
instanceUrl (security)	認証後に使用される Salesforce インスタンスの URL。デフォルトでは認証に成功したときに Salesforce から受信されます。		文字列
loginUrl (security)	認証に使用される Salesforce インスタンスに **必要な** URL。デフォルトでは https://login.salesforce.comに設定されます。	https://login.salesforce.com	文字列
clientId (security)	Salesforce インスタンス設定で設定される接続されたアプリケーションの OAuth Consumer Key が **必要です**。通常、接続されたアプリを設定する必要がありますが、パッケージをインストールして提供することができます。		文字列
clientSecret (security)	Salesforce インスタンス設定で設定される接続されたアプリケーションの OAuth Consumer Secret。		文字列
キーストア （セキュリティー）	OAuth JWT フローで使用するキーストアパラメーター。KeyStore には、秘密鍵と証明書が含まれるエントリーを 1 つだけ含める必要があります。Salesforce は証明書チェーンを検証しないため、これは簡単に自己署名証明書にすることができます。必ず、対応する接続したアプリケーションに証明書をアップロードするようにしてください。		KeyStoreParameters

Name	説明	デフォルト	Type
refreshToken （セキュリティー）	更新トークンの OAuth フローですでに取得されたトークンを更新します。1 つは Web アプリケーションを設定し、更新トークンを受信するようにコールバック URL を設定するか、https://login.salesforce.com/services/oauth2/success または https://test.salesforce.com/services/oauth2/success で組み込みコールバックを使用してから、フローの最後にある URL から refresh_token を再設定します。開発中の組織 Salesforce では、コールバック Web アプリケーションを localhost でホストできることに注意してください。		文字列
userName (security)	OAuth フローでアクセストークンへのアクセスを取得するために使用されるユーザー名。パスワード OAuth フローの使用は簡単ですが、通常は他のフローよりも安全性が低いとみなされるため回避する必要があります。		文字列
パスワード （セキュリティー）	アクセストークンにアクセスするために OAuth フローで使用されるパスワード。パスワード OAuth フローの使用は簡単ですが、通常は他のフローよりも安全性が低いとみなされるため回避する必要があります。パスワードの最後にセキュリティートークンを追加するようにしてください。		文字列
lazyLogin (security)	true に設定すると、コンポーネントはコンポーネントの先頭で Salesforce に対して認証されなくなります。通常、これを（デフォルト）false に設定し、早い段階で認証し、認証の問題をすぐに認識します。	false	boolean
config (common)	グローバルエンドポイント設定： すべてのエンドポイントに共通する値を設定するのに使用します。		SalesforceEndpoint Config
httpClientProperties (common)	ベースとなる HTTP クライアントで設定できる任意のプロパティーを設定するために使用されます。利用可能なすべてのオプションについて SalesforceHttpClient と Jetty HttpClient のプロパティーを確認します。		マップ
longPollingTransport Properties (common)	ストリーミング API によって使用される BayeuxClient(CometD)で使用される LongPollingTransport(LongPollingTransport)で設定できるプロパティーを設定するために使用されます。		マップ

Name	説明	デフォルト	Type
sslContextParameters (security)	使用する SSL パラメーター。利用可能なすべてのオプションについては SSLContextParameters クラスを参照してください。		SSLContextParameters
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用の有効化	false	boolean
httpProxyHost (proxy)	使用する HTTP プロキシーサーバーのホスト名。		文字列
httpProxyPort (proxy)	使用する HTTP プロキシーサーバーのポート番号。		整数
httpProxyUsername (security)	HTTP プロキシーサーバーに対して認証するために使用するユーザー名。		文字列
httpProxyPassword (security)	HTTP プロキシーサーバーに対して認証するために使用するパスワード。		文字列
isHttpProxySocks4 (proxy)	true に設定すると、HTTP プロキシーが SOCKS4 プロキシーとして使用するように設定されます。	false	boolean
isHttpProxySecure (security)	false に設定すると、HTTP プロキシーへのアクセス時に TLS の使用が無効になります。	true	boolean
httpProxyIncluded Addresses (proxy)	HTTP プロキシーサーバーを使用するアドレスの一覧。		Set
httpProxyExcluded Addresses (proxy)	HTTP プロキシーサーバーを使用できないアドレスの一覧。		Set
httpProxyAuthUri (security)	HTTP プロキシーサーバーに対する認証で使用され、httpProxyUsername および httpProxyPassword を認証に使用するにはプロキシーサーバーの URI と一致する必要があります。		文字列
httpProxyRealm (security)	HTTP プロキシーサーバーに対する プリエンプションプ Basic/Digest 認証メソッドで使用されるプロキシーサーバーのレルム。		文字列
httpProxyUseDigest Auth (security)	true に設定すると、HTTP プロキシーに対して認証時に true ダイジェスト認証が使用されます。そうでないと、Basic 認証方法が使用されます。	false	boolean

Name	説明	デフォルト	Type
packages (common)	生成された DTO クラスがどのパッケージであるか。通常、クラスは camel-salesforce-maven-plugin を使用して生成されます。生成された DTO を使用して、パラメーター/ヘッダー値で短い SObject 名を使用する利点を取得する場合はこれを設定します。		String[]
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Salesforce エンドポイントは、URI 構文を使用して設定します。

```
salesforce:operationName:topicName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

268.14.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
operationName	使用する操作		OperationName
topicName	使用するトピックの名前		文字列

268.14.2. クエリーパラメーター（44 パラメーター）：

Name	説明	デフォルト	Type
apexMethod (common)	APEX メソッド名		文字列
apexQueryParams (common)	APEX メソッドのクエリーパラメーター		マップ
apexUrl (common)	APEX メソッド URL		文字列

Name	説明	デフォルト	Type
apiVersion (common)	Salesforce API バージョン。デフォルトは SalesforceEndpointConfig.DEFAULT_VERSION です。		文字列
backoffIncrement (common)	Streaming connection restart attempts for attempting connection restart failed beyond CometD auto-reconnect をストリーミングするバックオフ間隔がインクリメントされます。		Long
batchId (common)	一括 API バッチ ID		文字列
contentType (common)	一括 API コンテンツタイプ、XML、CSV、ZIP_XML、ZIP_CSV のいずれか		ContentType
defaultReplayId (common)	リンク initialReplayIdMap に値が見つからない場合のデフォルト replayId 設定		Long
形式 (common)	JSON または XML のいずれかの Salesforce API 呼び出しに使用するペイロード形式。デフォルトは JSON です。		PayloadFormat
httpClient (common)	Salesforce への接続に使用するカスタム Jetty Http クライアント。		SalesforceHttpClient
includeDetails (common)	Salesforce1 Analytics レポートに詳細を含めます。デフォルトは false です。		ブール値
initialReplayIdMap (common)	チャネル名ごとに開始する ID を再生します。		マップ
instanceId (common)	Salesforce1 Analytics 実行インスタンス ID		文字列
jobId （共通）	一括 API ジョブ ID		文字列
制限 （共通）	返されるレコードの数の制限。一部の API に適用して、Salesforce ドキュメントを確認してください。		整数
maxBackoff （共通）	接続再起動のストリーミングの最大バックオフ間隔が CometD auto-reconnect 以外の失敗に対して試行されます。		Long

Name	説明	デフォルト	Type
notFoundBehaviour (common)	Salesforce API から受信した 404 not found ステータスの動作を設定します。ボディーを NULL リンク NotFoundBehaviourNULL に設定するか、エクスチェンジリンク NotFoundBehaviourEXCEPTION - デフォルトで例外が通知される必要があります。		NotFoundBehaviour
notifyForFields (common)	フィールドについて通知し、オプションは ALL、REFERENCED、SELECT、WHERE です。		NotifyForFieldsEnum
notifyForOperationCreate (common)	作成操作についての通知。デフォルトは false（API バージョン = 29.0）です。		ブール値
notifyForOperationDelete (common)	削除操作についての通知。デフォルトは false(API version = 29.0)です。		ブール値
notifyForOperations (common)	操作について通知。オプションは ALL、CREATE、EXTENDED、UPDATE（API バージョン 29.0）です。		NotifyForOperations Enum
notifyForOperationUndelete (common)	削除しない操作についての通知。デフォルトは false(API version = 29.0)です。		ブール値
notifyForOperationUpdate (common)	更新操作についての通知。デフォルトは false（API バージョン = 29.0）です。		ブール値
objectMapper (common)	Salesforce オブジェクトをシリアライズ/デシリアライズする際に使用するカスタム Jackson ObjectMapper。		ObjectMapper
rawPayload (common)	デフォルトでは、DTO ではなく、要求および応答に raw ペイロード文字列（JSON または XML のいずれか）を使用します。	false	boolean
reportId (common)	Salesforce1 Analytics レポート ID		文字列
reportMetadata (common)	フィルタリングに使用する Salesforce1 Analytics レポートメタデータ		ReportMetadata
resultId (common)	一括 API 結果 ID		文字列

Name	説明	デフォルト	Type
serializeNulls (common)	指定した DTO の NULL 値が空の(NULL)値でシリアライズされるかどうか。これは JSON データフォーマットにのみ影響します。	false	boolean
sObjectBlobFieldName (common)	SObject blob フィールド名		文字列
sObjectClass (common)	完全修飾 SObject クラス名。通常は camel-salesforce-maven-plugin を使用して生成されます。		文字列
sObjectFields (common)	取得する SObject フィールド		文字列
sObjectId (common)	SObject ID（API で必要な場合）		文字列
sObjectIdName (common)	SObject 外部 ID フィールド名		文字列
sObjectIdValue (common)	SObject 外部 ID フィールドの値		文字列
sObjectName (common)	SObject 名（必要な場合または API でサポートされる場合）		文字列
sObjectQuery (common)	Salesforce SOQL クエリー文字列		文字列
sObjectSearch (common)	Salesforce SOSL 検索文字列		文字列
updateTopic (common)	Streaming API の使用時に既存の Push Topic を更新するかどうか。デフォルトは false です。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
replayId (consumer)	サブスクライブ時に使用する replayId 値		Long

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

セキュリティー上の理由から、clientId、clientSecret、userName、および password フィールドは pom.xml に設定しないことが推奨されます。他のプロパティーに対してプラグインを設定し、以下のコマンドを使用して実行できます。

```
mvn camel-salesforce:generate -DcamelSalesforce.clientId=<clientid> -
DcamelSalesforce.clientSecret=<clientsecret> \
    -DcamelSalesforce.userName=<username> -DcamelSalesforce.password=<password>
```

生成された DTO は Jackson および XStream アノテーションを使用します。すべての Salesforce フィールドタイプがサポートされます。日付と時刻フィールドは Joda DateTime にマッピングされ、picklist フィールドは生成された Java Enumerations にマッピングされます。

268.15. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第269章 SAP コンポーネント

SAP コンポーネントは、10 つの異なる SAP コンポーネントスイートで構成されるパッケージです。sRFC、tRFC、qRFC プロトコルをサポートするリモート関数呼び出し(RFC)コンポーネントと、IDoc 形式のメッセージを使用した通信を容易にする IDoc コンポーネントがあります。コンポーネントは SAP Java Connector(SAP JCo)ライブラリーを使用して、SAP および SAP IDoc ライブラリーとの双方向通信を容易にし、Intermediate Document(IDoc)形式のドキュメントの送信を容易にします。

269.1. 概要

依存関係

このコンポーネントを使用するには、Maven ユーザーは以下の依存関係を pom.xml ファイルに追加する必要があります。

```
<dependency>
  <groupId>org.fusesource</groupId>
  <artifactId>camel-sap</artifactId>
  <version>x.x.x</version>
<dependency>
```

SAP コンポーネントの追加プラットフォームの制限

SAP コンポーネントはサードパーティーの JCo 3.0 ライブラリーおよび IDoc 3.0 ライブラリーに依存するため、これらのライブラリーがサポートするプラットフォームにのみインストールできます。プラットフォームの制限の詳細は、「Red Hat Fuse でサポートされる構成」を参照してください。

SAP JCo ライブラリーおよび SAP IDoc ライブラリー

SAP コンポーネントを使用するための前提条件として、SAP Java Connector(SAP JCo)ライブラリーと SAP IDoc ライブラリーが Java ランタイムの lib/ ディレクトリーにインストールされていることです。SAP Service Marketplace からターゲットオペレーティングシステムの適切な SAP ライブラリーセットをダウンロードすることを確認する必要があります。

ライブラリーファイルの名前は、表269.1「必要な SAP ライブラリー」に示されるように、ターゲットのオペレーティングシステムによって異なります。

表269.1 必要な SAP ライブラリー

SAP コンポーネント	Linux および UNIX	Windows

SAP コンポーネント	Linux および UNIX	Windows
SAP JCo 3	**sapjco3.jar** **libsapjco3.so**	**sapjco3.jar** **sapjco3.dll**
SAP IDoc	**sapidoc3.jar**	**sapidoc3.jar**

Fuse OSGi コンテナーへのデプロイ（Fabric 以外）

SAP JCo ライブラリーおよび SAP IDoc ライブラリーを、以下のように JBoss Fuse OSGi コンテナー（Fabric 以外）にインストールできます。

1. *SAP JCo ライブラリーおよび SAP IDoc ライブラリーを SAP Service Marketplace(http://service.sap.com/public/connectors)からダウンロードし、オペレーティングシステムに適したバージョンのライブラリーを選択するようにしてください。*

注記

バージョン 3.0.11 以上は、JCo ライブラリーおよび IDoc ライブラリーのバージョン 3.0.10 以上が必要です。これらのライブラリーをダウンロードして使用するには、SAP Service Marketplace Account が必要です。

2. *sapjco3.jar、libsapjco3.so （Windows の場合は sapjco3.dll ）、および sapidoc3.jar ライブラリーファイルを Fuse インストールの lib/ ディレクトリーにコピーします。*

3. *テキストエディターで設定プロパティーファイル etc/config.properties とカスタムプロパティーファイル(etc/custom.properties)の両方を開きます。etc/config.properties ファイルで、org.osgi.framework.system.packages.extra プロパティーを探し、完全なプロパティー設定をコピーします（この設定は、バックスラッシュ文字で複数の行を拡張し、行継続を示すために使用される \ です）。この設定を etc/custom.properties ファイルに貼り付けます。*

SAP ライブラリーのサポートに必要な追加パッケージを追加できるようになりました。etc/custom.properties ファイルで、以下のように org.osgi.framework.system.packages.extra 設定に必要なパッケージを追加します。

```
org.osgi.framework.system.packages.extra = \
... , \
com.sap.conn.idoc, \
com.sap.conn.idoc.jco, \
com.sap.conn.jco, \
```

```
com.sap.conn.jco.ext, \
com.sap.conn.jco.monitor, \
com.sap.conn.jco.rt, \
com.sap.conn.jco.server
```

リストが適切に続行されるように、各行の末尾にコンマとバックスラッシュ , , \ を追加することを忘れないでください。

4. これらの変更を有効にするには、コンテナーを再起動する必要があります。

5. コンテナーに camel-sap 機能をインストールする必要があります。Karaf コンソールで、以下のコマンドを入力します。

```
JBossFuse:karaf@root> features:install camel-sap
```

Fuse Fabric でのデプロイ

SAP コンポーネントを使用するための前提条件として、SAP Java Connector(SAP JCo)ライブラリーと SAP IDoc ライブラリーが Java ランタイムの lib/ ディレクトリー（ sapjco3.jar、libsapjco3.so（Windows では sapjco3. dll）、および sapidoc3.jar にインストールする必要があります。

Fuse Fabric デプロイメントの場合は、特別な設定が必要になります。Fabric コンテナーは、ネットワーク上のどこにでもデプロイ可能な必要があるため、単一のマシンの Java lib ディレクトリーに SAP ライブラリーをインストールすることのみはありません。正しい方法は、SAP JCo ライブラリーおよび SAP IDoc ライブラリーをダウンロードおよびインストールできる特別なプロファイルと、実行しているホストをすべて定義することです。

SAP JCo ライブラリーおよび SAP IDoc ライブラリーのプロファイルを以下のように定義できます。

1. JCo ライブラリーおよび IDoc ライブラリーで、 sapjco3.jar、libsapjco3.so （Windows の場合は sapjco3.dll ）、および sapidoc3.jar でアクセス可能な場所にデプロイします。たとえば、ライブラリーを Web サーバーにインストールし、JCo ライブラリーおよび IDoc ライブラリーを HTTP URL、http://mywebserver/sapjco3.jar、http://mywebserver/libsapjco3.so、および http://mywebserver/sapidoc3.jar を使用してダウンロードできます。

2. 以下のコンソールコマンドを入力して、新しい プロファイル camel-sap- profile を作成します。

```
JBossFuse:karaf@root> profile-create camel-sap-profile
```

3. 以下のコンソールコマンドを入力して、**camel-sap-profile** プロファイルのエージェントプロパティーを編集します。

```
JBossFuse:karaf@root> profile-edit camel-sap-profile
```

4. 組み込みプロファイルエディターが起動します。この組み込みテキストエディターを使用して、以下の内容をエージェントプロパティーに追加します。

```
# Profile:my-camel-sap-profile
attribute.parents = feature-camel

# Deploy JCo3 Libs to Container
lib.sapjco3.jar = http://mywebserver/sapjco3.jar
lib.sapjco3.so = http://mywebserver/libsapjco3.so
lib.sapidoc3.jar = http://mywebserver/sapidoc3.jar

# Append JCo3 Packages and IDoc packages to OSGi system property
# in order to expose JCo3 and IDoc classes to OSGi environment
config.org.osgi.framework.system.packages.extra= \
   ... pass:quotes[_Packages from etc/config.properties file_] ...\
   com.sap.conn.jco, \
   com.sap.conn.jco.ext, \
   com.sap.conn.jco.monitor, \
   com.sap.conn.jco.rt, \
   com.sap.conn.jco.server, \
   com.sap.conn.idoc, \
   com.sap.conn.idoc.jco
```

以下のようにプロパティー設定をカスタマイズします。

lib.sapjco3.jar

HTTP URL を Web サーバーの sapjco3.jar ファイルの実際の場所をカスタマイズします。

lib.sapjco3.so

HTTP URL を Web サーバーの libsapjco3.so ファイル（または sapjco3.dll）の実際の場所をカスタマイズします。

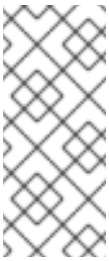
lib.sapidoc3.jar

HTTP URL を Web サーバーの sapidoc3.jar ファイルの実際の場所をカスタマイズします。

config.org.osgi.framework.system.packages.extra

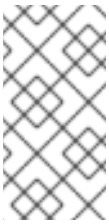
JBoss Fuse インストールのコンテナー設定プロパティーファイル etc/config.properties を開き、org.osgi.framework.system.packages.extra プロパティー設定を見つけます。その設定からパッケージの一覧をコピーし、それらをプロファイルのエージェントプロパティーに貼り付けて、行を置き換えます。

```
... Packages from etc/config.properties file ...\
```



注記

config.org.osgi.framework.system.packages.extra の config. プレフィックスは、プロファイルにコンテナー設定プロパティーを設定する Fabric を示します。*



注記

バックスラッシュ \ は、UNIX 規則（UNIX 規則）の行で、改行文字の直後に指定する必要があります。

完了したら、Ctrl-S と入力してプロパティーを保存します。

5. *camel-sap-profile プロファイルを、SAP コンポーネントを実行する Fabric コンテナーにデプロイできます。たとえば、camel-sap-profileプロファイルを sap-instance コンテナーにデプロイするには、以下を実行します。*

```
JBossFuse:karaf@root> container-add-profile sap-instance came-sap-profile
```

JBoss EAP コンテナーへのデプロイ

JBoss EAP コンテナーに SAP コンポーネントをデプロイするには、以下の手順を実行します。

1. *SAP JCo ライブラリーおよび SAP IDoc ライブラリーを SAP Service Marketplace(http://service.sap.com/public/connectors)からダウンロードし、オペレーティングシステムに適したバージョンのライブラリーを選択するようにしてください。*

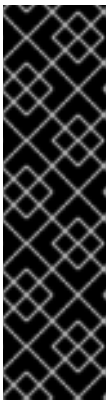
注記

バージョン 3.0.11 以上は、JCo ライブラリーおよび IDoc ライブラリーのバージョン 3.0.10 以上が必要です。これらのライブラリーをダウンロードして使用するには、SAP Service Marketplace Account が必要です。

2.

JCo ライブラリーファイルと IDoc ライブラリーファイルを JBoss EAP インストールの適切なサブディレクトリーにコピーします。たとえば、ホストのプラットフォームが 64 ビット Linux(linux-x86_64)の場合は、以下のようにライブラリーファイルをインストールします。

```
cp sapjco3.jar sapidoc3.jar
$JBOSS_HOME/modules/system/layers/fuse/com/sap/conn/jco/main/
mkdir -p $JBOSS_HOME/modules/system/layers/fuse/com/sap/conn/jco/main/lib/linux-
x86_64
cp libsapjco3.so
$JBOSS_HOME/modules/system/layers/fuse/com/sap/conn/jco/main/lib/linux-x86_64/
```



重要

ネイティブライブラリー（libsapjco3.soなど）を JBoss EAP インストールにインストールする場合は、ライブラリーサブディレクトリーの名前付けの標準化された規則を使用できます。この規則に従う必要があります。64 ビット Linux の場合、サブディレクトリーは linux-x86_ 64 です。その他のプラットフォームについては、https://docs.jboss.org/author/display/MODULES/Native+Libraries を参照してください。

3.

SwitchYard サブシステム設定で org.switchyard.component.camel.sap モジュールのコメントを解除します。たとえば、JBoss EAP スタンドアロンモードで SAP コンポーネントを有効にするには、$JBOSS_HOME/standalone/configuration/standalone.xml ファイルを編集し、以下の行を検索してコメントを解除します。

```
<!-- Uncomment this module to enable camel-sap binding
<module identifier="org.switchyard.component.camel.sap"
implClass="org.switchyard.component.camel.sap.deploy.CamelSapComponent"/>
-->
```

URI 形式

SAP コンポーネントには、RFC(Remote Function Call)エンドポイントと Intermediate Document(IDoc)エンドポイントという 2 種類のエンドポイントがあります。

RFC エンドポイントの URI 形式は以下のとおりです。

```
sap-srfc-destination:destinationName:rfcName
sap-trfc-destination:destinationName:rfcName
sap-qrfc-destination:destinationName:queueName:rfcName
sap-srfc-server:serverName:rfcName[?options]
sap-trfc-server:serverName:rfcName[?options]
```

IDoc エンドポイントの URI 形式は以下のとおりです。

```
sap-idoc-
destination:destinationName:idocType[:idocTypeExtension[:systemRelease[:applicationReleas
e]]]
sap-idoclist-
destination:destinationName:idocType[:idocTypeExtension[:systemRelease[:applicationReleas
e]]]
sap-qidoc-
destination:destinationName:queueName:idocType[:idocTypeExtension[:systemRelease[:applic
ationRelease]]]
sap-qidoclist-
destination:destinationName:queueName:idocType[:idocTypeExtension[:systemRelease[:applic
ationRelease]]]
sap-idoclist-
server:serverName:idocType[:idocTypeExtension[:systemRelease[:applicationRelease]]]
[?options]
```

sap-endpointKind-destination のプレフィックスが付けられた URI 形式は、宛先エンドポイント（つまり Camel プロデューサーエンドポイント）を定義するために使用されます。destinationName は SAP インスタンスへの特定のアウトバウンド接続の名前です。送信接続は、「宛先設定」 で説明されているように、コンポーネントレベルで名前が付けられ、設定されます。

sap-endpointKind-server で始まる URI 形式は、サーバーエンドポイント（つまり Camel コンシューマーエンドポイント）を定義するために使用されます。serverName は SAP インスタンスからの特定の受信接続の名前です。受信接続は、「サーバー設定」 で説明されているように、コンポーネントレベルで名前が付けられ、設定されます。

RFC エンドポイント URI の他のコンポーネントは以下のとおりです。

rfcName

（必須） 宛先エンドポイント URI は、接続された SAP インスタンスのエンドポイントによって呼び出される RFC の名前です。サーバーエンドポイント URI では、接続された SAP インスタンスから呼び出される際にエンドポイントによって処理される RFC の名前です。

queueName

このエンドポイントが SAP リクエストを送信するキューを指定します。

IDoc エンドポイント URI の他のコンポーネントは以下のとおりです。

idocType

（必須） このエンドポイントによって生成された IDoc の Basic IDoc タイプを指定します。

idocTypeExtension

IDoc Type Extension（存在する場合）は、このエンドポイントによって生成される IDoc を指定します。

systemRelease

このエンドポイントによって生成された IDoc が存在する場合は、関連する SAP Basis リリースを指定します。

applicationRelease

関連付けられたアプリケーションリリース（ある場合）を指定（ある場合）。

queueName

このエンドポイントが SAP リクエストを送信するキューを指定します。

RFC 宛先エンドポイントのオプション

RFC 宛先エンドポイント（sap-srfc-destination、sap-trfc-destination、および sap-qrfc-destination）は、以下の URI オプションをサポートします。

Name	デフォルト	説明
stateful	**false**	**true** の場合、このエンドポイントが SAP ステートフルセッションを開始するように指定します。
transacted	**false**	**true** の場合、このエンドポイントが SAP トランザクションを開始することを指定します。

RFC サーバーエンドポイントのオプション

SAP RFC サーバーエンドポイント（sap-srfc-server および sap-trfc-server）は、以下の URI オプションをサポートします。

Name	デフォルト	説明
stateful	**false**	**true** の場合、このエンドポイントが SAP ステートフルセッションを開始するように指定します。
propagateExceptions	**false**	**（SAP-trfc-server エンドポイントのみ） true** の場合、このエンドポイントはエクスチェンジの例外ハンドラーではなく、例外を SAP の呼び出し元に伝播することを指定します。

IDoc List Server エンドポイントのオプション

SAP IDoc List Server エンドポイント(sap-idoclist-server)は、以下の URI オプションをサポートします。

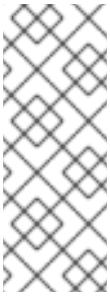
Name	デフォルト	説明
stateful	**false**	**true** の場合、このエンドポイントが SAP ステートフルセッションを開始するように指定します。
propagateExceptions	**false**	**true** の場合、このエンドポイントによってエクスチェンジの例外ハンドラーではなく、例外が SAP の呼び出し元に伝播されることを指定します。

RFC および IDoc エンドポイントの概要

SAP コンポーネントパッケージは、以下の RFC および IDoc エンドポイントを提供します。

sap-srfc-destination

JBoss Fuse SAP Synchronous Remote Function Call Destination Camel コンポーネントこのエンドポイントは、Camel ルートが SAP システムに対するリクエストの同期配信を必要とする場合に使用する必要があります。

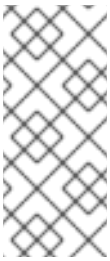


注記

このコンポーネントによって使用される sRFC プロトコルは、SAP システムに対するリクエストおよび応答を ベストエフォート で提供します。リクエストの送信時の通信エラーが発生した場合、受信側の SAP システムにおけるリモート関数呼び出しの完了ステータスは 不明な状態に なります。

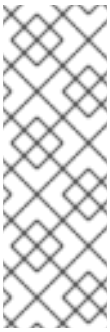
sap-trfc-destination

JBoss Fuse SAP Transactional Remote Function Call Destination Camel コンポーネントこのエンドポイントは、リクエストを 最大 1 回受信 SAP システムに配信する必要がある 場合に使用します。そのために、コンポーネントはトランザクション ID(tid)を生成し、ルートのエクスチェンジでコンポーネント経由で送信されたすべてのリクエストを処理します。受信 SAP システムは、リクエストを配信 する 前に関連するリクエストを記録します。SAP システムが同じ通信で再度要求を受け取った場合 は、リクエストを配信しません。そのため、ルートがこのコンポーネントのエンドポイント経由で要求を送信する際に通信エラーに遭遇した場合、そのルートは配信され 1 回のみ実行されることを知る同じエクスチェンジ内でリクエストの送信を再試行できます。



注記

このコンポーネントによって使用される tRFC プロトコルは非同期で、応答を返しません。そのため、このコンポーネントのエンドポイントは応答メッセージを返しません。

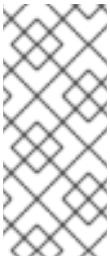


注記

このコンポーネントは、エンドポイント経由の一連のリクエストの順序を保証せず、通信エラーとリクエストの再送信により、これらの要求の配信順序と受信 SAP システムで異なる場合があります。Guaranteed Delivery order については、JBoss Fuse SAP Queued Remote Function Call Destination Camel コンポーネントを参照してください。

sap-qrfc-destination

JBoss Fuse SAP Queued Remote Function Call Destination Camel コンポーネント。このコンポーネントは、エンドポイントを介したリクエストの配信 に 配信保証を追加し、JBoss Fuse Transactional Remote Function Call Destination camel コンポーネントの機能を拡張します。このエンドポイントは、一連のリクエストが相互に依存し、最大で 1 回、かつ 順序で 受信 SAP システムに配信する必要がある 場合に使用します。コンポーネントは、JBoss Fuse SAP Transactional Remote Function Call Destination Camel コンポーネントと同じメカニズムを使用して、最大 1 回 の配信保証を実現します。順序の保証は、SAP システムが受信した順序で要求を受信キュー にシリアライズすることで実現されます。インバウンドキューは、SAP 内の QIN スケジューラー によって処理されます。インバウンドキューが アクティブになる と、QIN スケジューラーがキュー要求を順番に実行します。



注記

このコンポーネントによって使用される qRFC プロトコルは非同期で、応答を返しません。そのため、このコンポーネントのエンドポイントは応答メッセージを返しません。

sap-srfc-server

JBoss Fuse SAP Synchronous Remote Function Call Server Camel コンポーネントCamel ルートが SAP システムからの要求を同期的に処理するのに必要な場合は、このコンポーネントとそのエンドポイントを使用する必要があります。

sap-trfc-server

JBoss Fuse SAP Transactional Remote Function Call Server Camel コンポーネントこのエンドポイントは、送信 SAP システムが Camel ルートへのリクエストを 最大 1 回配信する必要がある 場合に使用する必要があります。そのため、送信した SAP システムはトランザクション ID を生成し、tid を生成し、リクエストを送信するたびにコンポーネントのエンドポイントに送信されます。SAP システムはまず、tid に関連付けられた一連のリクエストを送信する前に、指定された tid が受信したかどうかをチェックします。コンポーネントは、受信した tidの一覧を確認し、その一覧にない場合は送信された tid を記録します。次に、tid がすでに記録されたかどうかを示す、送信の SAP システムに応答します。次に、送信 SAP システムは、tid が以前に記録された場合にのみ一連のリクエストを送信します。これにより、SAP システムを送信して、Camel ルートに対して一連のリクエストを確実に送信できます。

sap-idoc-destination

JBoss Fuse SAP IDoc Destination Camel コンポーネントこのエンドポイントは、Camel ルートが Intermediate Documents(IDoc)の一覧を SAP システムに送信する必要がある場合に使用する必要があります。

sap-idoclist-destination

JBoss Fuse SAP IDoc List Destination Camel コンポーネント。このエンドポイントは、Camel ルートが Intermediate ドキュメント(IDoc)リストのリストを SAP システムに送信する必要がある場合に使用する必要があります。

sap-qidoc-destination

JBoss Fuse SAP Queued IDoc Destination Camel コンポーネント。Camel ルートが Intermediate ドキュメント(IDoc)の一覧を SAP システムに送信する必要がある場合に、このコンポーネントとそのエンドポイントを使用する必要があります。

sap-qidoclist-destination

JBoss Fuse SAP Queued IDoc List Destination Camel component.このコンポーネントとそのエンドポイントは、Camel ルートが Intermediate ドキュメント(IDocs)リストのリストを SAP システムに送信する必要がある場合に使用する必要があります。

sap-idoclist-server

JBoss Fuse SAP IDoc List Server Camel コンポーネント。このエンドポイントは、SAP システムの送信で Intermediate Document リストを Camel ルートに提供する必要がある場合に使用します。このコンポーネントは、sap-trfc-server-standalone クイックスタートで説明されているように、tRFC プロトコルを使用して SAP と通信します。

SAP RFC 宛先エンドポイント

RFC 宛先エンドポイントは SAP への送信通信をサポートします。これにより、これらのエンドポイントは SAP の ABAP 関数モジュールに対して RFC 呼び出しを行うことができます。RFC 宛先エンドポイントは、SAP インスタンスへの特定の接続で特定の ABAP 関数への RFC 呼び出しを行うように設定されます。RFC 宛先はアウトバウンド接続の論理設計であり、一意の名前を持ちます。RFC 宛先は、宛先データ と呼ばれる接続パラメーターのセットで指定されます。

RFC 宛先エンドポイントは、IN-OUT エクスチェンジの入力メッセージから RFC 要求を抽出し、SAP への関数呼び出しでその要求を受信してディスパッチします。関数呼び出しからの応答は、エクスチェンジの出力メッセージで返されます。SAP RFC 宛先エンドポイントはアウトバウンド通信のみをサポートするため、RFC 宛先エンドポイントはプロデューサーの作成のみをサポートします。

SAP RFC サーバーエンドポイント

RFC サーバーエンドポイントは SAP からのインバウンド通信をサポートします。これにより、SAP の ABAP アプリケーションがサーバーエンドポイントへの RFC 呼び出しを行うことができます。ABAP アプリケーションは RFC サーバーエンドポイントと対話し、リモート関数モジュールであるかのように対話します。RFC サーバーエンドポイントは、SAP インスタンスから特定の接続で特定の RFC 関数への RFC 呼び出しを受信するように設定されます。RFC サーバーは受信接続の論理設計で、一意の名前があります。RFC サーバーは、サーバーデータ と呼ばれる接続パラメーターのセットで指定されます。

RFC サーバーエンドポイントは受信した RFC 要求を処理し、IN-OUT 交換の入力メッセージとしてディスパッチします。エクスチェンジの出力メッセージは、RFC 呼び出しの応答として返されます。SAP RFC サーバーエンドポイントはインバウンド通信のみをサポートするため、RFC サーバーエンドポイントはコンシューマーの作成のみをサポートします。

SAP IDoc および IDoc リスト宛先エンドポイント

IDoc 宛先エンドポイントは SAP への送信通信をサポートし、IDoc メッセージでさらに処理できます。IDoc ドキュメントはビジネストランザクションを表し、SAP 以外のシステムと簡単に交換できます。IDoc 宛先は、宛先データ と呼ばれる接続パラメーターのセットで指定します。

IDoc 一覧の宛先エンドポイントは IDoc 宛先エンドポイントと似ていますが、処理するメッセージは

IDoc ドキュメントの 一覧 で構成されます。

SAP IDoc list server endpoint

IDoc リストサーバーのエンドポイントは SAP からのインバウンド通信をサポートし、Camel ルートが SAP システムから IDoc ドキュメントのリストを受信できるようにします。IDoc 一覧サーバーは、サーバーデータ と呼ばれる接続パラメーターのセットで指定されます。

メタデータリポジトリー

meta-data リポジトリーは、以下のようなメタデータを保存するために使用されます。

関数モジュールのインターフェース記述

このメタデータは、JCo および ABAP ランタイムが使用し、これらの呼び出しをディスパッチする前に通信パートナー間でデータをタイプセーフ転送するようにします。リポジトリーにはリポジトリーデータが入力されます。リポジトリーデータは名前付き関数テンプレートのマップです。関数テンプレートには、すべてのパラメーターと、関数モジュールへ渡された入力情報を記述し、それが記述する関数モジュールの一意の名前が含まれます。

IDoc タイプの説明

このメタデータは IDoc ランタイムで使用され、IDoc ドキュメントが通信パートナーに送信される前に正しくフォーマットされるようにします。基本的な IDoc タイプは、名前、許可されるセグメントの一覧、セグメント間の階層関係の説明で構成されます。一部の制約はセグメントに適用することができます。セグメントは必須またはオプションである可能性があり、各セグメントの最小/最大範囲を指定できます（そのセグメントで許可される繰り返しの数を定義できます）。

そのため、SAP 宛先およびサーバーエンドポイントでは、RFC 呼び出しを送受信し、IDoc ドキュメントを送受信するためにリポジトリーへのアクセスが必要になります。RFC 呼び出しでは、エンドポイントによって起動および処理されるすべての関数モジュールのメタデータはリポジトリー内になければなりません。また IDoc エンドポイントの場合は、エンドポイントが処理するすべての IDoc タイプと IDoc タイプの拡張機能のメタデータがリポジトリー内に存在している必要があります。宛先とサーバーエンドポイントによって使用されるリポジトリーの場所は、宛先データと各接続のサーバーデータに指定されます。

SAP 宛先エンドポイントの場合、使用するリポジトリーは通常 SAP システムにあり、デフォルトでは接続されている SAP システムに設定されます。このデフォルトには、宛先データで明示的な設定は必要ありません。さらに、宛先エンドポイントが生成するリモート関数呼び出しのメタデータは、それを呼び出す既存の関数モジュールのリポジトリーにすでに存在する。宛先エンドポイントが呼び出す呼び出しのメタデータ。そのため、SAP コンポーネントには設定は必要ありません。

一方、サーバーエンドポイントによって処理される関数呼び出しのメタデータは通常 SAP システム

のリポジトリーに存在しないため、代わりに SAP コンポーネントにあるリポジトリーを指定する必要があります。SAP コンポーネントは、名前付きメタデータリポジトリーのマップを維持します。リポジトリーの名前は、メタデータを提供するサーバー名に対応します。

269.2. 設定

SAP コンポーネントは、宛先データ、サーバーデータ、およびリポジトリーデータを保存する 3 つのマップを維持します。宛先データストア と サーバーデータストア は、特別な設定オブジェクト SapConnectionConfiguration で設定され、自動的に SAP コンポーネント（Blueprint XML 設定や Spring XML 設定ファイルのコンテキスト）に挿入されます。リポジトリーデータストア は、関連する SAP コンポーネントで直接設定する必要があります。

269.2.1. 設定の概要

概要

SAP コンポーネントは、宛先データ、サーバーデータ、およびリポジトリーデータを保存する 3 つのマップを維持します。コンポーネントのプロパティー destinationDataStore は、宛先名、プロパティー serverDataStore によって指定された宛先データキーが格納され、サーバー名とプロパティー repositoryDataStore によってキーが設定されたサーバーデータを保存し、リポジトリー名で鍵されたリポジトリーデータを保存します。これらの設定は、初期化中にコンポーネントに渡す必要があります。

例

以下の例は、Blueprint XML ファイルに宛先データストアのサンプルとサーバーデータストアを設定する方法を示しています。sap-configuration Bean（ SapConnectionConfigurationタイプ）は、この XML ファイルで使用される SAP コンポーネントに自動的に注入されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Configures the Inbound and Outbound SAP Connections -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="destinationDataStore">
            <map>
                <entry key="quickstartDest" value-ref="quickstartDestinationData" />
            </map>
        </property>
        <property name="serverDataStore">
            <map>
                <entry key="quickstartServer" value-ref="quickstartServerData" />
            </map>
        </property>
    </bean>
```

```
    <!-- Configures an Outbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment *** -->
    <bean id="quickstartDestinationData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl">
        <property name="ashost" value="example.com" />
        <property name="sysnr" value="00" />
        <property name="client" value="000" />
        <property name="user" value="username" />
        <property name="passwd" value="passowrd" />
        <property name="lang" value="en" />
    </bean>

    <!-- Configures an Inbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment ** -->
    <bean id="quickstartServerData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.ServerDataImpl">
        <property name="gwhost" value="example.com" />
        <property name="gwserv" value="3300" />
        <!-- The following property values should not be changed -->
        <property name="progid" value="QUICKSTART" />
        <property name="repositoryDestination" value="quickstartDest" />
        <property name="connectionCount" value="2" />
    </bean>
</blueprint>
```

269.2.2. 宛先設定

概要

宛先の設定は、**SAP** コンポーネントの **destinationDataStore** プロパティーで維持されます。このマップの各エントリーは、**SAP** インスタンスへの個別のアウトバウンド接続を設定します。各エントリーのキーはアウトバウンド接続の名前であり、「**URI format**」セクションで説明されているように、宛先エンドポイント **URI** の **destinationName** コンポーネントで使用されます。

各エントリーの値は、アウトバウンド **SAP** 接続の設定を指定する宛先データ設定オブジェクト(**org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl**)です。

宛先設定のサンプル

以下の **Blueprint XML** コードは、**quickstartDest** という名前のサンプル宛先を設定する方法を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Create interceptor to support tRFC processing -->
    <bean id="currentProcessorDefinitionInterceptor"
          class="org.fusesource.camel.component.sap.CurrentProcessorDefinitionInterceptStrategy" />
```

```
    <!-- Configures the Inbound and Outbound SAP Connections -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="destinationDataStore">
            <map>
                <entry key="quickstartDest" value-ref="quickstartDestinationData" />
            </map>
        </property>
    </bean>

    <!-- Configures an Outbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment *** -->
    <bean id="quickstartDestinationData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl">
        <property name="ashost" value="example.com" />
        <property name="sysnr" value="00" />
        <property name="client" value="000" />
        <property name="user" value="username" />
        <property name="passwd" value="password" />
        <property name="lang" value="en" />
    </bean>

</blueprint>
```

たとえば、前述の Blueprint XML ファイルにあるように宛先を設定したら、以下の URI を使用して quickstartDest 宛先で BAPI_FLCUST_GETLIST リモート関数呼び出しを呼び出すことができます。

```
sap-srfc-destination:quickstartDest:BAPI_FLCUST_GETLIST
```

tRFC および qRFC 宛先のインターセプター

前述の宛先設定のサンプルは、CurrentProcessorDefinitionInterceptStrategy オブジェクトのインスタンス化を示しています。このオブジェクトは Camel ランタイムにインターセプターをインストールします。これにより、RFC トランザクションの処理中に Camel SAP コンポーネントが Camel ルート内での位置を追跡できます。詳細は、「トランザクション RFC 宛先エンドポイント」 を参照してください。

重要

このインターセプターは、トランザクション RFC 宛先エンドポイント（sap-trfc-destination、sap-qrfc-destinationなど）で重要であり、アウトバウンドトランザクション RFC 通信を適切に管理するために Camel ランタイムにインストールする必要があります。Destination RFC Transaction Handlers は、ストラテジーが実行時に見つからない場合に警告を発行し、この場合に Camel ランタイムを再プロビジョニングして再起動し、アウトバウンドトランザクション RFC 通信を適切に管理する必要があります。

ログインオプションおよび認証オプション

以下の表は、SAP 宛先データストアで宛先を設定するための ログオンオプションおよび認証 オプションを示しています。

Name	デフォルト値	説明
client		SAP クライアント、必須のログオンパラメーター
user		Logon user, logon parameter for password based authentication（パスワードベースの認証の logon パラメーター）
aliasUser		Logon ユーザーエイリアス（ログオンユーザーの代わりに使用可能）
userId		ABAP AS へのログオンに使用されるユーザー ID。JCo ランタイムによって使用されます。宛先設定が認証に SSO/assertion チケット、証明書、現在のユーザー、または SNC 環境を使用する場合に使用されます。ユーザーやユーザーエイリアスが設定されていない場合は、ユーザー ID が必須です。この ID は SAP バックエンドに送信されず、JCo ランタイムによってローカルに使用されます。
passwd		ログインパスワード、パスワードベースの認証の logon パラメーター
lang		Logon 言語 - 定義されていない場合は、デフォルトのユーザー言語が使用されます。
mysapsso2		SSO ベースの認証のログオンチケットとして、指定された SAP Cookie Version 2 を使用します。
x509cert		証明書ベースの認証に、指定した X509 証明書を使用します。
lcheck		最初の呼び出しまで認証を延期します - 1(enable)。特殊なケースでのみ使用されます。

useSapGui		Use a visible, hidden, or do not use SAP GUI（表示、非表示、または SAP GUI を使用しない）の使用
codePage		logon パラメーターの変換に使用されるコードページを定義するための追加の logon パラメーター。特殊なケースでのみ使用
getsso2		ログイン後 SSO チケットの順序により、取得したチケットは宛先属性で利用可能になります。
denyInitialPassword		**1** に設定すると、初期パスワードを使用すると例外が発生します（デフォルトは **0**です）。

接続オプション

以下の表は、SAP 宛先データストアで宛先を設定するための 接続 オプションを示しています。

Name	デフォルト値	説明
saprouter		SAP ルーターの背後にあるシステムに接続するための SAP ルーター文字列。SAP Router 文字列には、SAP ルーターとそのポート番号のチェーンが含まれ、形式：**(/H/<host>[/S/<port>])+**
sysnr		SAP ABAP アプリケーションサーバーのシステム番号（直接接続に必要）
ashost		直接接続に必須、SAP ABAP アプリケーションサーバー
mshost		SAP メッセージサーバー（負荷分散接続に必須のプロパティー）

msserv		SAP メッセージサーバーポート、ロードバランシング接続の任意のプロパティー。サービス名 sapmsXXX を解決するために、**etc/services** 内のルックアップは、オペレーティングシステムのネットワーク層により実行されます。シンボリックサービス名の代わりにポート番号を使用する場合には、検索が行われず、追加のエントリーは必要ありません。
gwhost		アプリケーションサーバーへの接続を確立するために使用する 具体的なゲートウェイ の指定を許可します。指定のない場合は、アプリケーションサーバーのゲートウェイが使用されます。
gwserv		gwhost を使用する場合は設定する必要があります。そのゲートウェイで使用するポートの指定を許可します。指定のない場合は、アプリケーションサーバーのゲートウェイポートが使用されます。サービス名 sapgwXXX を解決するには、オペレーティングシステムのネットワーク層により etc/services でルックアップが実行されます。シンボリックサービス名の代わりにポート番号を使用する場合には、検索が行われず、追加のエントリーは必要ありません。
r3name		SAP システムのシステム ID。ロードバランシング接続の必須プロパティーです。
group		SAP アプリケーションサーバーのグループ（ロードバランシング接続に必須プロパティー）

接続プールのオプション

以下の表は、SAP 宛先データストアで宛先を設定するための 接続プール オプションを示しています。

Name	**デフォルト値**	**説明**

peakLimit	**0**	宛先に同時に作成できるアクティブなアウトバウンド接続の最大数。値が **0** の場合は、アクティブな接続の数が無制限になります。値が **jpoolCapacity** の値よりも小さい場合は、この値に自動で増加します。デフォルト設定は **poolCapacity** の値、**poolCapacity** の指定も指定されていない場合、デフォルトは **0**（無制限）です。
poolCapacity	**1**	宛先によって開放されているアイドルアウトバウンド接続の最大数。**0** の値は、接続プールが存在しないことを意味します（デフォルトは **1**です）。
expirationTime		宛先によって内部に保持される空き接続が閉じられるまでの時間（ミリ秒単位）。
expirationPeriod		宛先がリリースした接続の有効期限をチェックする期間（ミリ秒単位）。
maxGetTime		アプリケーションによって許可される最大接続数がすでに割り当てられている場合、接続を待つ最大時間（ミリ秒単位）。

セキュアなネットワーク接続オプション

以下の表は、SAP 宛先データストアで宛先を設定するための セキュアなネットワーク オプションを示しています。

Name	デフォルト値	説明
sncMode		Secure network connection(SNC)モード、**0** (off)または **1** (on)
sncPartnername		SNC パートナー（例：**p:CN=R3, O=XYZ-INC, C=EN**）

sncQop		SNC レベルのセキュリティー： **1** から **9**
sncMyname		独自の SNC 名。環境設定の上書き
sncLibrary		SNC サービスを提供するライブラリーへのパス

リポジトリーオプション

以下の表は、SAP 宛先データストアで宛先を設定するための リポジトリー オプションを示しています。

Name	**デフォルト値**	**説明**
repositoryDest		リポジトリーとして使用する宛先を指定します。
repositoryUser		リポジトリーの宛先が設定されていない場合、このプロパティーはリポジトリー呼び出しのユーザーとして使用されます。これにより、リポジトリー検索に別のユーザーを使用できます。
repositoryPasswd		リポジトリーユーザーのパスワード。リポジトリーユーザーを使用する場合は必須です。
repositorySnc		**（オプション）** SNC をこの宛先に使用する場合は、このプロパティーが **0** に設定されている場合は、リポジトリー接続に対してオフにすることができます。デフォルト設定は **jco.client.snc**_mode の値です。特殊なケースのみ。

repositoryRoundtripOptimization		**RFC_METADATA_GET** API を有効にします。これは、1 つのラウンドトリップでリポジトリーデータを提供します。 **1** ABAP システムで **RFC_METADATA_GET** の使用をアクティベートします。 **0** ABAP システムで **RFC_METADATA_GET** を無効にします。 プロパティーが設定されていない場合、宛先は最初に **RFC_METADATA_GET** が利用可能かどうかを確認するためのリモート呼び出しを行います。これが利用可能な場合、宛先はこれを使用します。 **注記：** リポジトリーがすでに初期化されている場合は（たとえば、他の宛先で使用されているため）、このプロパティーには影響がありません。通常、このプロパティーは ABAP System に関連し、同じ ABAP システムを指すすべての宛先に同じ値である必要があります。バックエンドの前提条件については、注記 1456826 を参照してください。

トレース設定オプション

以下の表は、SAP 宛先データストアで宛先を 設定するためのトレース設定 オプションを示しています。

Name	デフォルト値	説明
trace		RFC トレースの有効化/無効化（**0** または **1**）
cpicTrace		CPIC トレースの有効化/無効化 **[0..3]**

269.2.3. サーバー設定

概要

サーバーの設定は、SAP コンポーネントの serverDataStore プロパティーで維持されます。このマップの各エントリーは、SAP インスタンスとは異なる受信接続を設定します。各エントリーのキーはアウトバウンド接続の名前であり、「URI format」セクションで説明されているように、サーバーエンドポイント URI の serverName コンポーネントで使用されます。

各エントリーの値は、インバウンド SAP 接続の設定を定義するサーバーデータ設定オブジェクト org.fusesource.camel.component.sap.model.rfc.impl.ServerDataImpl です。

サーバー設定例

以下の Blueprint XML コードは、quickstartServer という名前のサンプルサーバー設定を作成する方法を説明します。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Configures the Inbound and Outbound SAP Connections -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="destinationDataStore">
            <map>
                <entry key="quickstartDest" value-ref="quickstartDestinationData" />
            </map>
        </property>
        <property name="serverDataStore">
            <map>
                <entry key="quickstartServer" value-ref="quickstartServerData" />
            </map>
        </property>
    </bean>

    <!-- Configures an Outbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment *** -->
    <bean id="quickstartDestinationData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl">
        <property name="ashost" value="example.com" />
        <property name="sysnr" value="00" />
        <property name="client" value="000" />
        <property name="user" value="username" />
        <property name="passwd" value="passowrd" />
        <property name="lang" value="en" />
    </bean>

    <!-- Configures an Inbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment ** -->
    <bean id="quickstartServerData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.ServerDataImpl">
        <property name="gwhost" value="example.com" />
```

```
        <property name="gwserv" value="3300" />
        <!-- The following property values should not be changed -->
        <property name="progid" value="QUICKSTART" />
        <property name="repositoryDestination" value="quickstartDest" />
        <property name="connectionCount" value="2" />
    </bean>
</blueprint>
```

この例では、サーバーがリモートの SAP インスタンスからメタデータを取得するのに使用する宛先接続 quickstartDest も設定します。この宛先は、repositoryDestination オプションを使用してサーバーデータで設定されます。このオプションを設定しない場合は、代わりにローカルのメタデータリポジトリーを作成する必要があります（「リポジトリーの設定」を参照）。

たとえば、前述の Blueprint XML ファイルに示されているように宛先を設定したら、以下の URI を使用して、呼び出したクライアントから BAPI_FLCUST_GETLIST リモート関数呼び出しを処理することができます。

```
sap-srfc-server:quickstartServer:BAPI_FLCUST_GETLIST
```

必須オプション

サーバー設定オブジェクトに必要なオプションは以下のとおりです。

Name	デフォルト値	説明
gwhost		サーバー接続を登録するゲートウェイホスト。
gwserv		ゲートウェイサービス。登録が可能なポートです。サービス名 **sapgwXXX** を解決するために、**etc/services** のルックアップは、オペレーティングシステムのネットワーク層により実行されます。シンボリックサービス名の代わりにポート番号を使用する場合には、検索が行われず、追加のエントリーは必要ありません。
progid		登録を行うプログラム ID。ゲートウェイ上の識別子および ABAP システムの宛先で機能します。

repositoryDestination		リモート SAP サーバーでホストされるメタデータリポジトリーからメタデータを取得するためにサーバーが使用できる宛先名を指定します。
connectionCount		ゲートウェイで登録されるべき接続の数。

セキュアなネットワーク接続オプション

サーバーデータ設定オブジェクトのセキュアなネットワーク接続オプションは以下のとおりです。

Name	デフォルト値	説明
sncMode		Secure network connection(SNC) モード、**0** (off)または **1** (on)
sncQop		SNC レベル、**1** から **9**
sncMyname		サーバーの SNC 名。デフォルトの SNC 名を上書きします。通常は、**p:CN=JCoServer、O=ACompany、C=EN** のように表示されます。
sncLib		SNC サービスを提供するライブラリーへのパス。このプロパティーが指定されていない場合は、代わりに **jco.middleware.snc_lib** プロパティーの値が使用されます。

その他のオプション

サーバーデータ設定オブジェクトの他のオプションは以下のとおりです。

Name	デフォルト値	説明

saprouter		ファイアウォールで保護されるシステムに使用する SAP ルーター文字列。そのため、ABAP システムのゲートウェイでサーバーを登録する場合にのみ SAProuter からアクセスできます。一般的なルーター文字列は **/H/firewall.hostname/H/**です。
maxStartupDelay		失敗時の起動試行の最大時間（秒単位）。待機時間は、起動に失敗した後に、最大値に達するまで、またはサーバーを正常に起動できるまで、最初の 1 秒から 2 倍になります。
trace		RFC トレースの有効化/無効化（**0** または **1**）
workerThreadCount		サーバー接続によって使用されるスレッドの最大数。設定されていない場合、**connectionCount** の値は **workerThreadCount** として使用されます。最大スレッド数は 99 を超えることができません。
workerThreadMinCount		サーバー接続によって使用されるスレッドの最小数。設定されていない場合、**connectionCount** の値は **workerThreadMinCount** として使用されます。

269.2.4. リポジトリーの設定

概要

リポジトリーの設定は、SAP コンポーネントの repositoryDataStore プロパティーで維持されます。このマップの各エントリーは、個別のリポジトリーを設定します。各エントリーのキーはリポジトリーの名前であり、このキーはこのリポジトリーが接続されるサーバー名にも対応します。

各エントリーの値は、メタデータリポジトリーの内容を定義するリポジトリーデータ設定オブジェクト org.fusesource.camel.component.sap.model.rfc.impl.RepositoryDataImpl です。リポジトリーデータオブジェクトは、関数テンプレート設定オブジェクト org.fuesource.camel.component.sap.model.rfc.impl.FunctionTemplateImpl のマップです。この

マップの各エントリーは、関数モジュールのインターフェースを指定し、各エントリーのキーは指定された function モジュールの名前です。

リポジトリーデータの例

以下のコードは、meta-data リポジトリーを設定する簡単な例を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Configures the sap-srfc-server component -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="repositoryDataStore">
            <map>
                <entry key="nplServer" value-ref="nplRepositoryData" />
            </map>
        </property>
    </bean>

    <!-- Configures a Meta-Data Repository -->
    <bean id="nplRepositoryData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.RepositoryDataImpl">
        <property name="functionTemplates">
            <map>
                <entry key="BOOK_FLIGHT" value-ref="bookFlightFunctionTemplate" />
            </map>
        </property>
    </bean>
    ...
</blueprint>
```

関数テンプレートプロパティー

関数モジュールのインターフェースは、RFC 呼び出しの function モジュールにデータを転送する 4 つのパラメーターリストで構成されます。各パラメーター一覧は 1 つ以上のフィールドで構成されており、各フィールドは RFC 呼び出しで転送される名前付きパラメーターです。以下のパラメーター list および 例外の一覧がサポートされます。

- **import** *パラメーター一覧 には、RFC 呼び出しの function モジュールに送信されるパラメーター値が含まれます。*
- **export** *パラメーターリスト には、RFC 呼び出しの関数モジュールによって返されるパラメーター値が含まれます。*
-

変更パラメーターリスト には、RFC 呼び出しの function モジュールに送信され、返されるパラメーター値が含まれます。

- *table パラメーターリスト には、RFC 呼び出しの function モジュールに送信され、返される内部テーブル値が含まれます。*

- *関数モジュールのインターフェースは、RFC 呼び出しでモジュールが呼び出されると発生する ABAP 例外の例外リスト も含まれています。*

関数テンプレートは、関数インターフェースの各パラメーター一覧のパラメーターおよびタイプ、および関数によってスローされた ABAP 例外を記述します。関数テンプレートオブジェクトは、以下の表で説明されているように、メタデータオブジェクトの 5 つのプロパティー一覧を維持します。

プロパティー	説明
importParameterList	リストフィールドメタデータオブジェクト(**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**)の一覧。RFC 呼び出しで関数モジュールに送信されるパラメーターを指定します。
changingParameterList	リストフィールドメタデータオブジェクト(**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**)の一覧。関数モジュールとの間で RFC 呼び出しで送受信されるパラメーターを指定します。
exportParameterList	リストフィールドメタデータオブジェクト(**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**)の一覧。関数モジュールから RFC 呼び出しで返されるパラメーターを指定します。
tableParameterList	リストフィールドメタデータオブジェクト(**org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**)の一覧。関数モジュールとの間で RFC 呼び出しで送受信されるテーブルパラメーターを指定します。
exceptionList	ABAP 例外メタデータオブジェクト(**org.fusesource.camel.component.sap.model.rfc.impl.AbapExceptionImpl**)の一覧。関数モジュールの RFC 呼び出しで発生する可能性がある ABAP 例外を指定します。

関数テンプレートの例

以下の例は、関数テンプレートの設定方法の概要を示しています。

```
<bean id="bookFlightFunctionTemplate"
    class="org.fusesource.camel.component.sap.model.rfc.impl.FunctionTemplateImpl">
    <property name="importParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="changingParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="exportParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="tableParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="exceptionList">
        <list>
            ...
        </list>
    </property>
</bean>
```

フィールドメタデータプロパティーの一覧を表示します。

リストフィールドの ***meta-data*** *オブジェクト(* ***org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl*** *)は、パラメーター一覧のフィールドの名前とタイプを指定します。要素的なパラメーターフィールド（**CHAR**、**DATE**、**BCD**、**TIME TIME**、**BYTE**、**NUM**、**FLOAT**、**INT**、**INT1**、**INT1**、**INT2**、**DECF16**、**DECF34**、文字列）の場合、以下の表は、リストフィールドメタデータオブジェクトに設定できる設定プロパティーを一覧表示しています。*

Name	デフォルト値	説明
name	-	パラメーターフィールドの名前。
type	-	フィールドのパラメータータイプ。

byteLength	-	非Unicodeレイアウトのフィールドの長さ（バイト単位）。この値は、パラメーターのタイプによって異なります。「RFC のメッセージボディー」 を参照してください。
unicodeByteLength	-	Unicode レイアウトのフィールドの長さ（バイト単位）。この値は、パラメーターのタイプによって異なります。「RFC のメッセージボディー」 を参照してください。
decimals	**0**	フィールド値の 10 進数の数。パラメータータイプ BCD および FLOAT にのみ必要です。「RFC のメッセージボディー」 を参照してください。
任意	**false**	**true** の場合、このフィールドはオプションで、RFC 呼び出しで設定する必要はありません。

すべての要素的なパラメーターフィールドでは、フィールドの meta-data オブジェクトで 名前、タイプ、バイト長、および unicodeByteLength プロパティーを指定する必要があることに注意してください。さらに、BCD、FLOAT、DECF16、および DECF34 の各フィールドでは、10 進数のプロパティーをフィールド meta-data オブジェクトで指定する必要があります。

TABLE または STRUCTURE タイプの複雑なパラメーターフィールドの場合、以下の表は、リストフィールドメタデータオブジェクトに設定できる設定プロパティーを示しています。

Name	デフォルト値	説明
name	-	パラメーターフィールドの名前
type	-	フィールドのパラメータータイプ
recordMetaData	-	構造またはテーブルのメタデータ。レコードメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl** を渡して、構造またはテーブル行にフィールドを指定します。

任意	**false**	**true** の場合、このフィールドはオプションで、RFC 呼び出しで設定する必要はありません。

すべての複雑なパラメーターフィールドでは、名前、type、および recordMetaData プロパティーをフィールド meta-data オブジェクトで指定する必要があります。recordMetaData プロパティーの値は、ネスト化された構造の構造を指定するレコードフィールド meta-data オブジェクトの org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl です。

要素リストフィールドメタデータの例

以下のメタデータ設定は、TICKET_PRICE という名前の 2 進数の場所で、24 桁のパッシングされた BCD 番号パラメーターを指定します。

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMetaDataImpl">
   <property name="name" value="TICKET_PRICE" />
   <property name="type" value="BCD" />
   <property name="byteLength" value="12" />
   <property name="unicodeByteLength" value="24" />
   <property name="decimals" value="2" />
   <property name="optional" value="true" />
</bean>
```

複合リストフィールドメタデータの例

以下の meta-data 設定は、connectionInfo レコードのメタデータオブジェクトで指定される行構造を持つ CONNINFO という名前の必要な TABLE パラメーターを指定します。

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMetaDataImpl">
   <property name="name" value="CONNINFO" />
   <property name="type" value="TABLE" />
   <property name="recordMetaData" ref="connectionInfo" />
</bean>
```

レコードメタデータプロパティー

レコードメタデータオブジェクト org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl は、ネスト化された STRUCTURE または TABLE パラメーターの名前および内容を指定します。レコードメタデータオブジェクトは、ネスト化された構造またはテーブル行にあるパラメーターを指定するレコードフィールドメタデータデータオブジェクト(org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl)の一覧を維持します。

以下の表は、レコードメタデータオブジェクトに設定できる設定プロパティーの一覧です。

Name	デフォルト値	説明
name	-	レコードの名前。
recordFieldMetaData	-	レコードフィールドメタデータオブジェクト(**org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl**)の一覧。構造に含まれるフィールドを指定します。

注記

record meta-data オブジェクトのすべてのプロパティーが必要です。

レコードメタデータの例

以下の例は、レコードメタデータオブジェクトの設定方法を示しています。

```
<bean    id="connectionInfo"
        class="org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl">
    <property name="name" value="CONNECTION_INFO" />
    <property name="recordFieldMetaData">
        <list>
            ...
        </list>
    </property>
</bean>
```

レコードフィールドメタデータプロパティー

*レコードフィールドの meta-data オブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl** は、構造を持つパラメーターフィールドの名前およびタイプを指定します。*

レコードフィールドメタデータオブジェクトはパラメーターフィールドの meta-data オブジェクトと似ていますが、ネストされた構造やテーブル行内の個々のフィールドの場所のオフセットは追加で指定する必要があります。個々のフィールドのユニコード以外のオフセットと Unicode オフセットは、構造または行の前のフィールドの非Unicodeおよび Unicode バイト長の合計から計算して指定する必要があります。ネストされた構造でフィールドのオフセットを適切に指定できず、テーブル行により、

基礎となる JCo および ABAP ランタイムのパラメーターのフィールドストレージが重複し、RFC 呼び出しの値が適切に転送されないことに注意してください。

CHAR,DATE,BCD,TIME,BYTE,NUM,FLOAT, INTINT1 INT1 INT2INT 2 DECF16,DECF34,STRING,XSTRING）の場合、以下の表は、レコードフィールドメタデータオブジェクトに設定できる設定プロパティーを一覧表示しています。

Name	デフォルト値	説明
name	-	パラメーターフィールドの名前
type	-	フィールドのパラメータータイプ
byteLength	-	非Unicodeレイアウトのフィールドの長さ（バイト単位）。この値は、パラメーターのタイプによって異なります。「RFC のメッセージボディー」 を参照してください。
unicodeByteLength	-	Unicode レイアウトのフィールドの長さ（バイト単位）。この値は、パラメーターのタイプによって異なります。「RFC のメッセージボディー」 を参照してください。
byteOffset	-	非Unicodeレイアウトのバイト単位のフィールドオフセット。このオフセットは、エンクロージング構造内のフィールドのバイトロケーションです。
unicodeByteOffset	-	Unicode レイアウトのフィールドオフセット（バイト単位）。このオフセットは、エンクロージング構造内のフィールドのバイトロケーションです。
decimals	**0**	フィールド値の 10 進数の数。パラメータータイプ **BCD** および **FLOAT** にのみ必要です。「RFC のメッセージボディー」 を参照してください。

TABLE または STRUCTURE タイプの複雑なパラメーターフィールドの場合、以下の表は、レコードフィールドメタデータオブジェクトに設定できる設定プロパティーを示しています。

Name	デフォルト値	説明
name	-	パラメーターフィールドの名前
type	-	フィールドのパラメータータイプ
byteOffset	-	非Unicodeレイアウトのバイト単位のフィールドオフセット。このオフセットは、エンクロージング構造内のフィールドのバイトロケーションです。
unicodeByteOffset	-	Unicode レイアウトのフィールドオフセット（バイト単位）。このオフセットは、エンクロージング構造内のフィールドのバイトロケーションです。
recordMetaData	-	構造またはテーブルのメタデータ。レコードメタデータオブジェクト **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl** を渡して、構造またはテーブル行にフィールドを指定します。

要素レコードフィールドメタデータの例

以下のメタデータ構成は、ARR DATE という名前の DATE フィールドパラメーターが、Unicode 以外のレイアウトの場合は 85 バイトをエンクロージング構造に指定し、Unicode レイアウトの場合の囲み構造に 170 バイトを指定します。

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl">
    <property name="name" value="ARRDATE" />
    <property name="type" value="DATE" />
    <property name="byteLength" value="8" />
    <property name="unicodeByteLength" value="16" />
    <property name="byteOffset" value="85" />
    <property name="unicodeByteOffset" value="170" />
</bean>
```

複雑なレコードフィールドメタデータの例

以下の meta-data 設定は、flightInfo record meta-data オブジェクトで指定された構造を持つ FLTINFO という名前の STRUCTURE フィールドパラメーターを指定します。パラメーターは、非 Unicode レイアウトと Unicode レイアウトの両方で、囲み構造の先頭に置かれます。

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl">
   <property name="name" value="FLTINFO" />
   <property name="type" value="STRUCTURE" />
   <property name="byteOffset" value="0" />
   <property name="unicodeByteOffset" value="0" />
   <property name="recordMetaData" ref="flightInfo" />
</bean>
```

269.3. メッセージヘッダー

SAP コンポーネントは、以下のメッセージヘッダーをサポートします。

ヘッダー	説明
CamelSap.scheme	メッセージを処理する最後のエンドポイントの URI スキーム。以下のいずれかの値を使用します。 **sap-srfc-destination** **sap-trfc-destination** **sap-qrfc-destination** **sap-srfc-server** **sap-trfc-server** **sap-idoc-destination** **sap-idoclist-destination** **sap-qidoc-destination** **sap-qidoclist-destination** **sap-idoclist-server**
CamelSap.destinationName	メッセージを処理する最後の宛先エンドポイントの宛先名。
CamelSap.serverName	メッセージを処理する最後のサーバーエンドポイントのサーバー名。
CamelSap.queueName	メッセージを処理する最後のキューエンドポイントのキュー名。
CamelSap.rfcName	メッセージを処理する最後の RFC エンドポイントの RFC 名。
CamelSap.idocType	メッセージを処理する最後の IDoc エンドポイントの IDoc タイプ。

CamelSap.idocTypeExtension	IDoc タイプ拡張（ある場合）は、メッセージを処理する最後の IDoc エンドポイントになります。
CamelSap.systemRelease	システムリリース（ある場合）は、メッセージを処理する最後の IDoc エンドポイントになります。
CamelSap.applicationRelease	アプリケーションリリース（ある場合）は、メッセージを処理する最後の IDoc エンドポイントになります。

269.4. エクスチェンジプロパティー

SAP コンポーネントは、以下のエクスチェンジプロパティーを追加します。

プロパティー	説明
CamelSap.destinationPropertiesMap	エクスチェンジによって発生した各 SAP 宛先のプロパティーを含むマップ。マップは宛先名でキーされ、各エントリーはその宛先の設定プロパティーが含まれる **java.util.Properties** オブジェクトです。
CamelSap.serverPropertiesMap	エクスチェンジによって発生した各 SAP サーバーのプロパティーが含まれるマップ。マップはサーバー名によってキーされ、各エントリーはそのサーバーの設定プロパティーが含まれる **java.util.Properties** オブジェクトです。

269.5. RFC のメッセージボディー

要求および応答オブジェクト

SAP エンドポイントは、SAP リクエストオブジェクトを含むメッセージボディーを持つメッセージを受信し、SAP 応答オブジェクトが含まれるメッセージのボディーが含まれるメッセージを返します。SAP リクエストと応答は、名前付きフィールドを含む名前付きフィールドを含むデータ構造を、事前定義したデータタイプを持つ固定です。

SAP リクエストと応答の名前付きフィールドは SAP エンドポイントに固有のもので、SAP リクエストのパラメーターを定義するエンドポイントごとに、受け入れる応答があります。SAP エンドポイントは、それに固有のリクエストおよび応答オブジェクトを作成するためのファクトリーメソッドを提供します。

```
public class SAPEndpoint ... {
    ...
```

```
    public Structure getRequest() throws Exception;

    public Structure getResponse() throws Exception;
    ...
}
```

構造オブジェクト

SAP リクエストおよび応答オブジェクトはいずれも、org.fusesource.camel.component.sap.model.rfc.Structure インターフェースをサポートする構造オブジェクトとして Java で表されます。このインターフェースは、java.util.Map インターフェースと org.eclipse.emf.ecore.EObject インターフェースの両方を拡張します。

```
public interface Structure extends org.eclipse.emf.ecore.EObject,
                                   java.util.Map<String, Object> {

    <T> T get(Object key, Class<T> type);

}
```

構造オブジェクトのフィールド値は、マップインターフェースのフィールドの getter メソッドを介してアクセスされます。さらに、構造インターフェースは、フィールド値を取得するタイプ制限方法を提供します。

構造オブジェクトは、Eclipse Modeling Framework(EMF)を使用してコンポーネントランタイムに実装され、そのフレームワークの EObject インターフェースをサポートします。構造オブジェクトのインスタンスには、提供されるフィールドのマップの構造と内容を定義して制限する meta-data がアタッチされています。このメタデータには、EMF が提供する標準メソッドを使用してアクセスし、イントロスペクションすることができます。詳細は、EMF ドキュメントを参照してください。

注記

構造オブジェクトに定義されていないパラメーターの取得を試みると、null を返します。構造に定義されていないパラメーターの設定を試みると、例外がスローされ、間違ったタイプでパラメーターの値を設定しようとします。

以下のセクションで説明されているように、構造オブジェクトには、複雑なフィールドタイプの値（STRUCTURE および TABLE ）が含まれるフィールドを含めることができます。これらのタイプのインスタンスを作成して構造に追加する必要はないことに注意してください。エンクロージング構造でアクセスする際に必要に応じて、このフィールド値のインスタンスがオンデマンドで作成されます。

フィールドタイプ

SAP リクエストまたはレスポンスの構造オブジェクト内にあるフィールドは、要素的 または 複雑 である場合があります。elementary フィールドにはスカラー値が 1 つ含まれますが、複雑なフィールドには、単数または複雑な型の 1 つ以上のフィールドが含まれます。

要素フィールドタイプ

要素フィールドには、文字、数字、16 進数、または文字列フィールドタイプを使用できます。以下の表は、構造オブジェクトに存在する可能性のある要素フィールドのタイプをまとめたものです。

フィールドタイプ	対応する Java タイプ	Byte Length	Unicode Byte Length	number Decimals Digits	説明
CHAR	**java.lang.String**	1 から 65535	1 から 65535	-	ABAP Type 'C': サイズ化された文字文字列を修正
DATE	**java.util.Date**	8	16	-	ABAP タイプ 'D': 日付（形式：YYYYMMDD）
BCD	**java.math.BigDecimal**	1 から 16	1 から 16	0 から 14	ABAP Type 'P': Packed BCD 番号。BCD 番号には、1 バイトあたり 2 桁の数字が含まれます。
時間	**java.util.Date**	6	12	-	ABAP タイプ 'T': 時間（形式：HHMMSS）
BYTE	**byte[]**	1 から 65535	1 から 65535	-	ABAP Type 'X':Fixed sized byte array
NUM	**java.lang.String**	1 から 65535	1 から 65535	-	ABAP Type 'N': サイズ化された数値文字列の修正
浮動小数点	**java.lang.Double**	8	8	0 から 15	ABAP Type 'F': Floating Point number

INT	**java.lang.Integer**	4	4	-	ABAP Type 'I': 4 バイト整数
INT2	**java.lang.Integer**	2	2	-	ABAP Type 'S': 2 バイト整数
INT1	**java.lang.Integer**	1	1	-	ABAP Type 'B': 1 バイト整数
DECF16	**java.match.BigDecimal**	8	8	16	ABAP Type 'decfloat16': 8 -byte Decimal Floating Point Number
DECF34	**java.math.BigDecimal**	16	16	34	ABAP Type 'decfloat34': 16 バイトデクリアル浮動小数点数
STRING	**java.lang.String**	8	8	-	ABAP Type 'G': 変数の長さの文字列
XSTRING	**byte[]**	8	8	-	ABAP Type 'Y': 変数長バイト配列

文字フィールドタイプ

文字フィールドには、基礎となる JCo および ABAP ランタイムでUnicode 以外の文字エンコーディングを使用できる固定された文字文字列が含まれます。ユニコード以外の文字文字列は、1 バイトごとに 1 文字エンコードされます。Unicode 文字文字列は、UTF-16 エンコーディングを使用して 2 つのバイトでエンコードされます。文字フィールドの値は Java で java.lang.String オブジェクトとして表され、基礎となる JCo ランタイムは ABAP 表現への変換を行います。

文字フィールドは、関連付けられた バイト長と unicodeByteLength プロパティーでフィールドの長さを宣言します。これは、各エンコーディングシステムのフィールドの文字文字列の長さを決定します。

CHAR

CHAR 文字フィールドは、英数字を含むテキストフィールドで、ABAP タイプ C に対応します。

NUM

NUM 文字フィールドは数字のみが含まれる数値テキストフィールドで、ABAP タイプ N に対応します。

DATE

DATE 文字フィールドは 8 文字の日付フィールドで、YYYYMMDD としてフォーマットされた月、日付け、および ABAP タイプ D に対応します。

時間

TIME 文字フィールドは、時間、分、秒が HHMMSS としてフォーマットされ、ABAP タイプ T に対応する 6 文字の時間フィールドです。

数値フィールドタイプ

数値フィールドには数字が含まれます。以下の数値フィールドタイプがサポートされます。

INT

INT 数値フィールドは、基礎となる JCo および ABAP ランタイムの 4 バイトの整数値として保存される整数値で、ABAP タイプ I に対応します。INT フィールドの値は、Java で java.lang.Integer オブジェクトとして表します。

INT2

INT2 数値フィールドは、基礎となる JCo および ABAP ランタイムの 2 バイトの整数値として保存される整数値で、ABAP タイプ S に対応します。INT2 フィールドの値は、Java で java.lang.Integer オブジェクトとして表します。

INT1

INT1 フィールドは、基礎となる JCo および ABAP ランタイム値の 1 バイトの整数値として保存される整数値で、BAAP タイプ B に対応します。INT1 フィールドの値は、Java で java.lang.Integer オブジェクトとして表します。

浮動小数点

FLOAT フィールドは、基礎となる JCo および ABAP ランタイムの 8 バイトの二重値として保存されるバイナリー浮動小数点数フィールドで、ABAP タイプ F に対応します。FLOAT フィールドは、フィールドの値が関連する 10 進数プロパティーに含まれる 10 進数の数字を宣言します。FLOAT フィールドの場合、この 10 進数プロパティーには 1 桁から 15 桁の数字の値を指定できます。FLOAT フィールドの値は、Java で java.lang.Double オブジェクトとして表されます。

BCD

BCD フィールドは、基礎となる JCo および ABAP ランタイムの 1 から 16 バイトパック番号

として保存されたバイナリーコード 10 進数フィールドで、ABAP タイプ P に対応します。パック番号は、1 バイトあたり 10 進数のデータを格納します。BCD フィールドは、関連付けられた バイト長および unicodeByteLength プロパティーでフィールドの長さを宣言します。BCD フィールドの場合、これらのプロパティーは 1 バイトから 16 バイトの間に値を持ち、両方のプロパティーに同じ値が設定されます。BCD フィールドは、フィールドの値が関連付けられた 10 進数プロパティーに含まれる 10 進数の数を宣言します。BCD フィールドの場合、この 10 進数プロパティーには 1 桁から 14 桁の値を使用できます。BCD フィールドの値は、Java で java.math.BigDecimal として表されます。

DECF16

DECF16 フィールドは 10 進数の浮動小数点として、基礎となる JCo および ABAP ランタイムの 8 バイトの IEEE 754 浮動小数点として保存され、ABAP タイプの decfloat16 に対応します。DECF16 フィールドの値には 16 桁の数字があります。DECF16 フィールドの値は、Java では java.math.BigDecimal として表されます。

DECF34

DECF34 フィールドは 10 進数の浮動小数点として、基礎となる JCo および ABAP ランタイムの 16 バイトの IEEE 754 浮動小数点として保存され、ABAP タイプの decfloat34 に対応します。DECF34 フィールドの値には、34 桁の数字があります。DECF34 フィールドの値は、Java では java.math.BigDecimal として表されます。

16 進数のフィールドタイプ

16 進数フィールドには raw バイナリーデータが含まれます。以下の 16 進数のフィールドタイプがサポートされます。

BYTE

BYTE フィールドは、基礎となる JCo および ABAP ランタイムのバイト配列として保存される固定されたバイト文字列で、ABAP タイプ X に対応します。BYTE フィールドは、関連する バイト長および unicodeByteLength プロパティーでフィールドの長さを宣言します。BYTE フィールドの場合、これらのプロパティーの値は 1 から 65535 バイトで、両方のプロパティーの値は同じになります。BYTE フィールドの値は、Java で byte[] オブジェクトとして表されます。

string field types

文字列フィールドは変数長の文字列値を参照します。その文字列値の長さは、ランタイムまで固定されません。文字列値のストレージは、基礎となる JCo および ABAP ランタイムに動的に作成されます。文字列フィールド自体のストレージが修正され、文字列ヘッダーのみが含まれます。

STRING

STRING フィールドは文字文字列を参照し、基盤となる JCo および ABAP ランタイムに 8 バイトの値として保存されます。これは ABAP タイプ G に対応します。STRING フィールドの値は

Java で java.lang.String オブジェクトとして表されます。

XSTRING

XSTRING フィールドはバイト文字列を参照し、基盤となる JCo および ABAP ランタイムに 8 バイトの値として保存されます。これは ABAP タイプ Y に対応します。STRING フィールドの値は Java で byte[] オブジェクトとして表されます。

複雑なフィールドタイプ

複雑なフィールドは structure または table フィールドタイプのいずれかになります。以下の表では、これらの複雑なフィールドタイプをまとめています。

フィールドタイプ	対応する Java タイプ	Byte Length	Unicode Byte Length	number Decimals Digits	説明
構造	**org.fusesource.camel.component.sap.model.rfc.Structure**	個別のフィールドバイト長の合計	個別のフィールド Unicode バイトの長さの合計	-	ABAP Type 'u' & 'v': 異種構造
テーブル	**org.fusesource.camel.component.sap.model.rfc.Table**	行構造のバイト長	行構造の Unicode バイト長	-	ABAP Type 'h': Table

構造のフィールド型

STRUCTURE フィールドには構造オブジェクトが含まれ、BAAP 構造レコードとして基礎となる JCo および ABAP ランタイムに保存されます。ABAP タイプの u または v に対応します。STRUCTURE フィールドの値は、Java で org.fusesource.camel.component.sap.model.rfc.Structure のインターフェースを持つ構造オブジェクトとして表現されます。

テーブルフィールドタイプ

TABLE フィールドには、テーブルオブジェクトが含まれており、BAAP 内部テーブルとして基礎となる JCo および ABAP ランタイムに保存されます。ABAP タイプ h に対応します。フィールドの値は、テーブルオブジェクトにより、org.fusesource.camel.component.sap.model.rfc.Table のインターフェースを持つテーブルオブジェクトによって Java で表されます。

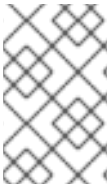
テーブルオブジェクト

テーブルオブジェクトは、同じ構造を持つ構造オブジェクトの行が含まれる同種のリストデータ構造です。このインターフェースは、**java.util.List** インターフェースと **org.eclipse.emf.ecore.EObject** インターフェースの両方を拡張します。

```
public interface Table<S extends Structure>
    extends org.eclipse.emf.ecore.EObject,
    java.util.List<S> {

    /**
     * Creates and adds table row at end of row list
     */
    S add();

    /**
     * Creates and adds table row at index in row list
     */
    S add(int index);

}
```

テーブルオブジェクトの行のリストは、リストインターフェースで定義された標準のメソッドを使用してアクセスおよび管理されます。テーブルインターフェース以外に、構造オブジェクトを行リストに追加する **2** つのファクトリーメソッドが提供されます。

テーブルオブジェクトは、**Eclipse Modeling Framework(EMF)**を使用してコンポーネントランタイムに実装され、そのフレームワークの **EObject** インターフェースをサポートします。テーブルオブジェクトのインスタンスには、提供される行の構造と内容を定義して制限する **meta-data** がアタッチされています。このメタデータには、**EMF** が提供する標準メソッドを使用してアクセスし、イントロスペクションすることができます。詳細は、**EMF** ドキュメントを参照してください。



注記

誤ったタイプの行構造の値を追加または設定しようとすると、例外が発生します。

269.6. IDOC のメッセージボディー

IDoc メッセージタイプ

IDoc Camel SAP エンドポイントの **1** つを使用する場合、メッセージボディーのタイプは使用している特定のエンドポイントによって異なります。

sap-idoc-destination エンドポイントまたは sap-qidoc-destination エンドポイントの場合、メッセージのボディーは Document タイプになります。

```
org.fusesource.camel.component.sap.model.idoc.Document
```

sap-idoclist-destination エンドポイント、sap-qidoclist-destination エンドポイント、または sap-idoclist-server エンドポイントの場合、メッセージボディーは DocumentList タイプになります。

```
org.fusesource.camel.component.sap.model.idoc.DocumentList
```

IDoc ドキュメントモデル

Camel SAP コンポーネントでは、IDoc のドキュメントは、基礎となる SAP IDoc API に関連したラッパー API を提供する Eclipse Modelling Framework(EMF)を使用してモデル化されます。このモデルで最も重要なタイプは、以下のとおりです。

```
org.fusesource.camel.component.sap.model.idoc.Document
org.fusesource.camel.component.sap.model.idoc.Segment
```

Document タイプは、IDoc ドキュメントインスタンスを表します。まとめると、Document インターフェースは以下のメソッドを公開します。

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface Document extends EObject {
    // Access the field values from the IDoc control record
    String getArchiveKey();
    void setArchiveKey(String value);
    String getClient();
    void setClient(String value);
    ...

    // Access the IDoc document contents
    Segment getRootSegment();
}
```

Document インターフェースでは、以下のようなメソッドが公開されます。

コントロールレコードにアクセスする方法

ほとんどのメソッドは、IDoc コントロールレコードのフィールド値にアクセスまたは変更するためのものです。これらのメソッドは **AttributeName, AttributeName** *の形式です。*

attributeName はフィールド値の名前です（表269.2「IDoc ドキュメント属性」を参照してください）。

ドキュメントコンテンツにアクセスする方法

getRootSegment メソッドは、ドキュメントコンテンツ（IDoc データレコード）へのアクセスを提供し、そのコンテンツを Segment オブジェクトとして返します。各 セグメント には任意の数の子セグメントを含めることができ、セグメントは任意のレベルにネストできます。

ただし、セグメント階層の正確なレイアウトは、ドキュメントの特定の IDoc タイプ で定義されている点に注意してください。したがって、セグメント階層を作成（または読み取り）する場合、IDoc タイプで定義される正確な構造に従う必要があります。

Segment タイプは、IDoc ドキュメントのデータレコードにアクセスするために使用されます。ここでは、セグメントはドキュメントの IDoc タイプで定義される構造に従ってレイアウトされます。要約すると、Segment インターフェースは以下のメソッドを公開します。

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface Segment extends EObject, java.util.Map<String, Object> {
   // Returns the value of the '<em><b>Parent</b></em>' reference.
   Segment getParent();

   // Return a immutable list of all child segments
   <S extends Segment> EList<S> getChildren();

   // Returns a list of child segments of the specified segment type.
   <S extends Segment> SegmentList<S> getChildren(String segmentType);

   EList<String> getTypes();

   Document getDocument();

   String getDescription();

   String getType();

   String getDefinition();

   int getHierarchyLevel();

   String getIdocType();

   String getIdocTypeExtension();

   String getSystemRelease();

   String getApplicationRelease();
```

```
    int getNumFields();

    long getMaxOccurrence();

    long getMinOccurrence();

    boolean isMandatory();

    boolean isQualified();

    int getRecordLength();

    <T> T get(Object key, Class<T> type);
}
```

getChildren(String segmentType) メソッドは、特に新しい（入れ子）子をセグメントに追加する際に特に便利です。これは、以下のように定義される型 SegmentList のオブジェクトを返します。

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface SegmentList<S extends Segment> extends EObject, EList<S> {
    S add();

    S add(int index);
}
```

したがって、E1SCU_CRE タイプのデータレコードを作成するには、以下のような Java コードを使用できます。

```
Segment rootSegment = document.getRootSegment();

Segment E1SCU_CRE_Segment = rootSegment.getChildren("E1SCU_CRE").add();
```

IDoc がドキュメントオブジェクトにどのように関連しているのか

SAP ドキュメントによると、IDoc ドキュメントは以下の主な部分で構成されます。

コントロールレコード

制御レコード（IDoc ドキュメントのメタデータを含む）は、Document オブジェクト attribute の属性によって表されます。詳細は、表269.2「IDoc ドキュメント属性」 を参照してください。

データレコード

データレコードは、セグメントのネストされた階層として構築される Segment オブジェクトによって表されます。Document.getRootSegment メソッドを使用してルートセグメントにアクセス

できます。

ステータスレコード

Camel SAP コンポーネントでは、ステータスレコードはドキュメントモデルでは表示され ません。ただし、コントロールレコードの status 属性を介して最新の status 値にアクセスできます。

ドキュメントインスタンスの作成例

たとえば、例269.1「Java での IDoc ドキュメントの作成」 は、Java で IDoc モデル API を使用して IDoc タイプ FLCUSTOMER_CREATEFROMDATA01 の IDoc ドキュメントを作成する方法を示しています。

例269.1 Java での IDoc ドキュメントの作成

```
// Java
import org.fusesource.camel.component.sap.model.idoc.Document;
import org.fusesource.camel.component.sap.model.idoc.Segment;
import org.fusesource.camel.component.sap.util.IDocUtil;

import org.fusesource.camel.component.sap.model.idoc.Document;
import org.fusesource.camel.component.sap.model.idoc.DocumentList;
import org.fusesource.camel.component.sap.model.idoc.IdocFactory;
import org.fusesource.camel.component.sap.model.idoc.IdocPackage;
import org.fusesource.camel.component.sap.model.idoc.Segment;
import org.fusesource.camel.component.sap.model.idoc.SegmentChildren;
...
//
// Create a new IDoc instance using the modelling classes
//

// Get the SAP Endpoint bean from the Camel context.
// In this example, it's a 'sap-idoc-destination' endpoint.
SapTransactionalIDocDestinationEndpoint endpoint =
    exchange.getContext().getEndpoint(
        "bean:SapEndpointBeanID",
        SapTransactionalIDocDestinationEndpoint.class
    );

// The endpoint automatically populates some required control record attributes
Document document = endpoint.createDocument()

// Initialize additional control record attributes
document.setMessageType("FLCUSTOMER_CREATEFROMDATA");
document.setRecipientPartnerNumber("QUICKCLNT");
document.setRecipientPartnerType("LS");
document.setSenderPartnerNumber("QUICKSTART");
document.setSenderPartnerType("LS");

Segment rootSegment = document.getRootSegment();
```

```
Segment E1SCU_CRE_Segment = rootSegment.getChildren("E1SCU_CRE").add();

Segment E1BPSCUNEW_Segment =
E1SCU_CRE_Segment.getChildren("E1BPSCUNEW").add();
E1BPSCUNEW_Segment.put("CUSTNAME", "Fred Flintstone");
E1BPSCUNEW_Segment.put("FORM", "Mr.");
E1BPSCUNEW_Segment.put("STREET", "123 Rubble Lane");
E1BPSCUNEW_Segment.put("POSTCODE", "01234");
E1BPSCUNEW_Segment.put("CITY", "Bedrock");
E1BPSCUNEW_Segment.put("COUNTR", "US");
E1BPSCUNEW_Segment.put("PHONE", "800-555-1212");
E1BPSCUNEW_Segment.put("EMAIL", "fred@bedrock.com");
E1BPSCUNEW_Segment.put("CUSTTYPE", "P");
E1BPSCUNEW_Segment.put("DISCOUNT", "005");
E1BPSCUNEW_Segment.put("LANGU", "E");
```

文書化属性

表269.2「IDoc ドキュメント属性」 は、Document オブジェクトに設定できるコントロールレコード属性を示しています。

表269.2 IDoc ドキュメント属性

属性	長さ	SAP フィールド	説明
archiveKey	70	**ARCKEY**	EDI アーカイブキー
client	3	**MANDT**	クライアント
creationDate	8	**CREDAT**	日付 IDoc の作成
creationTime	6	**CRETIM**	時間 IDoc の作成
direction	1	**DIRECT**	方向
eDIMessage	14	**REFMES**	メッセージへの参照
eDIMessageGroup	14	**REFGRP**	メッセージグループへの参照
eDIMessageType	6	**STDMES**	EDI メッセージタイプ
eDIStandardFlag	1	**STD**	EDI 標準
eDIStandardVersion	6	**STDVRS**	EDI 標準のバージョン
eDITransmissionFile	14	**REFINT**	交換ファイルへの参照

属性	長さ	SAP フィールド	説明
iDocCompoundType	8	**DOCTYP**	IDoc タイプ
iDocNumber	16	**DOCNUM**	IDoc 番号
iDocSAPRelease	4	**DOCREL**	IDoc の SAP リリース
iDocType	30	**IDOCTP**	基本的な IDoc タイプの名前
iDocTypeExtension	30	**CIMTYP**	エクステンションタイプの名前
messageCode	3	**MESCOD**	論理メッセージコード
messageFunction	3	**MESFCT**	論理メッセージ関数
messageType	30	**MESTYP**	論理メッセージタイプ
outputMode	1	**OUTMOD**	出力モード
recipientAddress	10	**RCVSAD**	受信者アドレス(SADR)
recipientLogicalAddress	70	**RCVLAD**	受信側の論理アドレス
recipientPartnerFunction	2	**RCVPFC**	レシーバーのパートナー機能
recipientPartnerNumber	10	**RCVPRN**	レシーバーのパートナー番号
recipientPartnerType	2	**RCVPRT**	パートナータイプの受信側
recipientPort	10	**RCVPOR**	レシーバーポート（SAP System、EDI サブシステム）
senderAddress		**SNDSAD**	送信側アドレス(SADR)
senderLogicalAddress	70	**SNDLAD**	送信者の論理アドレス
senderPartnerFunction	2	**SNDPFC**	送信者のパートナー機能

属性	長さ	SAP フィールド	説明
senderPartnerNumber	10	**SNDPRN**	パートナー向け送信者のパートナー数
senderPartnerType	2	**SNDPRT**	パートナーレベルの送信者
senderPort	10	**SNDPOR**	送信者ポート（SAP System、EDI サブシステム）
serialization	20	**SERIAL**	EDI/ALE: Serialization フィールド
status	2	**ステータス**	IDoc のステータス
testFlag	1	**テスト**	test フラグ

Java でのドキュメント属性の設定

Java で制御レコード属性（ 表269.2「IDoc ドキュメント属性」から）を設定する場合、Java Bean プロパティーの通常の規則に従います。つまり、属性値の取得および設定のために getName メソッドおよび setName メソッドを介して name 属性にアクセスできます。たとえば、iDocType 属性、iDocType Extension 属性、および messageType 属性は、Document オブジェクトで以下のように設定できます。

```
// Java
document.setIDocType("FLCUSTOMER_CREATEFROMDATA01");
document.setIDocTypeExtension("");
document.setMessageType("FLCUSTOMER_CREATEFROMDATA");
```

XML でのドキュメント属性の設定

XML でコントロールレコード属性を設定する場合は、idoc:Document 要素に属性を設定する必要があります。たとえば、iDocType 属性、 iDocType Extension 属性、および messageType 属性は以下のように設定できます。

```
<?xml version="1.0" encoding="ASCII"?>
<idoc:Document ...
        iDocType="FLCUSTOMER_CREATEFROMDATA01"
        iDocTypeExtension=""
        messageType="FLCUSTOMER_CREATEFROMDATA" ... >
    ...
</idoc:Document>
```

269.7. トランザクションサポート

BAPI トランザクションモデル

SAP コンポーネント は、SAP を用いたアウトバウンド通信用に BAPI トランザクションモデルをサポートします。トランザクションオプションが true に設定された URL が含まれる宛先エンドポイントの場合、必要な場合は、エンドポイントのアウトバウンド接続でステートフルセッションを開始し、エクスチェンジで Camel Synchronization オブジェクトを登録します。この同期オブジェクトは BAPI サービスメソッド BAPI_TRANSACTION_COMMIT を呼び出し、メッセージエクスチェンジの処理が完了するとステートフルセッションを終了します。メッセージエクスチェンジの処理に失敗した場合、同期オブジェクトは BAPI サーバーメソッド BAPI_TRANSACTION_ROLLBACK を呼び出してステートフルセッションを終了します。

RFC トランザクションモデル

tRFC プロトコルは、一意のトランザクション識別子(TID)で各トランザクション要求を特定し、AT-MOST-ONCE 配信と処理の保証を達成します。TID は、プロトコルで送信される各リクエストを出力します。tRFC プロトコルを使用した送信アプリケーションは、リクエストの送信時に一意の TID で要求の各インスタンスを特定する必要があります。アプリケーションは指定の TID で要求を複数回送信できますが、プロトコルにより、リクエストが受信システムで配信および処理されるようになります。アプリケーションは、要求の送信時に通信またはシステムエラーに遭遇すると、指定の TID でリクエストを再送信することを選択できます。そのため、その要求が受信システムで配信および処理されたかどうかの疑いがあります。通信エラーが発生したときにリクエストを再送信することで、tRFC プロトコルを使用するクライアントアプリケーションが EXACTLY-ONCE 配信と要求に対する保証を処理することができます。

使用するトランザクションモデル

BAPI トランザクションはアプリケーションレベルのトランザクションで、ACID は SAP データベースの BAPI メソッドまたは RFC 機能によって実行される永続データの変更に保証されます。RFC トランザクションは通信トランザクションで、BAPI メソッドおよび RFC 関数への要求に配信保証（AT-MOST-ONCE、EXACTLY-ONCE-IN-ORDER）を課す意味です。

トランザクション RFC 宛先エンドポイント

以下の宛先エンドポイントは RFC トランザクションをサポートします。

- sap-trfc-destination
- sap-qrfc-destination

単一の Camel ルートには、複数のトランザクション RFC 宛先エンドポイントを含めることができ、メッセージを複数の RFC 宛先に送信したり、同じ RFC 宛先にメッセージを送信したりすることもできます。これは、Camel SAP コンポーネントが、ルートを通過する各 Exchange オブジェクトの 多くのトランザクション ID(TID)を追跡する必要があることを意味します。ルートの処理が失敗し、再試行する必要がある場合、状況は非常に複雑になります。RFC トランザクションセマンティクスにより、ルートと共に各 RFC の宛先がルートと共に最初に使用されたものと 同じ TID を使用して呼び出す必要があります（およびそれぞれの宛先の TID は相互に異なる）。つまり、Camel SAP コンポーネントはルート内のどの時点で使用された TID を追跡し、TID が正しい順序で再生できるように、この情報を覚えておく必要があります。

デフォルトでは、Camel は Exchange がルート内のどこにあるかを認識できるようにするメカニズムを提供しません。このようなメカニズムを提供するには、CurrentProcessorDefinitionInterceptStrategy インターセプターを Camel ランタイムにインストールする必要があります。Camel SAP コンポーネントがルートの TID を追跡するには、このインターセプターを Camel ランタイムにインストールする必要があります。インターセプターの設定方法は、「tRFC および qRFC 宛先のインターセプター」 を参照してください。

トランザクション RFC サーバーエンドポイント

以下のサーバーエンドポイントは RFC トランザクションをサポートします。

- sap-trfc-server

トランザクションリクエストを処理する Camel エクスチェンジが処理エラーに遭遇すると、Camel は標準のエラー処理メカニズムを介して処理エラーを処理します。エクスチェンジを処理する Camel ルートがエラーを呼び出し元に伝播するように設定されている場合、エクスチェンジを開始した SAP サーバーエンドポイントは失敗を認識し、送信している SAP システムがエラーについて通知されます。その後、リクエストを再度処理するために同じ TID で別のトランザクションリクエストを送信することで、SAP システムを送信すると応答できます。

269.8. RFC の XML シリアライゼーション

概要

SAP リクエストおよび応答オブジェクトは XML シリアライゼーション形式をサポートします。

XML 名前空間

リポジトリーの各 RFC は、要求および応答オブジェクトのシリアライズ形式を構成する要素の特定の XML ネームスペースを定義します。この名前空間 URL の形式は以下のとおりです。

```
http://sap.fusesource.org/rfc/<Repository Name>/<RFC Name>
```

RFC 名前空間 URL には、一般的な http://sap.fusesource.org/rfc プレフィックスがあり、その後に RFC のメタデータが定義されているリポジトリーの名前が続きます。URL の最後のコンポーネントは RFC 自体の名前です。

リクエストおよび応答 XML ドキュメント

SAP リクエストオブジェクトは、Request という名前のドキュメントのルート要素で XML ドキュメントにシリアライズされ、リクエストの RFC の namespace によってスコープされます。

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Request
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:Request>
```

SAP 応答オブジェクトは、response という名前のドキュメントのルート要素で XML ドキュメントにシリアライズされ、応答の RFC の namespace によってスコープされます。

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Response
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:Response>
```

構造フィールド

パラメーターリストまたはネストされた構造の構造フィールドは、要素としてシリアライズされます。シリアライズされた構造の要素名は、エンクロージングパラメーターリスト、構造、またはテーブル行エントリー内の構造のフィールド名に対応します。

```
<BOOK_FLIGHT:FLTINFO
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:FLTINFO>
```

RFC namespace の structure 要素のタイプ名は、以下の例のように構造を定義するレコードメタデータデータオブジェクトの名前に対応することに注意してください。

```
<xs:schema
    targetNamespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
    ...
    <xs:complexType name="FLTINFO_STRUCTURE”>
    ...
    </xs:complexType>
    ...
</xs:schema>
```

この区別は、「例 3: SAP からのリクエストの処理」 にあるように、構造をマーシャリングおよびアンマーシャリングする JAXB Bean を指定する場合に重要です。

テーブルフィールド

パラメーターリストまたはネストされた構造の表フィールドは、要素としてシリアライズされます。シリアライズされた構造の要素名は、エンクロージングパラメーターリスト、構造、またはテーブル行エントリー内のテーブルのフィールド名に対応します。table 要素には、テーブルの行エントリーのシリアライズされた値を保持する一連の行要素が含まれます。

```
<BOOK_FLIGHT:CONNINFO
     xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
    <row ... > ... </row>
    ...
    <row ... > ... </row>
</BOOK_FLIGHT:CONNINFO>
```

RFC namespace のテーブル要素のタイプ名は、_TABLE が付けられたテーブルの行構造を定義するレコードメタデータオブジェクトの名前に対応することに注意してください。RFC 名のテーブル行要素のタイプ名は、以下の例のように、テーブルの行構造を定義するレコードメタデータオブジェクトの名前に対応します。

```
<xs:schema
     targetNamespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
    ...
    <xs:complextType name="CONNECTION_INFO_STRUCTURE_TABLE”>
        <xs:sequence>
            <xs:element
                name="row”
                minOccures="0”
                maxOccurs="unbounded”
                type="CONNECTION_INFO_STRUCTURE”/>
            ...
            <xs:sequence>
        </xs:sequence>
    </xs:complexType>

    <xs:complextType name="CONNECTION_INFO_STRUCTURE”>
            ...
```

```
    </xs:complexType>
    ...
</xs:schema>
```

この区別は、「例 3: SAP からのリクエストの処理」 にあるように、構造をマーシャリングおよびアンマーシャリングする JAXB Bean を指定する場合に重要です。

要素フィールド

パラメーターリストまたはネストされた構造の要素フィールドは、エンクロージングパラメーターリストまたは構造の要素の属性としてシリアライズされます。シリアライズされたフィールドの属性名は、以下の例のように、エンクロージングパラメーターリスト、構造、またはテーブル行エントリー内のフィールドのフィールド名に対応します。

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Request
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT"
    CUSTNAME="James Legrand"
   PASSFORM="Mr"
   PASSNAME="Travelin Joe"
   PASSBIRTH="1990-03-17T00:00:00.000-0500"
   FLIGHTDATE="2014-03-19T00:00:00.000-0400"
   TRAVELAGENCYNUMBER="00000110"
   DESTINATION_FROM="SFO"
   DESTINATION_TO="FRA"/>
```

日付と時刻の形式

日付および時間フィールドは、以下の形式で属性値にシリアライズされます。

```
yyyy-MM-dd'T'HH:mm:ss.SSSZ
```

日付フィールドは、年、月、日、タイムゾーンコンポーネントのみが設定された状態でシリアライズされます。

```
DEPDATE="2014-03-19T00:00:00.000-0400"
```

時間フィールドは、時間、分、秒、およびタイムゾーンコンポーネントのみで設定されたシリアライズされます。

```
DEPTIME="1970-01-01T16:00:00.000-0500"
```

269.9. IDOC の XML シリアライゼーション

概要

IDoc メッセージボディーは、組み込み型コンバーターを使用して XML 文字列フォーマットにシリアライズできます。

XML 名前空間

シリアライズされた IDoc はそれぞれ XML 名前空間に関連付けられ、以下の一般的な形式があります。

> http://sap.fusesource.org/idoc/**repositoryName/idocType/idocTypeExtension/systemRelease/applicationRelease**

repositoryName （リモート SAP メタデータリポジトリーの名前）と **idocType** （IDoc ドキュメントタイプ）の両方は必須ですが、**namespace** の他のコンポーネントは空白のままにすることができます。たとえば、以下のような XML 名前空間があるとします。

```
http://sap.fusesource.org/idoc/MY_REPO/FLCUSTOMER_CREATEFROMDATA01///
```

組み込み型コンバーター

Camel SAP コンポーネントには組み込みの型コンバーターがあり、Document オブジェクトまたは DocumentList オブジェクトを String 型との間で変換できます。

たとえば、Document オブジェクトを XML 文字列にシリアライズするには、以下の行を XML DSL のルートに追加できます。

```
<convertBodyTo type="java.lang.String"/>
```

この手法を使用して、シリアライズされた XML メッセージを Document オブジェクトに使用することもできます。たとえば、現在のメッセージボディーがシリアライズされた XML 文字列である場合、以下の行を XML DSL のルートに追加して Document オブジェクトに変換することができます。

```
<convertBodyTo type="org.fusesource.camel.component.sap.model.idoc.Document"/>
```

XML 形式の IDoc メッセージボディーの例

IDoc メッセージを String に変換すると、これは XML ドキュメントにシリアライズされます。ルート要素は idoc:Document （単一ドキュメントの場合）または idoc:DocumentList （ドキュメントのリスト）になります。例269.2「XML の IDoc メッセージボディー」 は、idoc:Document 要素にシリアライズされた単一の IDoc ドキュメントを示しています。

例269.2 XML の IDoc メッセージボディー

```
<?xml version="1.0" encoding="ASCII"?>
<idoc:Document
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:FLCUSTOMER_CREATEFROMDATA01---
="http://sap.fusesource.org/idoc/XXX/FLCUSTOMER_CREATEFROMDATA01///"
    xmlns:idoc="http://sap.fusesource.org/idoc"
    creationDate="2015-01-28T12:39:13.980-0500"
    creationTime="2015-01-28T12:39:13.980-0500"
    iDocType="FLCUSTOMER_CREATEFROMDATA01"
    iDocTypeExtension=""
    messageType="FLCUSTOMER_CREATEFROMDATA"
    recipientPartnerNumber="QUICKCLNT"
    recipientPartnerType="LS"
    senderPartnerNumber="QUICKSTART"
    senderPartnerType="LS">
  <rootSegment xsi:type="FLCUSTOMER_CREATEFROMDATA01---:ROOT" document="/">
    <segmentChildren parent="//@rootSegment">
      <E1SCU_CRE parent="//@rootSegment" document="/">
        <segmentChildren parent="//@rootSegment/@segmentChildren/@E1SCU_CRE.0">
          <E1BPSCUNEW parent="//@rootSegment/@segmentChildren/@E1SCU_CRE.0"
              document="/"
              CUSTNAME="Fred Flintstone" FORM="Mr."
              STREET="123 Rubble Lane"
              POSTCODE="01234"
              CITY="Bedrock"
              COUNTR="US"
              PHONE="800-555-1212"
              EMAIL="fred@bedrock.com"
              CUSTTYPE="P"
              DISCOUNT="005"
              LANGU="E"/>
        </segmentChildren>
      </E1SCU_CRE>
    </segmentChildren>
  </rootSegment>
</idoc:Document>
```

269.10. 例 1: SAP からのデータの読み取り

概要

この例は、SAP から FlightCustomer ビジネスオブジェクトデータを読み取るルートを示しています。ルートは、SAP 同期 RFC 宛先エンドポイントを使用して FlightCustomer BAPI メソッド

BAPI_FLCUST_GETLIST を呼び出してデータを取得します。

ルートの Java DSL

サンプルルートの Java DSL は以下のとおりです。

```
from("direct:getFlightCustomerInfo")
    .to("bean:createFlightCustomerGetListRequest")
    .to("sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST")
    .to("bean:returnFlightCustomerInfo");
```

ルートの XML DSL

同じルートの Spring DSL は以下のとおりです。

```
<route>
    <from uri="direct:getFlightCustomerInfo"/>
    <to uri="bean:createFlightCustomerGetListRequest"/>
    <to uri="sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST"/>
    <to uri="bean:returnFlightCustomerInfo"/>
</route>
```

createFlightCustomerGetListRequest bean

createFlightCustomerGetListRequest Bean は、後続の SAP エンドポイントの RFC 呼び出しで使用されるエクスチェンジメソッドに SAP 要求オブジェクトをビルドします。以下のコードスニペットは、要求オブジェクトをビルドする操作シーケンスを示しています。

```
public void create(Exchange exchange) throws Exception {

    // Get SAP Endpoint to be called from context.
    SapSynchronousRfcDestinationEndpoint endpoint =
        exchange.getContext().getEndpoint("sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST",
                                          SapSynchronousRfcDestinationEndpoint.class);

    // Retrieve bean from message containing Flight Customer name to
    // look up.
    BookFlightRequest bookFlightRequest =
        exchange.getIn().getBody(BookFlightRequest.class);

    // Create SAP Request object from target endpoint.
    Structure request = endpoint.getRequest();

    // Add Customer Name to request if set
    if (bookFlightRequest.getCustomerName() != null &&
        bookFlightRequest.getCustomerName().length() > 0) {
```

```
            request.put("CUSTOMER_NAME",
                    bookFlightRequest.getCustomerName());
        }
    } else {
        throw new Exception("No Customer Name");
    }

    // Put request object into body of exchange message.
    exchange.getIn().setBody(request);
}
```

returnFlightCustomerInfo bean

returnFlightCustomerInfo Bean は、以前の SAP エンドポイントから受信するエクスチェンジメソッドで SAP 応答オブジェクトからデータを抽出します。以下のコードスニペットは、応答オブジェクトからデータを抽出する操作シーケンスを示しています。

```
public void createFlightCustomerInfo(Exchange exchange) throws Exception {

    // Retrieve SAP response object from body of exchange message.
    Structure flightCustomerGetListResponse =
        exchange.getIn().getBody(Structure.class);

    if (flightCustomerGetListResponse == null) {
        throw new Exception("No Flight Customer Get List Response");
    }

    // Check BAPI return parameter for errors
    @SuppressWarnings("unchecked")
    Table<Structure> bapiReturn =
        flightCustomerGetListResponse.get("RETURN", Table.class);
    Structure bapiReturnEntry = bapiReturn.get(0);
    if (bapiReturnEntry.get("TYPE", String.class) != "S") {
        String message = bapiReturnEntry.get("MESSAGE", String.class);
        throw new Exception("BAPI call failed: " + message);
    }

    // Get customer list table from response object.
    @SuppressWarnings("unchecked")
    Table<? extends Structure> customerList =
        flightCustomerGetListResponse.get("CUSTOMER_LIST", Table.class);

    if (customerList == null || customerList.size() == 0) {
        throw new Exception("No Customer Info.");
    }

    // Get Flight Customer data from first row of table.
    Structure customer = customerList.get(0);

    // Create bean to hold Flight Customer data.
    FlightCustomerInfo flightCustomerInfo = new FlightCustomerInfo();

    // Get customer id from Flight Customer data and add to bean.
```

```
    String customerId = customer.get("CUSTOMERID", String.class);
    if (customerId != null) {
        flightCustomerInfo.setCustomerNumber(customerId);
    }

    ...

    // Put bean into body of exchange message.
    exchange.getIn().setHeader("flightCustomerInfo", flightCustomerInfo);

}
```

269.11. 例 2: SAP へのデータの作成

概要

この例は、SAP で FlightTrip ビジネスオブジェクトインスタンスを作成するルートを示しています。ルートは、宛先エンドポイントを使用してオブジェクトを作成する FlightTrip BAPI メソッド BAPI_FLTRIP_CREATE を呼び出します。

ルートの Java DSL

サンプルルートの Java DSL は以下のとおりです。

```
from("direct:createFlightTrip")
    .to("bean:createFlightTripRequest")
    .to("sap-srfc-destination:nplDest:BAPI_FLTRIP_CREATE?transacted=true")
    .to("bean:returnFlightTripResponse");
```

ルートの XML DSL

同じルートの Spring DSL は以下のとおりです。

```
<route>
    <from uri="direct:createFlightTrip"/>
    <to uri="bean:createFlightTripRequest"/>
    <to uri="sap-srfc-destination:nplDest:BAPI_FLTRIP_CREATE?transacted=true"/>
    <to uri="bean:returnFlightTripResponse"/>
</route>
```

トランザクションサポート

SAP エンドポイントの URL では、transacted オプションは true に設定されていることに注意してください。「トランザクションサポート」 で説明されているように、このオプションを有効にすると、エンドポイントは RFC 呼び出しを呼び出す前に SAP トランザクションセッションが開始されているこ

とを確認します。このエンドポイントの RFC は SAP で新規データを作成するため、このオプションは SAP でルートの変更を永続化する必要があります。

要求パラメーターの設定

createFlightTripRequest および returnFlightTripResponse Bean は、リクエストパラメーターを SAP リクエストに入力し、前の例と同じ操作シーケンスに従って SAP 応答からの応答パラメーターを抽出する役割を担います。

269.12. 例 3: SAP からのリクエストの処理

概要

この例は、ルートによって実装される、SAP から BOOK_FLIGHT RFC への要求を処理するルートを示しています。さらに、JAXB を使用して SAP リクエストオブジェクトと応答オブジェクトをカスタム Bean にアンマーシャリングし、マーシャリングすることで、コンポーネントの XML シリアライゼーションサポートを実証します。

このルートは、移動エージェント FlightCustomer の代わりに FlightTrip ビジネスオブジェクトを作成します。ルートは、SAP サーバーエンドポイントによって受信される SAP リクエストオブジェクトをカスタムの JAXB Bean にアンマーシャリングします。その後、このカスタム Bean はエクスチェンジで 3 つのサブルートにマルチキャストされます。これは、フライトトリップの作成に必要な移動エージェント、フライト接続、および乗客情報を収集します。最後のサブルートは、前の例で示すように、SAP でフライトトリップオブジェクトを作成します。最後のサブルートは、SAP 応答オブジェクトにマーシャリングされ、サーバーエンドポイントによって返されるカスタム JAXB Bean も作成および返します。

ルートの Java DSL

サンプルルートの Java DSL は以下のとおりです。

```
DataFormat jaxb = new JaxbDataFormat("org.fusesource.sap.example.jaxb");

from("sap-srfc-server:nplserver:BOOK_FLIGHT")
    .unmarshal(jaxb)
    .multicast()
    .to("direct:getFlightConnectionInfo",
        "direct:getFlightCustomerInfo",
        "direct:getPassengerInfo")
    .end()
    .to("direct:createFlightTrip")
    .marshal(jaxb);
```

ルートの XML DSL

*同じルートの **XML DSL** は以下のとおりです。*

```
<route>
    <from uri="sap-srfc-server:nplserver:BOOK_FLIGHT"/>
    <unmarshal>
        <jaxb contextPath="org.fusesource.sap.example.jaxb"/>
    </unmarshal>
    <multicast>
        <to uri="direct:getFlightConnectionInfo"/>
        <to uri="direct:getFlightCustomerInfo"/>
        <to uri="direct:getPassengerInfo"/>
    </multicast>
    <to uri="direct:createFlightTrip"/>
    <marshal>
        <jaxb contextPath="org.fusesource.sap.example.jaxb"/>
    </marshal>
</route>
```

BookFlightRequest bean

*以下のリストは、**SAP BOOK_FLIGHT** リクエストオブジェクトのシリアライズされた形式でアンマーシャリングする **JAXB Bean** を示しています。*

```
@XmlRootElement(name="Request",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class BookFlightRequest {

    @XmlAttribute(name="CUSTNAME")
    private String customerName;

    @XmlAttribute(name="FLIGHTDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date flightDate;

    @XmlAttribute(name="TRAVELAGENCYNUMBER")
    private String travelAgencyNumber;

    @XmlAttribute(name="DESTINATION_FROM")
    private String startAirportCode;

    @XmlAttribute(name="DESTINATION_TO")
    private String endAirportCode;

    @XmlAttribute(name="PASSFORM")
    private String passengerFormOfAddress;

    @XmlAttribute(name="PASSNAME")
    private String passengerName;
```

```
    @XmlAttribute(name="PASSBIRTH")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date passengerDateOfBirth;

    @XmlAttribute(name="CLASS")
    private String flightClass;

    ...
}
```

BookFlightResponse Bean

以下のリストは、SAP BOOK_FLIGHT 応答オブジェクトのシリアライズ形式にマーシャリングする JAXB Bean を示しています。

```
@XmlRootElement(name="Response",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class BookFlightResponse {

    @XmlAttribute(name="TRIPNUMBER")
    private String tripNumber;

    @XmlAttribute(name="TICKET_PRICE")
    private BigDecimal ticketPrice;

    @XmlAttribute(name="TICKET_TAX")
    private BigDecimal ticketTax;

    @XmlAttribute(name="CURRENCY")
    private String currency;

    @XmlAttribute(name="PASSFORM")
    private String passengerFormOfAddress;

    @XmlAttribute(name="PASSNAME")
    private String passengerName;

    @XmlAttribute(name="PASSBIRTH")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date passengerDateOfBirth;

    @XmlElement(name="FLTINFO")
    private FlightInfo flightInfo;

    @XmlElement(name="CONNINFO")
    private ConnectionInfoTable connectionInfo;

    ...
}
```

注記

応答オブジェクトの複雑なパラメーターフィールドは、レスポンスの子要素としてシリアライズされます。

FlightInfo bean

*以下のリストは、複雑な構造パラメーター **FLTINFO** のシリアライズ形式にマーシャリングする **JAXB Bean** を示しています。*

```
@XmlRootElement(name="FLTINFO",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class FlightInfo {

    @XmlAttribute(name="FLIGHTTIME")
    private String flightTime;

    @XmlAttribute(name="CITYFROM")
    private String cityFrom;

    @XmlAttribute(name="DEPDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date departureDate;

    @XmlAttribute(name="DEPTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date departureTime;

    @XmlAttribute(name="CITYTO")
    private String cityTo;

    @XmlAttribute(name="ARRDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date arrivalDate;

    @XmlAttribute(name="ARRTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date arrivalTime;

    ...
}
```

ConnectionInfoTable Bean

*以下のリストは、複雑なテーブルパラメーター **CONNINFO** のシリアライズ形式にマーシャリングする **JAXB Bean** を示しています。**JAXB Bean** のルート要素タイプの名前は、**_TABLE** が付けられた行構造型名に対応し、**Bean** には行要素のリストが含まれることに注意してください。*

```
@XmlRootElement(name="CONNINFO_TABLE",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class ConnectionInfoTable {

    @XmlElement(name="row")
    List<ConnectionInfo> rows;

    ...
}
```

ConnectionInfo bean

*以下のリストは、上記のテーブル行要素のシリアライズ形式にマーシャリングする **JAXB Bean** を示しています。*

```
@XmlRootElement(name="CONNINFO",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class ConnectionInfo {

    @XmlAttribute(name="CONNID")
    String connectionId;

    @XmlAttribute(name="AIRLINE")
    String airline;

    @XmlAttribute(name="PLANETYPE")
    String planeType;

    @XmlAttribute(name="CITYFROM")
    String cityFrom;

    @XmlAttribute(name="DEPDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date departureDate;

    @XmlAttribute(name="DEPTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date departureTime;

    @XmlAttribute(name="CITYTO")
    String cityTo;

    @XmlAttribute(name="ARRDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date arrivalDate;

    @XmlAttribute(name="ARRTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date arrivalTime;
```

```
    ...
}
```

第270章 SAP NETWEAVER COMPONENT

Camel バージョン 2.12 から利用可能

sap-netweaver は、HTTP トランスポートを使用して SAP NetWeaver ゲートウェイ と統合します。

この Camel コンポーネントはプロデューサーエンドポイントのみをサポートします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sap-netweaver</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

270.1. URI 形式

sap netweaver ゲートウェイコンポーネントの URI スキームは以下のようになります。

```
sap-netweaver:https://host:8080/path?username=foo&password=secret
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

270.2. 前提条件

このコンポーネントを利用するには、SAP NetWeaver システムにアカウントが必要です。SAP は、アカウントに必要な デモ用設定 を提供します。

このコンポーネントは、SAP NetWeaver にログインするために Basic 認証スキームを使用します。

270.3. SAPNETWEAVER オプション

SAP NetWeaver コンポーネントにはオプションがありません。

SAP NetWeaver エンドポイントは、URI 構文を使用して設定します。

```
sap-netweaver:url
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

270.3.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
url	SAP net-weaver ゲートウェイサーバーへの URL が **必要**。		文字列

270.3.2. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
flatternMap (producer)	JSON マップに単一のエントリーのみが含まれる場合、その単一のエントリー値をメッセージボディーとして格納してフラット化します。	true	boolean
JSON （プロデューサー）	JSON 形式のデータを返すかどうか。このオプションが false の場合、XML は Atom 形式で返されます。	true	boolean
jsonAsMap (producer)	JSON を String からメッセージのボディーの Map に変換するには、以下を行います。	true	boolean
password (producer)	アカウントに **必要な** パスワード。		文字列
username (producer)	アカウントに **必要な** ユーザー名		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

270.4. メッセージヘッダー

以下のヘッダーはプロデューサーで使用できます。

Name	タイプ	説明
CamelNetWeaverCommand	**文字列**	**必須**: MS ADO.Net Data Service 形式で実行するコマンド。

270.5. 例

この例では、SAP のフライトデモ例を使用して い ます。これは、インターネット上でオンラインで利用できます。

以下のルートで、以下の URL を使用して SAP NetWeaver デモサーバーを要求します。

```
https://sapes1.sapdevcenter.com/sap/opu/odata/IWBEP/RMTSAMPLEFLIGHT_2/
```

そして、以下のコマンドを実行します。

```
FlightCollection(AirLineID='AA',FlightConnectionID='0017',FlightDate=datetime'2012-08-29T00%3A00%3A00')
```

指定のフライトのフライト詳細を取得します。コマンド構文は MS ADO.Net Data Service 形式です。

以下の Camel ルートがあります。

```
from("direct:start")
    .setHeader(NetWeaverConstants.COMMAND, constant(command))
    .toF("sap-netweaver:%s?username=%s&password=%s", url, username, password)
    .to("log:response")
    .to("velocity:flight-info.vm")
```

url、ユーザー名、パスワード、およびコマンドは以下のように定義されます。

```
    private String username = "P1909969254";
```

```
    private String password = "TODO";
    private String url =
"https://sapes1.sapdevcenter.com/sap/opu/odata/IWBEP/RMTSAMPLEFLIGHT_2/";
    private String command =
"FlightCollection(AirLineID='AA',FlightConnectionID='0017',FlightDate=datetime'2012-08-
29T00%3A00%3A00')";
```

パスワードは無効です。最初に SAP でアカウントを作成し、デモを実行する必要があります。

velocity テンプレートは、基本的な HTML ページへの応答のフォーマットに使用されます。

```
<html>
  <body>
  Flight information:

  <p/>
  <br/>Airline ID: $body["AirLineID"]
  <br/>Aircraft Type: $body["AirCraftType"]
  <br/>Departure city: $body["FlightDetails"]["DepartureCity"]
  <br/>Departure airport: $body["FlightDetails"]["DepartureAirPort"]
  <br/>Destination city: $body["FlightDetails"]["DestinationCity"]
  <br/>Destination airport: $body["FlightDetails"]["DestinationAirPort"]

  </body>
</html>
```

アプリケーションを実行すると、サンプリングの出力が表示されます。

```
Flight information:
Airline ID: AA
Aircraft Type: 747-400
Departure city: new york
Departure airport: JFK
Destination city: SAN FRANCISCO
Destination airport: SFO
```

270.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- *エンドポイント*
- *はじめに*
- ***HTTP***

第271章 スケジューラーコンポーネント

Camel バージョン 2.15 から利用可能

scheduler: コンポーネントは、スケジューラーの実行時にメッセージエクスチェンジを生成するために使用されます。このコンポーネントは Timer コンポーネントと似ていますが、スケジューリングの観点でより多くの機能を提供します。また、このコンポーネントは JDK ScheduledExecutorService を使用します。タイマーは JDK Timer を使用します。

このエンドポイントからのみイベントを消費できます。

271.1. URI 形式

```
scheduler:name[?options]
```

name は、エンドポイント全体で作成され、共有されるスケジューラーの名前です。そのため、すべてのタイマーエンドポイントに同じ名前を使用する場合は、1 つのスケジューラースレッドプールとスレッドのみが使用されますが、スレッドプールを設定して、複数の同時スレッドを許可することができます。

URI にクエリーオプションを追加するには、?option=value&option=value&…

注記： 生成されたエクスチェンジの IN ボディーは null です。So exchange.getIn().getBody() returns null.

271.2. オプション

Scheduler コンポーネントは以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
concurrentTasks (scheduler)	スケジューリングスレッドプールによって使用されるスレッドの数。は、デフォルトで単一のスレッドを使用します。	1	int

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

スケジューラーエンドポイントは、URI 構文を使用して設定します。

```
scheduler:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

271.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	***必須*** のスケジューラーの名前		文字列

271.2.2. クエリーパラメーター（20 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。つまり、ポーリングによって情報の収集中にエラーが発生しました。たとえば、ファイルネットワークへのアクセスに失敗したため、Camel はファイルスキャンのためにアクセスできません。デフォルトの実装では、WARN レベルで原因となる例外がログに記録され、無視されます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
concurrentTasks (scheduler)	スケジューリングスレッドプールによって使用されるスレッドの数。は、デフォルトで単一のスレッドを使用します。	1	int

Name	説明	デフォルト	Type
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。デフォルト値は 500 です。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。デフォルト値は 1000 です。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。このオプションを使用すると、複数のコンシューマー間でスレッドプールを共有できます。		ScheduledExecutor Service
scheduler (scheduler)	カスタムの org.apache.camel.spi.ScheduledPollConsumerScheduler をポーリングコンシューマーの実行時に実行するスケジューラーとして使用することができます。デフォルトの実装では ScheduledExecutorService を使用し、CRON 式をサポートする Quartz2 および Spring ベースのものがあります。注記： カスタムスケジューラーを使用する場合は、initialDelay のオプション、useFixedDelay、timeUnit、および scheduledExecutorService が使用されていない可能性があります。quartz2 のテキストを使用して Quartz2 スケジューラーを参照し、Spring のテキストを使用して Spring ベースを使用し、myScheduler のテキストを使用してレジストリーの id でカスタムスケジューラーを参照します。例については、Quartz2 ページを参照してください。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ

Name	説明	デフォルト	Type
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

271.3. 詳細情報

このコンポーネントは、上記のオプションに関する詳細情報を見つけることができるスケジューラーポーリングコンシューマー です。また、「Polling Consumer」ページを参照してください。

271.4. エクスチェンジプロパティー

タイマーが実行されると、エクスチェンジに以下の情報がプロパティーとして追加されます。

Name	タイプ	説明
Exchange.TIMER_NAME	**文字列**	**name** オプションの値。
Exchange.TIMER_FIRED_TIME	**Date**	コンシューマーが実行される時間。

271.5. 例

60 秒ごとにイベントを生成するルートを設定するには、以下を実行します。

```
from("scheduler://foo?period=60s").to("bean:myBean?method=someMethodName");
```

上記のルートはイベントを生成し、JNDI や Spring などのレジストリーで myBean という Bean で

someMethodName メソッドを呼び出します。

Spring DSL のルートの場合：

```
<route>
  <from uri="scheduler://foo?period=60s"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

271.6. 完了後にスケジューラーがすぐにトリガーされるように強制する

以前のタスクが完了した直後にスケジューラーがトリガーされるようにするには、greedy=true オプションを設定します。ただし、スケジューラーは常に実行を継続します。したがって、注意してこれを使用してください。

271.7. スケジューラーを強制的にアイドル状態にする

*スケジューラーがトリガーし、処理する必要があるユースケースがあります。ただし、ポーリングするタスクがない「スケジューラー」が必要になる場合があり、スケジューラーは backoff オプションを使用してアイドルモードに変更することができます。そのためには、
キー Exchange.SCHEDULER_POLLED_MESSAGES を使用してエクスチェンジにプロパティーをブール値 false に設定する必要があります。これにより、コンシューマーがポーリングされたメッセージがないことを示します。*

コンシューマーは、コンシューマーがエクスチェンジの処理を完了するたびに、デフォルトでスケジューラーにポーリングされた 1 メッセージを返します。

271.8. 関連項目

- *Timer*
- *Quartz*

第272章 SCHEMATRON コンポーネント

Camel バージョン 2.15 から利用可能

Schematron は、XML インスタンスのドキュメントを検証する XML ベースの言語です。これは、XML ドキュメントのデータに関するアサーションの作成に使用され、運用およびビジネスルールを表現するために使用されます。Schematron は ISO 標準です。schematron コンポーネントは、ISO schematron の主要な 実装 を使用します。これは XSLT ベースの実装です。schematron ルールは 4 つの XSLT パイプライン を介して実行されます。これは、XML ドキュメントに対してアサーションを実行するベースとして使用される最終的な XSLT を生成します。コンポーネントは、スキーマルールがエンドポイントの先頭（一度だけ）に読み込まれる方法で記述されます。これは、ルールを表す Java テンプレートオブジェクトをインスタンス化するオーバーヘッドを最小限に抑えることです。

272.1. URI 形式

```
schematron://path?[options]
```

272.2. URI オプション

Schematron コンポーネントにはオプションがありません。

Schematron エンドポイントは URI 構文を使用して設定されます。

```
schematron:path
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

272.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
path	schematron ルールファイルへのパスが **必要です**。ファイルシステムのクラスパスまたは場所のいずれかになります。		文字列

272.2.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
強制終了 （プロデューサー）	ルートを中止し、スキーマ検証例外をスローするフラグ。	false	boolean
ルール （プロデューサー）	パスから読み込む代わりに、指定の schematron ルールを使用します。		テンプレート
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
uriResolver (advanced)	schematron の解決に使用する URIResolver をルールファイルに追加します。		URIResolver

272.3. HEADERS

Name	詳細	Type	In/Out
CamelSchematronValidationStatus	スキーマの検証ステータス：SUCCESS / FAILED	文字列	IN
CamelSchematronValidationReport	XML 形式のスキーマレポートボディー。以下の例を参照してください。	文字列	IN

272.4. URI およびパス構文

以下の例は、Java DSL で schematron プロセッサーを呼び出す方法を示しています。schematron ルールファイルは、クラスパスから取得されます。

```
from("direct:start").to("schematron://sch/schematron.sch").to("mock:result")
```

以下の例は、XML DSL で schematron プロセッサーを呼び出す方法を示しています。スキーマルールファイルは、ファイルシステムから取得されます。

```
<route>
  <from uri="direct:start" />
  <to uri="schematron:///usr/local/sch/schematron.sch" />
  <log message="Schematron validation status: ${in.header.CamelSchematronValidationStatus}" />
  <choice>
    <when>
      <simple>${in.header.CamelSchematronValidationStatus} == 'SUCCESS'</simple>
      <to uri="mock:success" />
    </when>
    <otherwise>
      <log message="Failed schematron validation" />
      <setBody>
        <header>CamelSchematronValidationReport</header>
      </setBody>
      <to uri="mock:failure" />
    </otherwise>
  </choice>
</route>
```

ヒント

schematron ルールを保存する場所Schematron ルールはビジネス要件で変更する可能性があります。そのため、これらのルールをファイルシステム内のどこかに保存することが推奨されます。schematron コンポーネントエンドポイントが起動すると、ルールは XSLT に Java テンプレートオブジェクトとしてコンパイルされます。これは、Java テンプレートオブジェクトのインスタンス化のオーバーヘッドを最小化するために 1 回のみ行われます。これは、大量のルールセットで高価な操作となる可能性があり、プロセスが XSLT 変換 の 4 つのパイプラインを経由すると仮定します。したがって、更新時にファイルシステムにルールを保存する場合は、ルートまたはコンポーネントを再起動するだけで済みます。ただし、これらのルールをクラスパスに保存することの影響はありませんが、変更を取得するためにコンポーネントをビルドし、デプロイする必要があります。

272.5. SCHEMATRON ルールおよびレポートサンプル

schematron ルールの例を以下に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
  <title>Check Sections 12/07</title>
  <pattern id="section-check">
```

```
    <rule context="section">
      <assert test="title">This section has no title</assert>
      <assert test="para">This section has no paragraphs</assert>
    </rule>
  </pattern>
</schema>
```

schematron レポートの例を以下に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<svrl:schematron-output xmlns:svrl="http://purl.oclc.org/dsdl/svrl"
 xmlns:iso="http://purl.oclc.org/dsdl/schematron"
 xmlns:saxon="http://saxon.sf.net/"
 xmlns:schold="http://www.ascc.net/xml/schematron"
 xmlns:xhtml="http://www.w3.org/1999/xhtml"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" schemaVersion="" title="">

  <svrl:active-pattern document="" />
  <svrl:fired-rule context="chapter" />
  <svrl:failed-assert test="title" location="/doc[1]/chapter[1]">
    <svrl:text>A chapter should have a title</svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="chapter" />
  <svrl:failed-assert test="title" location="/doc[1]/chapter[2]">
    <svrl:text>A chapter should have a title</svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="chapter" />
</svrl:schematron-output>
```

ヒント

*便利なリンクとリソース * Mulleberry テクノロジー によるスキーマ処理の概要Schematron で開始するための PDF の優れたドキュメントです。* Schematron の公式サイト他のリソースへのリンクが含まれます。*

第273章 SCP コンポーネント

Camel バージョン 2.10 で利用可能

camel-jsch コンポーネントは、Jsch プロジェクトの Client API を使用して SCP プロトコル をサポートします。jsch は、sftp: プロトコルの FTP コンポーネントによってすでに camel で使用されています。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jsch</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

273.1. URI 形式

```
scp://host[:port]/destination[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&...

ファイル名は、URI の <path> の部分またはメッセージの「CamelFileName」ヘッダーとして指定できます（コードで使用されている場合はExchange.FILE_NAME ）。

273.2. オプション

SCP コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
verboseLogging (producer)	JSCH は追加設定なしで詳細なロギングです。そのため、デフォルトでロギングを DEBUG ロギングまでオフにします。ただし、このオプションを true に設定すると、再度詳細なロギングが有効になります。	false	boolean

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SCP エンドポイントは、URI 構文を使用して設定します。

```
scp:host:port/directoryName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

273.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
host	FTP サーバーに **必要な** ホスト名		文字列
port	FTP サーバーのポート		int
directoryName	起動ディレクトリー		文字列

273.2.2. クエリーパラメーター（20 パラメーター）：

Name	説明	デフォルト	Type
切断 （共通）	使用後にリモートの FTP サーバーから適切な接続を解除するかどうか。切断すると、現在の FTP サーバーへの接続が切断されるだけです。停止するコンシューマーがある場合は、代わりにコンシューマー/ルートを停止する必要があります。	false	boolean
chmod （プロデューサー）	保存したファイルで chmod を設定できます。例：chmod=664	664	文字列

Name	説明	デフォルト	Type
fileName (producer)	File Language などの式を使用して、ファイル名を動的に設定します。コンシューマーの場合は、ファイル名フィルターとして使用されます。プロデューサーの場合、書き込みするファイル名を評価するために使用されます。式が設定されている場合は、CamelFileName ヘッダーよりも優先されます。（注記： ヘッダー自体は式にすることもできます。式オプションは String タイプと Expression 型の両方をサポートします。式が String タイプである場合、これは常に File 言語を使用して評価されます。式が Expression タイプである場合、指定された式タイプが使用されます。たとえば、OGNL 式を使用できます。コンシューマーの場合は、これを使用してファイル名を絞り込むことができます。そのため、インスタンスは File Language 構文 mydata-$date:yyyyMMdd.txt を使用して現在のファイルを使用できます。プロデューサーは、既存の CamelFileName ヘッダーよりも優先される CamelOverruleFileName ヘッダーをサポートします。CamelOverruleFileName は 1 度だけ使用されるヘッダーで、CamelFileName を一時的に保存し、後で復元する必要があるため、より簡単に使用できます。		文字列
flatten (producer)	フラット化は、ファイル名パスをフラット化して先頭のパスを削除するために使用されるので、ファイル名だけになります。これにより、サブディレクトリーに再帰的に消費できますが、たとえば別のディレクトリーにファイルを書き込むと、それらのファイルは単一のディレクトリーに書き込まれます。これをプロデューサーで true に設定すると、CamelFileName ヘッダーのファイル名が任意の先頭パスに対して削除されます。	false	boolean
strictHostKeyChecking (producer)	厳密なホストキーチェックを使用するかどうかを設定します。使用できる値は no、yes です。	いいえ	文字列
allowNullBody (producer)	ファイルの書き込み中に null ボディーを許可するかどうかを指定するのに使用します。true に設定すると、空のファイルが作成され、false に設定され、null ボディーを file コンポーネントに送信しようとすると、'Cannot write null body to file.' の GenericFileWriteException がスローされます。fileExist オプションを 'Override' に設定すると、ファイルが切り捨てられます。追加する場合は、ファイルは変更されません。	false	boolean

Name	説明	デフォルト	Type
disconnectOnBatchComplete (producer)	Batch アップロードの完了後にリモート FTP サーバーから右に切断するかどうか。disconnectOnBatchComplete は、現在の FTP サーバーの接続のみを切断します。	false	boolean
connectTimeout (advanced)	FTPClient と JSCH の両方によって接続が確立されるまでの接続タイムアウトを設定します。	10000	int
soTimeout (advanced)	FTPClient でのみ使用されるようにタイムアウトを設定します。	300000	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
タイムアウト （詳細）	FTPClient のみで使用された応答を待機するデータタイムアウトを設定します。	30000	int
knownHostsFile (security)	known_hosts ファイルを設定し、jsch エンドポイントがホストキーの検証を実行できるようにします。classpath: をプレフィックスとして指定すると、ファイルシステムの代わりにクラスパスからファイルを読み込むことができます。		文字列
パスワード （セキュリティー）	ログインに使用するパスワード		文字列
preferredAuthentications (security)	希望順に使用する認証のコンマ区切りリストを設定します。可能な認証方法は JCraft JSCH で定義されます。例には、gssapi-with-mic,publickey,keyboard-interactive,password などがあります。指定されていない場合は、JSCH やシステムのデフォルトが使用されます。		文字列
privateKeyBytes (security)	エンドポイントが秘密鍵の検証を実行できるように、秘密鍵バイトを設定します。これは、privateKeyFile が設定されていない場合にのみ使用してください。そうでないと、ファイルは優先されます。		byte[]
privateKeyFile (security)	エンドポイントが秘密鍵の検証を実行できるように、秘密鍵ファイルを設定します。classpath: をプレフィックスとして指定すると、ファイルシステムの代わりにクラスパスからファイルを読み込むことができます。		文字列

Name	説明	デフォルト	Type
privateKeyFilePassphrase (security)	エンドポイントが秘密鍵の検証を実行できるように、秘密鍵ファイルのパスフレーズを設定します。		文字列
ユーザー名 （セキュリティー）	ログインに使用するユーザー名		文字列
useUserKnownHostsFile (security)	knownHostFile が明示的に設定されていない場合は、System.getProperty(user.home)/.ssh/known_hosts のホストファイルを使用します。	true	boolean
暗号 化（セキュリティー）	優先順に使用する暗号のコンマ区切りリストを設定します。可能な暗号名は JCraft JSCH で定義されます。例として、aes128-ctr,aes128-cbc,3des-ctr,3des-cbc,blowfish-cbc,aes192-cbc,aes256-cbc,aes256-cbc などがあります。指定されていない場合は、JSCH のデフォルト一覧が使用されます。		文字列

273.3. 制限

現在、camel-jsch は プロデューサー のみをサポートします（つまり、別のホストにファイルをコピーします）。

273.4. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第274章 CAMEL SCR（非推奨）

Camel 2.15 から利用可能

SCR はサービスコンポーネントランタイムを表し、OSGi Declarative Services 仕様の実装です。SCR により、プレーンな古い Java オブジェクトが、ボイラープレートコードなしで OSGi サービスを公開し、使用できるようになります。

OSGi フレームワークは、通常 org.apache.felix:maven-scr-plugin などのプラグインによって Java アノテーションから生成される SCR 記述子ファイルをバンドルで確認します。

Camel を SCR バンドルで実行することは、Spring DM と Blueprint ベースのソリューション向けの便利な選択肢で、ユーザーと OSGi フレームワーク間のコード行は大幅に少なくなります。SCR を使用すると、バンドルは完全に Java グローバルに残ることができます。XML やプロパティーファイルの編集は必要ありません。これにより、すべてを完全に制御でき、選択した IDE はプロジェクトで何が起こっているのかを正確に認識することを意味します。

274.1. CAMEL SCR のサポート

camel-scr バンドルは Apache Camel バージョン before 2.15.0 に含まれていませんが、アーティファクト自体は 2.12.0 以降の Camel バージョンで使用できます。

org.apache.camel/camel-scr バンドルは、Camel コンテキストと、ユニットテストで SCR プロパティーを使用するためにヘルパークラス ScrHelper を管理するベースクラス AbstractCamelRunner を提供します。Apache Karaf の camel-scr 機能は、Camel を SCR バンドルで実行するために必要なすべての機能とバンドルを定義します。

AbstractCamelRunner クラスは CamelContext のライフサイクルを Service Component のライフサイクルに結び付け、Camel の PropertiesComponent を利用して設定を処理します。サービスコンポーネントを java クラスから外すには、AbstractCamelRunner から拡張し、クラスレベルで以下の org.apache.felix.scr.annotations を追加するだけです。

必要なアノテーションの追加

```
@Component
@References({
    @Reference(name = "camelComponent",referenceInterface = ComponentResolver.class,
        cardinality = ReferenceCardinality.MANDATORY_MULTIPLE, policy =
ReferencePolicy.DYNAMIC,
```

```
    policyOption = ReferencePolicyOption.GREEDY, bind = "gotCamelComponent", unbind =
"lostCamelComponent")
})
```

次に、実行する Camel ルートを返す getRouteBuilders（） メソッドを実装します。

Implement getRouteBuilders()

```
@Override
protected List<RoutesBuilder> getRouteBuilders() {
    List<RoutesBuilder> routesBuilders = new ArrayList<>();
    routesBuilders.add(new YourRouteBuilderHere(registry));
    routesBuilders.add(new AnotherRouteBuilderHere(registry));
    return routesBuilders;
}
```

最後に、以下の設定を提供します。

アノテーションのデフォルト設定

```
@Properties({
  @Property(name = "camelContextId", value = "my-test"),
  @Property(name = "active", value = "true"),
  @Property(name = "...", value = "..."),
  ...
})
```

以上です。また、camel-archetype-scr を使用してプロジェクトを生成する場合は、これはすでに行います。

以下は、camel-archetype-scr によって生成された完全な Service Component クラスの例になります。

CamelScrExample.java

```
// This file was generated from org.apache.camel.archetypes/camel-archetype-scr/2.15-
SNAPSHOT
```

```
package example;

import java.util.ArrayList;
import java.util.List;

import org.apache.camel.scr.AbstractCamelRunner;
import example.internal.CamelScrExampleRoute;
import org.apache.camel.RoutesBuilder;
import org.apache.camel.spi.ComponentResolver;
import org.apache.felix.scr.annotations.*;

@Component(label = CamelScrExample.COMPONENT_LABEL, description =
CamelScrExample.COMPONENT_DESCRIPTION, immediate = true, metatype = true)
@Properties({
    @Property(name = "camelContextId", value = "camel-scr-example"),
    @Property(name = "camelRouteId", value = "foo/timer-log"),
    @Property(name = "active", value = "true"),
    @Property(name = "from", value = "timer:foo?period=5000"),
    @Property(name = "to", value = "log:foo?showHeaders=true"),
    @Property(name = "messageOk", value = "Success: {{from}} -> {{to}}"),
    @Property(name = "messageError", value = "Failure: {{from}} -> {{to}}"),
    @Property(name = "maximumRedeliveries", value = "0"),
    @Property(name = "redeliveryDelay", value = "5000"),
    @Property(name = "backOffMultiplier", value = "2"),
    @Property(name = "maximumRedeliveryDelay", value = "60000")
})
@References({
    @Reference(name = "camelComponent",referenceInterface = ComponentResolver.class,
        cardinality = ReferenceCardinality.MANDATORY_MULTIPLE, policy =
ReferencePolicy.DYNAMIC,
        policyOption = ReferencePolicyOption.GREEDY, bind = "gotCamelComponent", unbind =
"lostCamelComponent")
})
public class CamelScrExample extends AbstractCamelRunner {

    public static final String COMPONENT_LABEL = "example.CamelScrExample";
    public static final String COMPONENT_DESCRIPTION = "This is the description for camel-
scr-example.";

    @Override
    protected List<RoutesBuilder> getRouteBuilders() {
        List<RoutesBuilder> routesBuilders = new ArrayList<>();
        routesBuilders.add(new CamelScrExampleRoute(registry));
        return routesBuilders;
    }
}
```

CamelContextId と アクティブな プロパティーは、CamelContext の名前（デフォルトは「camel-runner-default」）を制御し、起動すべきかどうか（デフォルトは「false」）を制御します。これらの

他に、任意の数のプロパティーを追加して使用できます。Camel の PropertiesComponent は、再帰的なプロパティーと、問題なくフォールバックを接頭辞として処理します。

AbstractCamelRunner は、Camel の PropertiesComponent を利用してこれらのプロパティーを RouteBuilder で利用できるようにし、名前が一致するとそれらの値を Service Component のフィールドおよび RouteBuilder のフィールドに注入します。フィールドは可視性レベルで宣言でき、多くのタイプはサポートされます（String、int、boolean、URL、...）。

以下は、camel-archetype-scr によって生成された RouteBuilder クラスの例になります。

CamelScrExampleRoute.java

```
// This file was generated from org.apache.camel.archetypes/camel-archetype-scr/2.15-SNAPSHOT
package example.internal;

import org.apache.camel.LoggingLevel;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.impl.SimpleRegistry;
import org.apache.commons.lang.Validate;

public class CamelScrExampleRoute extends RouteBuilder {

    SimpleRegistry registry;

    // Configured fields
    private String camelRouteId;
    private Integer maximumRedeliveries;
    private Long redeliveryDelay;
    private Double backOffMultiplier;
    private Long maximumRedeliveryDelay;

    public CamelScrExampleRoute(final SimpleRegistry registry) {
        this.registry = registry;
    }

    @Override
    public void configure() throws Exception {
        checkProperties();

        // Add a bean to Camel context registry
        registry.put("test", "bean");

        errorHandler(defaultErrorHandler()
            .retryAttemptedLogLevel(LoggingLevel.WARN)
```

```
            .maximumRedeliveries(maximumRedeliveries)
            .redeliveryDelay(redeliveryDelay)
            .backOffMultiplier(backOffMultiplier)
            .maximumRedeliveryDelay(maximumRedeliveryDelay));

        from("{{from}}")
            .startupOrder(2)
            .routeId(camelRouteId)
            .onCompletion()
                .to("direct:processCompletion")
            .end()
            .removeHeaders("CamelHttp*")
            .to("{{to}}");


        from("direct:processCompletion")
            .startupOrder(1)
            .routeId(camelRouteId + ".completion")
            .choice()
                .when(simple("${exception} == null"))
                    .log("{{messageOk}}")
                .otherwise()
                    .log(LoggingLevel.ERROR, "{{messageError}}")
            .end();
        }
    }

    public void checkProperties() {
        Validate.notNull(camelRouteId, "camelRouteId property is not set");
        Validate.notNull(maximumRedeliveries, "maximumRedeliveries property is not set");
        Validate.notNull(redeliveryDelay, "redeliveryDelay property is not set");
        Validate.notNull(backOffMultiplier, "backOffMultiplier property is not set");
        Validate.notNull(maximumRedeliveryDelay, "maximumRedeliveryDelay property is not
set");
    }
}
```

詳細は、CamelScrExampleRoute をご覧ください。

```
    // Configured fields
    private String camelRouteId;
    private Integer maximumRedeliveries;
    private Long redeliveryDelay;
    private Double backOffMultiplier;
    private Long maximumRedeliveryDelay;
```

これらのフィールドの値は、名前に一致することでプロパティーの値で設定されます。

```
        // Add a bean to Camel context registry
        registry.put("test", "bean");
```

ルート用に一部の Bean を CamelContext のレジストリーに追加する必要がある場合は、以下のように Bean を追加できます。

```
    public void checkProperties() {
        Validate.notNull(camelRouteId, "camelRouteId property is not set");
        Validate.notNull(maximumRedeliveries, "maximumRedeliveries property is not set");
        Validate.notNull(redeliveryDelay, "redeliveryDelay property is not set");
        Validate.notNull(backOffMultiplier, "backOffMultiplier property is not set");
        Validate.notNull(maximumRedeliveryDelay, "maximumRedeliveryDelay property is not
 set");
    }
```

必要なパラメーターが設定され、それらに意味のある値があることを確認すると、ルートの起動を許可することが推奨されます。

```
        from("{{from}}")
            .startupOrder(2)
            .routeId(camelRouteId)
            .onCompletion()
                .to("direct:processCompletion")
            .end()
            .removeHeaders("CamelHttp*")
            .to("{{to}}");


        from("direct:processCompletion")
            .startupOrder(1)
            .routeId(camelRouteId + ".completion")
            .choice()
                .when(simple("${exception} == null"))
                    .log("{{messageOk}}")
```

```
            .otherwise()
                .log(LoggingLevel.ERROR, "{{messageError}}")
        .end();
```

ルート内のほぼすべてがプロパティーで設定されていることに注意してください。これにより、RouteBuilder がテンプレートになります。SCR を使用すると、代替設定を指定するだけで、ルートのインスタンスをさらに作成することができます。本セクションの詳細は、「Camel SCR bundle をテンプレートとして使用する」を 参照してください。

274.2. SCR での ABSTRACTCAMELRUNNER のライフサイクル

1. コンポーネントの設定ポリシーと必須参照が SCR 呼び出し active ( )が満たされる場合。これは、activate（）→ prepare（）→ createCamelContext（）→ setupPropertiesComponent（） → configure（） → setup CamelContext（）の呼び出しチェーンを使用して CamelContext を作成 し、 設定 します。最後に、コンテキストは、AbstractCamelRunner.START_DELAY と runWithDelay（）で定義された遅延後に開始するようスケジュールされます。

2. Camel コンポーネント（ComponentResolver サービス）が OSGi に登録されると、SCR 呼び出し gotCamelComponent'（）' が表示され、CamelContext は同じ AbstractCamelRunner.START_DELAY によってさらに起動します。これにより、CamelContext はすべての Camel コンポーネントが読み込まれるまで待機するか、またはそれらのコンポーネント間で十分なギャップが発生するまで待機します。Camel コンポーネントをさらに追加するたびに、同じロジックが failed-to-start CamelContext に再度試行するように指示します。

3. Camel コンポーネントの登録が解除された場合、SCR 呼び出しは 失われた CamelComponent'（）' を呼び出します。この呼び出しは何もしません。

4. 呼び出しの アクティブ化（）を行う要件（） が失われた場合、SCR は非アクティブな状態 ( )を呼び出し ます。これにより、CamelContext がシャットダウンされます。

（OSGi 以外の）ユニットテストでは、より粒度の細かい制御を行うために、activate （）→ deactivate （）の代わりに prepare （）→ run （）→ stop （）を使用する必要があります。また、これにより、テストでの SCR 固有の操作を回避できます。

274.3. CAMEL-ARCHETYPE-SCR の使用

Camel SCR バンドルプロジェクトを作成するための最も簡単な方法として、camel-archetype-scr および Maven を使用できます。

以下の手順によりプロジェクトを生成できます。

プロジェクトの生成

```
$ mvn archetype:generate -Dfilter=org.apache.camel.archetypes:camel-archetype-scr

Choose archetype:
1: local -> org.apache.camel.archetypes:camel-archetype-scr (Creates a new Camel SCR
bundle project for Karaf)
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): : 1
Define value for property 'groupId': : example
[INFO] Using property: groupId = example
Define value for property 'artifactId': : camel-scr-example
Define value for property 'version': 1.0-SNAPSHOT: :
Define value for property 'package': example: :
[INFO] Using property: archetypeArtifactId = camel-archetype-scr
[INFO] Using property: archetypeGroupId = org.apache.camel.archetypes
[INFO] Using property: archetypeVersion = 2.15-SNAPSHOT
Define value for property 'className': : CamelScrExample
Confirm properties configuration:
groupId: example
artifactId: camel-scr-example
version: 1.0-SNAPSHOT
package: example
archetypeArtifactId: camel-archetype-scr
archetypeGroupId: org.apache.camel.archetypes
archetypeVersion: 2.15-SNAPSHOT
className: CamelScrExample
Y: :
```

完了しました！

次を実行します。

```
mvn install
```

また、バンドルをデプロイする準備が整います。

274.4. CAMEL ルートのテスト

サービスコンポーネントは POJO で、OSGi 以外のユニットテストに対する特別な要件はありません。しかし、Camel SCR 固有の手法や、テストを簡単にする手法がいくつかあります。

以下は、camel-archetype-scr によって生成されたユニットテストの例です。

```
// This file was generated from org.apache.camel.archetypes/camel-archetype-scr/2.15-
SNAPSHOT
package example;

import java.util.List;

import org.apache.camel.scr.internal.ScrHelper;
import org.apache.camel.builder.AdviceWithRouteBuilder;
import org.apache.camel.component.mock.MockComponent;
import org.apache.camel.component.mock.MockEndpoint;
import org.apache.camel.model.ModelCamelContext;
import org.apache.camel.model.RouteDefinition;
import org.junit.After;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.TestName;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.junit.runner.RunWith;
import org.junit.runners.JUnit4;

@RunWith(JUnit4.class)
public class CamelScrExampleTest {

    Logger log = LoggerFactory.getLogger(getClass());

    @Rule
    public TestName testName = new TestName();

    CamelScrExample integration;
    ModelCamelContext context;

    @Before
    public void setUp() throws Exception {
        log.info("************************************************************");
        log.info("Test: " + testName.getMethodName());
        log.info("************************************************************");

        // Set property prefix for unit testing
        System.setProperty(CamelScrExample.PROPERTY_PREFIX, "unit");

        // Prepare the integration
        integration = new CamelScrExample();
        integration.prepare(null, ScrHelper.getScrProperties(integration.getClass().getName()));
        context = integration.getContext();

        // Disable JMX for test
        context.disableJMX();

        // Fake a component for test
        context.addComponent("amq", new MockComponent());
```

```
    }

    @After
    public void tearDown() throws Exception {
        integration.stop();
    }

    @Test
    public void testRoutes() throws Exception {
        // Adjust routes
        List<RouteDefinition> routes = context.getRouteDefinitions();

        routes.get(0).adviceWith(context, new AdviceWithRouteBuilder() {
            @Override
            public void configure() throws Exception {
                // Replace "from" endpoint with direct:start
                replaceFromWith("direct:start");
                // Mock and skip result endpoint
                mockEndpoints("log:*");
            }
        });

        MockEndpoint resultEndpoint = context.getEndpoint("mock:log:foo",
MockEndpoint.class);
        // resultEndpoint.expectedMessageCount(1); // If you want to just check the number of
messages
        resultEndpoint.expectedBodiesReceived("hello"); // If you want to check the contents

        // Start the integration
        integration.run();

        // Send the test message
        context.createProducerTemplate().sendBody("direct:start", "hello");

        resultEndpoint.assertIsSatisfied();
    }
}
```

それでは、1つずつ関心のあるビットを見てみましょう。

プロパティー接頭辞の使用

```
        // Set property prefix for unit testing
        System.setProperty(CamelScrExample.PROPERTY_PREFIX, "unit");
```

これにより、プロパティーの前に「unit」をプレフィックスして、設定の一部を上書きできます。たとえば、ユニットテストから unit. from を上書きします。

接頭辞を使用すると、ルートが実行されるランタイム環境の相違点を処理できます。開発、テスト、および実稼働環境を介して変更されていないバンドルを移動することは、典型的なユースケースです。

アノテーションからのテスト設定の取得

```
integration.prepare(null, ScrHelper.getScrProperties(integration.getClass().getName()));
```

ここでは、OSGi 環境で使用されるプロパティーと同じプロパティーでテストの Service コンポーネントを設定します。

テスト用のコンポーネントのモック化

```
// Fake a component for test
context.addComponent("amq", new MockComponent());
```

テストで利用できないコンポーネントは、ルートの起動を許可するためにモック化できます。

テスト用のルートの調整

```
// Adjust routes
List<RouteDefinition> routes = context.getRouteDefinitions();

routes.get(0).adviceWith(context, new AdviceWithRouteBuilder() {
    @Override
    public void configure() throws Exception {
        // Replace "from" endpoint with direct:start
        replaceFromWith("direct:start");
        // Mock and skip result endpoint
```

```
            mockEndpoints("log:*");
        }
    });
```

Camel の AdviceWith 機能を使用すると、ルートをテスト用に変更できます。

ルートの起動

```
    // Start the integration
    integration.run();
```

ここでは、サービスコンポーネントとルートを起動します。

テストメッセージの送信

```
    // Send the test message
    context.createProducerTemplate().sendBody("direct:start", "hello");
```

ここでは、テスト中のルートにメッセージを送信します。

274.5. APACHE KARAF でのバンドルの実行

バンドルが mvn install でビルドされたら、デプロイする準備が整います。 Apache Karaf にバンドルをデプロイするには、Karaf コマンドラインで以下の手順を実行します。

Apache Karaf へのバンドルのデプロイ

```
# Add Camel feature repository
karaf@root> features:chooseurl camel 2.15-SNAPSHOT

# Install camel-scr feature
karaf@root> features:install camel-scr
```

```
# Install commons-lang, used in the example route to validate parameters
karaf@root> osgi:install mvn:commons-lang/commons-lang/2.6

# Install and start your bundle
karaf@root> osgi:install -s mvn:example/camel-scr-example/1.0-SNAPSHOT

# See how it's running
karaf@root> log:tail -n 10

Press ctrl-c to stop watching the log.
```

274.5.1. デフォルト設定の上書き

デフォルトでは、Service Component の設定 PID はクラスの完全修飾名になります。Karaf の config:* コマンドを使用して、バンドルのプロパティー例を変更することができます。

プロパティーの上書き

```
# Override 'messageOk' property
karaf@root> config:propset -p example.CamelScrExample messageOk "This is better
logging"
```

または、Karaf の etc フォルダーでプロパティーファイルを編集して設定を変更できます。

274.5.2. Camel SCR バンドルのテンプレートとしての使用

頻繁に使用するインテグレーションパターンを実装する Camel SCR バンドルがあるとします（例：→ から）、成功/失敗ロギングと再配信があり、これはルート例が実装するパターンでもあります。インスタンスごとに個別のバンドルを作成したくないでしょう。Worries なし、SCR の対象です。

サービスコンポーネントの設定 PID を作成しますが、ダッシュと SCR でテールを追加して、その設定を使用してコンポーネントの新規インスタンスを作成します。

新規サービスコンポーネントインスタンスの作成

```
# Create a PID with a tail
karaf@root> config:edit example.CamelScrExample-anotherone

# Override some properties
karaf@root> config:propset camelContextId my-other-context
karaf@root> config:propset to "file://removeme?fileName=removemetoo.txt"
```

```
# Save the PID
karaf@root> config:update
```

これにより、オーバーライドされたプロパティーで新しい CamelContext が起動します。便利な方法。

274.6. 備考

Service コンポーネントを設計してテンプレートとして設計する場合、通常はデフォルト設定で「調整された」設定を行わずに起動しないようにしてください。

サービスコンポーネントがデフォルト設定のデフォルトの設定で開始されないようにするには、ConfigurationPolicy.REQUIRE 'to the class level '@Component アノテーションに ' ConfigurationPolicy.REQUIRE' を追加します。

第275章 XML SECURITY DATAFORMAT

Camel バージョン 2.0 で利用可能

XMLSecurity Data Format を使用すると、Document、Element、および Element Content レベルでの XML ペイロードの暗号化と復号が容易になります（XPath を使用した同時マルチノード暗号化/復号化を含む）。XML 署名仕様を使用してメッセージに署名するには、Camel XML Security コンポーネントを参照してください。

暗号化機能は、Apache XML Security(Santuario)プロジェクトを使用してサポートされる形式に基づいています。対称暗号化/復号化は、現在 Triple-DES および AES（128、192、および 256）暗号化形式を使用してサポートされます。必要に応じて、追加形式を簡単に追加できます。この機能により、Camel ユーザーはルートでディスパッチまたは受信中にペイロードを暗号化/復号化できます。

Camel 2.9 で利用可能
は、XMLSecurity Data Format は非対称鍵の暗号化をサポートします。この暗号化モデルでは、対称鍵が生成され、XML コンテンツの暗号化または復号の実行に使用されます。この「コンテンツ暗号化キー」は、受信者の公開鍵を「鍵暗号化キー」として利用する非対称暗号化アルゴリズムを使用して暗号化されます。非対称鍵暗号化アルゴリズムを使用すると、受信者の秘密鍵のホルダーのみが生成された対称暗号化鍵にアクセスできるようになります。そのため、秘密鍵のホルダーのみがメッセージをデコードできます。XMLSecurity Data Format は、非対称鍵の暗号化を使用してメッセージコンテンツと暗号化キーを暗号化および復号化するのに必要なすべてのロジックを処理します。

XMLSecurity Data Format は、暗号化のコンテンツを選択する XPath クエリーを処理する際の名前空間のサポートも改善しました。namespace 定義マッピングは、データ形式設定の一部として含めることができます。これにより、XPath クエリーとターゲット xml ドキュメントの接頭辞の値が同等の文字列でなくても、true 名前空間の照合が可能になります。

275.1. XMLSECURITY オプション

XML セキュリティーデータ形式は、以下に示す 12 個のオプションをサポートします。

Name	デフォルト	Java タイプ	説明

Name	デフォルト	Java タイプ	説明
xmlCipherAlgorithm	**TIPLEDES**	**文字列**	XML メッセージコンテンツの暗号化/復号化に使用する暗号アルゴリズム。XMLCipher.TRIPLEDES XMLCipher.AES_128 XMLCipher.AES_128_GCM XMLCipher.AES_192 XMLCipher.AES_192_GCM XMLCipher.AES_256 XMLCipher.AES_256_GCM XMLCipher.SEED_128 XMLCipher.CAMELLIA_128 XMLCipher.CAMELLIA_192 XMLCipher.CAMELLIA_256 XMLCipher.CAMELLIA_192 XMLCipher.CAMELLIA_256 は MLCipher.TRIPLEDES
passPhrase		**文字列**	コンテンツを暗号化/復号化するために passPhrase として使用される文字列。passPhrase を指定する必要があります。passPhrase が指定されていない場合は、デフォルトの passPhrase が使用されます。passPhrase は、適切な暗号化アルゴリズムと併用する必要があります。たとえば TRIPLEDES を使用する場合、PassPhase を別の 24 バイトキーのみにすることができます。
secureTag		**文字列**	暗号化/復号化用に選択した XML 要素への XPath 参照。タグが指定されていない場合、ペイロード全体が暗号化/復号化されます。
secureTagContents	**false**	**ブール値**	XML 要素が暗号化されるかどうか、または XML 要素 false = 要素レベルの内容を指定するブール値。
keyCipherAlgorithm	**RSA_OAEP**	**文字列**	非対称鍵の暗号化/復号化に使用する暗号アルゴリズム。使用できる選択肢は XMLCipher.RSA_v1dot5 XMLCipher.RSA_OAEP XMLCipher.RSA_OAEP_11 です。デフォルト値は XMLCipher.RSA_OAEP です。
recipientKeyAlias		**文字列**	非対称鍵の暗号化または復号化の実行時に KeyStore から受信側の公開鍵または秘密鍵を取得する際に使用されるキーエイリアス。
keyOrTrustStoreParametersId		**文字列**	KeyStore インスタンスを参照してレジストリーで検索します。これは、送信者の trustStore または受信者の keyStore を表す KeyStore インスタンスを作成し、読み込む設定オプションに使用されます。
keyPassword		**文字列**	KeyStore から秘密鍵を取得するために使用されるパスワード。このキーは非対称復号化に使用されます。
digestAlgorithm	**SHA1**	**文字列**	RSA OAEP アルゴリズムで使用するダイジェストアルゴリズム。使用できる選択肢は XMLCipher.SHA1 XMLCipher.SHA256 XMLCipher.SHA512 です。デフォルト値は XMLCipher.SHA1 です。

Name	デフォルト	Java タイプ	説明
mgfAlgorithm	**MGF1_SHA1**	**文字列**	RSA OAEP アルゴリズムで使用する MGF アルゴリズム。EncryptionConstants.MGF1_SHA1 EncryptionConstants.MGF1_SHA256 EncryptionConstants.MGF1_SHA512 を選択できます。デフォルト値は EncryptionConstants.MGF1_SHA1 です。
addKeyValueForEncryptedKey	**true**	**ブール値**	暗号化キーの構造で KeyValue としてセッションキーの暗号化に使用される公開鍵を追加するかどうか。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

275.1.1. キー暗号アルゴリズム

Camel 2.12.0 より、デフォルトの キー暗号アルゴリズム は XMLCipher.RSA_v1dot5 ではなく XMLCipher.RSA_OAEP になりました。さまざまな攻撃により、XMLCipher.RSA_v1dot5 の使用は推奨されません。RSA v1.5 を鍵暗号アルゴリズムとして使用する要求は、鍵暗号アルゴリズムとして明示的に設定されていない限り拒否されます。

275.2. マーシャリング

ペイロードを暗号化するには、マーシャリング プロセッサーをルートに適用してから secureXML（） タグを適用する必要があります。

275.3. アンマーシャリング

ペイロードを復号するには、unmarshal プロセッサーをルートに適用し、その後に secureXML（） タグを付ける必要があります。

275.4. 例

以下は、Document、Element、および Content レベルでマーシャリングを実行する方法の例です。

275.4.1. 完全な Payload 暗号化/復号化

```
from("direct:start")
```

```
    .marshal().secureXML()
    .unmarshal().secureXML()
    .to("direct:end");
```

275.4.2. 部分的な Payload コンテンツの暗号化/復号化

```
String tagXPATH = "//cheesesites/italy/cheese";
boolean secureTagContent = true;
...
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent)
    .unmarshal().secureXML(tagXPATH, secureTagContent)
    .to("direct:end");
```

275.4.3. 部分的な Multi Node Payload Content Only encryption/decryption

```
String tagXPATH = "//cheesesites/*/cheese";
boolean secureTagContent = true;
...
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent)
    .unmarshal().secureXML(tagXPATH, secureTagContent)
    .to("direct:end");
```

275.4.4. passPhrase(password)を選択による部分的な Payload コンテンツのみの暗号化/復号化

```
String tagXPATH = "//cheesesites/italy/cheese";
boolean secureTagContent = true;
...
String passPhrase = "Just another 24 Byte key";
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent, passPhrase)
    .unmarshal().secureXML(tagXPATH, secureTagContent, passPhrase)
    .to("direct:end");
```

275.4.5. passPhrase(password)およびアルゴリズムを使用した部分的な Payload コンテンツのみの暗号化/復号化

```
import org.apache.xml.security.encryption.XMLCipher;
....
String tagXPATH = "//cheesesites/italy/cheese";
boolean secureTagContent = true;
String passPhrase = "Just another 24 Byte key";
String algorithm= XMLCipher.TRIPLEDES;
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent, passPhrase, algorithm)
    .unmarshal().secureXML(tagXPATH, secureTagContent, passPhrase, algorithm)
    .to("direct:end");
```

275.4.6. namespace がサポートされる部分的な Payload コンテンツ

Java DSL

```
final Map<String, String> namespaces = new HashMap<String, String>();
namespaces.put("cust", "http://cheese.xmlsecurity.camel.apache.org/");

final KeyStoreParameters tsParameters = new KeyStoreParameters();
tsParameters.setPassword("password");
tsParameters.setResource("sender.ts");

context.addRoutes(new RouteBuilder() {
   public void configure() {
      from("direct:start")
        .marshal().secureXML("//cust:cheesesites/italy", namespaces, true, "recipient",
                        testCypherAlgorithm, XMLCipher.RSA_v1dot5, tsParameters)
        .to("mock:encrypted");
   }
}
```

Spring XML

camelContext 定義の一部として定義される namespace プレフィックスは、secureXML 要素のデータ形式の secureTag 属性内でコンテキストで再利用できます。

```
<camelContext id="springXmlSecurityDataFormatTestCamelContext"
              xmlns="http://camel.apache.org/schema/spring"
              xmlns:cheese="http://cheese.xmlsecurity.camel.apache.org/">
   <route>
      <from uri="direct://start"/>
         <marshal>
            <secureXML secureTag="//cheese:cheesesites/italy"
                       secureTagContents="true"/>
         </marshal>
         ...
```

275.4.7. 非対称鍵の暗号化

Spring XML Sender

```
<!--  trust store configuration -->
<camel:keyStoreParameters id="trustStoreParams" resource="./sender.ts" password="password"/>

<camelContext id="springXmlSecurityDataFormatTestCamelContext"
              xmlns="http://camel.apache.org/schema/spring"
              xmlns:cheese="http://cheese.xmlsecurity.camel.apache.org/">
   <route>
      <from uri="direct://start"/>
```

```
        <marshal>
          <secureXML secureTag="//cheese:cheesesites/italy"
                secureTagContents="true"
                xmlCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
                keyCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
                recipientKeyAlias="recipient"
                keyOrTrustStoreParametersId="trustStoreParams"/>
        </marshal>
        ...
```

Spring XML Recipient

```
<!--  key store configuration -->
<camel:keyStoreParameters id="keyStoreParams" resource="./recipient.ks" password="password" />

<camelContext id="springXmlSecurityDataFormatTestCamelContext"
            xmlns="http://camel.apache.org/schema/spring"
            xmlns:cheese="http://cheese.xmlsecurity.camel.apache.org/">
  <route>
    <from uri="direct://encrypted"/>
      <unmarshal>
        <secureXML secureTag="//cheese:cheesesites/italy"
                secureTagContents="true"
                xmlCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
                keyCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
                recipientKeyAlias="recipient"
                keyOrTrustStoreParametersId="keyStoreParams"
                keyPassword="privateKeyPassword" />
      </unmarshal>
      ...
```

275.5. 依存関係

このデータ形式は **camel-xmlsecurity** コンポーネント内に提供されます。

第276章 SEDA コンポーネント

Camel バージョン 1.1 で利用可能

seda: コンポーネントは非同期 SEDA 動作を提供するため、メッセージは BlockingQueue で交換され、コンシューマーはプロデューサーとは別のスレッドで呼び出されます。

キューは 単一 の CamelContext 内でのみ表示されることに注意してください。CamelContext インスタンス全体で通信する場合（Web アプリケーション間の通信など）は、VM コンポーネントを参照してください。

このコンポーネントは、メッセージが処理される間に仮想マシンが終了した場合、あらゆる種類の永続性やリカバリーを実装しません。永続性、信頼性、または分散 SEDA が必要な場合は、JMS または ActiveMQ のいずれかを使用してみます。

TIP:*Synchronous* Direct コンポーネントは、プロデューサーがメッセージエクスチェンジを送信する際にコンシューマーの同期呼び出しを提供します。

276.1. URI 形式

```
seda:someName[?options]
```

someName には、現在の CamelContext 内のエンドポイントを一意に識別する文字列を使用できます。

URI には、?option=value&option=value&... という形式でクエリーオプションを追加できます。

276.2. オプション

SEDA コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
queueSize (advanced)	SEDA キューのデフォルト最大容量（つまり、保持できるメッセージの数）を設定します。		int

Name	説明	デフォルト	Type
concurrentConsumers (consumer)	エクスチェンジを処理するデフォルトの同時スレッド数を設定します。	1	int
defaultQueueFactory (advanced)	デフォルトのキューファクトリーを設定します。		Exchange>
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SEDA エンドポイントは、URI 構文を使用して設定します。

```
seda:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

276.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	***必要な*** キューの名前		文字列

276.2.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
size (common)	SEDA キューの最大容量（つまり、保持できるメッセージの数）。	2147483647	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean

Name	説明	デフォルト	Type
concurrentConsumers (consumer)	エクスチェンジを処理する同時スレッドの数。	1	int
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
limitConcurrentConsumers (consumer)	concurrentConsumer の数を最大 500 に制限するかどうか。デフォルトでは、エンドポイントが数値が大きい場合に例外が発生します。このオプションをオフにして、チェックを無効にすることができます。	true	boolean
multipleConsumers (consumer)	複数のコンシューマーを許可するかどうかを指定します。有効にすると、Publish-Subscribe messaging に SEDA を使用できます。つまり、SEDA キューにメッセージを送信し、各コンシューマーがメッセージのコピーを受け取ることができます。このオプションを有効にすると、すべてのコンシューマーエンドポイントでこのオプションを指定する必要があります。	false	boolean
pollTimeout (consumer)	ポーリング時に使用されるタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
purgeWhenStopping (consumer)	コンシューマー/ルートを停止する際にタスクキューをパージするかどうか。これにより、キューで保留中のメッセージが破棄されるため、より早く停止することができます。	false	boolean
blockWhenFull (producer)	メッセージを完全な SEDA キューに送信するスレッドが、キューの容量が使い切られるまでブロックするかどうか。デフォルトでは、キューが満杯であることを示す例外がスローされます。このオプションを有効にすると、呼び出しスレッドがブロックされ、メッセージが受け入れられるまで待機します。	false	boolean

Name	説明	デフォルト	Type
discardIfNoConsumers (producer)	アクティブなコンシューマーのないキューに送信するときに、プロデューサーがメッセージを破棄するかどうか（メッセージをキューに追加しないでください）。同時に有効にできるオプションは discardIfNoConsumers および failIfNoConsumers の 1 つのみです。	false	boolean
failIfNoConsumers (producer)	アクティブなコンシューマーのないキューに送信するときに、プロデューサーが例外を発生させて失敗するかどうか。同時に有効にできるオプションは discardIfNoConsumers および failIfNoConsumers の 1 つのみです。	false	boolean
タイムアウト （プロデューサー）	SEDA プロデューサーが非同期タスクの完了の待機を停止するまでのタイムアウト（ミリ秒単位）。タイムアウトを無効にするには、0 または負の値を使用します。	30000	Long
waitForTaskToComplete (producer)	非同期タスクが完了するまで待機すべきかどうかを指定するオプション。続行します。Always、Never、または IfReplyExpected の 3 つのオプションがサポートされます。最初の 2 つの値は self-explany です。IfReplyExpected の最後の値は、メッセージが Request Reply based の場合にのみ待機します。デフォルトオプションは IfReplyExpected です。	IfReplyExpected	WaitForTaskToComplete
queue （詳細）	エンドポイントによって使用されるキューインスタンスを定義します。このオプションは、カスタムキューインスタンスを使用するまれなユースケースのみとなります。		BlockingQueue
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

276.3. BLOCKINGQUEUE 実装の選択

Camel 2.12 から利用可能

デフォルトでは、SEDA コンポーネントは常に LinkedBlockingQueue を意図していますが、異なる実装を使用することもできますが、独自の BlockingQueue 実装を参照できます。この場合、size オプションは使用されません。

```
<bean id="arrayQueue" class="java.util.ArrayBlockingQueue">
  <constructor-arg index="0" value="10" ><!-- size -->
```

```
  <constructor-arg index="1" value="true" ><!-- fairness -->
</bean>

<!-- ... and later -->
<from>seda:array?queue=#arrayQueue</from>
```

または、BlockingQueueFactory 実装を参照する場合、3 つの実装は LinkedBlockingQueueFactory、ArrayBlockingQueueFactory、および PriorityBlockingQueueFactory が提供されます。

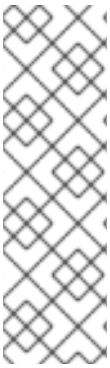
```
<bean id="priorityQueueFactory"
class="org.apache.camel.component.seda.PriorityBlockingQueueFactory">
  <property name="comparator">
    <bean class="org.apache.camel.demo.MyExchangeComparator" />
  </property>
</bean>

<!-- ... and later -->
<from>seda:priority?queueFactory=#priorityQueueFactory&size=100</from>
```

276.4. 要求応答の使用

SEDA コンポーネントは Request Reply の使用をサポートします。呼び出し元は Async ルートが完了するまで待機します。たとえば、以下のようになります。

```
from("mina:tcp://0.0.0.0:9876?textline=true&sync=true").to("seda:input");

from("seda:input").to("bean:processInput").to("bean:createResponse");
```

上記のルートには、受信要求を許可するポート 9876 に TCP リスナーがあります。リクエストは seda:input キューにルーティングされます。Request Reply メッセージであるため、レスポンスを待機します。seda:input キューのコンシューマーが完了すると、応答を元のメッセージの応答にコピーします。



注記

2 .2 まで： Request Reply over SEDA または VM は 2 つのエンドポイントでのみ機能します。A → B → C などのエンドポイントに送信してエンドポイントをチェーンすることは できません。A から B の間のみ。理由は実装ロジックが非常にシンプルです。3+ エンドポイントをサポートするために、ロジックは待機スレッド間の順序付けおよび通知の処理が非常に複雑になります。Camel 2.3 以降では改善され、エンドポイントをいくつでもチェーンすることができます。

276.5. 同時コンシューマー

デフォルトでは、SEDA エンドポイントは単一のコンシューマースレッドを使用しますが、同時実行コンシューマースレッドを使用するように設定できます。したがって、スレッドプールの代わりに以下を使用できます。

```
from("seda:stageName?concurrentConsumers=5").process(...)
```

この 2 つの間の違いと同様に、スレッドプール は負荷に応じてランタイム時に動的に増減する可能性があることに注意してください。一方、同時コンシューマーの数は常に固定されます。

276.6. スレッドプール

以下のように、スレッドプールを SEDA エンドポイントに追加することに注意してください。

```
from("seda:stageName").thread(5).process(...)
```

1 つは SEDA エンドポイントからの 2 つの BlockQueues で、もう 1 つは、スレッドプールのワークキューからのものであり、望ましい動作ではない可能性があります。代わりに、メッセージを同期的および非同期的に処理できるスレッドプールで Direct エンドポイントを設定する必要がある場合があります。以下に例を示します。

```
from("direct:stageName").thread(5).process(...)
```

concurrentConsumers オプションを使用して、SEDA エンドポイントでメッセージを処理するスレッドの数を直接設定することもできます。

276.7. 例

以下のルートでは、SEDA キューを使用してリクエストをこの非同期キューに送信し、別のスレッドでさらに処理するために fire-and-forget メッセージを送信し、このスレッド内の定数応答を元の呼び出し元に返します。

ここでは Hello World のメッセージを送信し、応答が OK であることを想定します。

「Hello World」メッセージは、さらなる処理のために、別のスレッドの SEDA キューから消費されます。これはユニットテストからのものであるため、ユニットテストでアサーションを実行できる モック エンドポイントに送信されます。

276.8. MULTIPLECONSUMERS の使用

Camel 2.2 で利用可能

この例では、2 つのコンシューマーを定義し、それらを Spring Bean として登録しました。

seda foo エンドポイントで multipleConsumers=true を指定しているため、これら 2 つのコンシューマーはメッセージ独自のコピーを pub-sub スタイルのメッセージングとして受け取ることができます。

Bean はユニットテストの一部として、メッセージをモックエンドポイントに送信しますが、@Consume を使用して seda キューから消費できることに注意してください。

276.9. キュー情報の抽出。

必要な場合は、以下のように JMX を使用せずにキューサイズなどの情報を取得できます。

```
SedaEndpoint seda = context.getEndpoint("seda:xxxx");
int size = seda.getExchanges().size();
```

276.10. 関連項目

- VM
- Disruptor
- Direct
- 非同期

第277章 JAVA OBJECT SERIALIZATION DATAFORMAT

Camel バージョン 2.12 から利用可能

シリアライゼーションとは、標準の Java シリアライゼーションメカニズムを使用してバイナリーペイロードを Java オブジェクトにアンマーシャリングしたり、Java オブジェクトをバイナリーブロブにマーシャリングしたりするデータ形式です。
たとえば、以下は Java シリアライゼーションを使用してバイナリーファイルをアンマーシャリングし、ObjectMessage として ActiveMQ に送信します。

```
from("file://foo/bar").
  unmarshal().serialization().
  to("activemq:Some.Queue");
```

277.1. オプション

Java Object Serialization データフォーマットは、以下に示す 1 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

277.2. 依存関係

このデータ形式は camel-core で提供されるため、追加の依存関係は必要ありません。

第278章 SERVICENOW コンポーネント

Camel バージョン 2.18 から利用可能

ServiceNow コンポーネントは、REST API 経由で ServiceNow プラットフォームへのアクセスを提供します。

注記

Camel 2.18.1 以降、コンポーネントは、デフォルトで Helsinki に複数のバージョンの ServiceNow プラットフォームをサポートします。サポートされるバージョンは 表278.1「API マッピング」 です。 表278.2「API マッピング」

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-servicenow</artifactId>
  <version>${camel-version}</version>
</dependency>
```

278.1. URI 形式

```
servicenow://instanceName?[options]
```

278.2. オプション

ServiceNow コンポーネントは、以下に示す 14 個のオプションをサポートします。

Name	説明	デフォルト	Type
instanceName (advanced)	ServiceNow インスタンス名		文字列
Configuration (advanced)	ServiceNow のデフォルト設定		ServiceNowConfiguration
apiUrl (producer)	ServiceNow REST API の URL		文字列

Name	説明	デフォルト	Type
userName (security)	ServiceNow ユーザーアカウント名		文字列
パスワード （セキュリティー）	ServiceNow アカウントパスワード		文字列
oauthClientId (security)	OAuth2 ClientID		文字列
oauthClientSecret (security)	OAuth2 ClientSecret		文字列
oauthTokenUrl (security)	OAuth トークン URL		文字列
proxyHost (advanced)	プロキシーホスト名		文字列
proxyPort (advanced)	プロキシーポート番号		整数
proxyUserName (security)	プロキシー認証のユーザー名		文字列
proxyPassword (security)	プロキシー認証のパスワード		文字列
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

ServiceNow エンドポイントは URI 構文を使用して設定されます。

```
servicenow:instanceName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

278.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
instanceName	**必須**: ServiceNow インスタンス名		文字列

278.2.2. クエリーパラメーター（44 パラメーター）：

Name	説明	デフォルト	Type
display (producer)	indicator Display フィールドが選択されている場合には、このパラメーターを true に設定します。このパラメーターを all に設定して、Display フィールドの値でスコアカードを返します。このパラメーターはデフォルトで true です。	true	文字列
displayValue (producer)	参照フィールドの表示値(true)、実際の値(false)、またはその両方（すべて）を返します（デフォルト：false）。	false	文字列
excludeReferenceLink (producer)	True 参照フィールドのテーブル API リンクを除外する（デフォルト： false）		ブール値
お気に入り （プロデューサー）	このパラメーターを true に設定すると、クエリーユーザーのお気に入りのスコアカードのみが返されます。		ブール値
includeAggregates (producer)	アグリゲートがすでに適用されたかどうかを含め、このパラメーターを true に設定して、インディケーターで利用可能な集約をすべて常に返します。値の指定がない場合は、このパラメーターはデフォルトで false に設定され、アグリゲートは返されません。		ブール値
includeAvailableAggregates (producer)	アグリゲートが適用されていない場合に、このパラメーターを true に設定して、インジケーターで利用可能な集約をすべて返します。値の指定がない場合は、このパラメーターはデフォルトで false に設定され、アグリゲートは返されません。		ブール値
includeAvailableBreakdowns (producer)	インジケーターで利用可能な内訳をすべて返すには、このパラメーターを true に設定します。値の指定がない場合は、このパラメーターのデフォルトは false で、内訳を返さない。		ブール値
includeScoreNotes (producer)	スコアに関連付けられた全注記を返すには、このパラメーターを true に設定します。note 要素には、注記テキストと、注記の追加時の作成者およびタイムスタンプが含まれます。		ブール値

Name	説明	デフォルト	Type
includeScores (producer)	スコアカードのすべてのスコアを返すには、このパラメーターを true に設定します。値の指定がない場合は、このパラメーターのデフォルトは false で、最新のスコア値のみを返します。		ブール値
inputDisplayValue (producer)	True: 入力フィールドの raw 値を設定します（デフォルト： false）。		ブール値
キー （プロデューサー）	キーインジケーターのスコアカードのみを返すには、このパラメーターを true に設定します。		ブール値
モデル （プロデューサー）	リクエストモデルと応答モデルの両方を定義します。		文字列
perPage (producer)	各クエリーが返すことができるスコアカードの最大数を入力します。デフォルト値は 10 で、最大値は 100 です。	10	整数
リリース （プロデューサー）	ターゲットに設定する ServiceNow リリース。デフォルトは Helsinki です。https://docs.servicenow.com	HELSINKI	ServiceNowRelease
requestModels (producer)	リクエストモデルを定義します。		文字列
resource (producer)	デフォルトのリソースは、ヘッダー CamelServiceNowResource で上書きできます。		文字列
responseModels (producer)	応答モデルを定義します。		文字列
sortBy (producer)	結果をソートする際に使用する値を指定します。デフォルトでは、クエリーはレコードを値別に並べ替えます。		文字列
sortDir (producer)	ソート方向（昇順または降順）を指定します。デフォルトでは、クエリーはレコードを降順に分類します。sysparm_sortdir=asc を使用して昇順で並び替えます。		文字列
suppressAutoSysField (producer)	システムフィールドの自動生成を抑制するために true（デフォルト： false）		ブール値
suppressPaginationHeader (producer)	この値を true に設定して、応答から Link ヘッダーを削除します。Link ヘッダーを使用すると、クエリーに一致するレコードの数がクエリー制限を超えた場合、データの追加ページをリクエストできます。		ブール値

Name	説明	デフォルト	Type
テーブル （プロデューサー）	デフォルトのテーブルは CamelServiceNowTable ヘッダーで上書きできます。		文字列
ターゲット （プロデューサー）	このパラメーターを true に設定すると、ターゲットを持つスコアカードのみが返されます。		ブール値
topLevelOnly (producer)	親がカタログであるカテゴリーのみを取得します。		ブール値
apiVersion （詳細）	ServiceNow REST API バージョン（デフォルトは latest）		文字列
dateFormat (advanced)	Json シリアライゼーション/デシリアライズに使用される日付形式	yyyy-MM-dd	文字列
dateTimeFormat (advanced)	Json シリアライゼーション/デシリアライズに使用される日付形式	YYYY-MM-dd HH:mm:ss	文字列
httpClientPolicy (advanced)	http-client を設定するには、以下を行います。		HTTPClientPolicy
Mapper (advanced)	リクエスト/応答に使用する Jackson の ObjectMapper を設定します。		ObjectMapper
proxyAuthorizationPolicy (advanced)	プロキシー認証を設定します。		ProxyAuthorization ポリシー
retrieveTargetRecordOn Import (advanced)	インポートセット api の使用時にターゲットレコードを取得するには、このパラメーターを true に設定します。インポートセットの結果はターゲットレコードに置き換えられます。	false	ブール値
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
timeFormat (advanced)	Json シリアライゼーション/デシリアライズに使用される時間形式	HH:mm:ss	文字列
proxyHost (proxy)	プロキシーホスト名		文字列

Name	説明	デフォルト	Type
proxyPort (proxy)	プロキシーポート番号		整数
apiUrl (security)	ServiceNow REST API の URL		文字列
oauthClientId (security)	OAuth2 ClientID		文字列
oauthClientSecret (security)	OAuth2 ClientSecret		文字列
oauthTokenUrl (security)	OAuth トークン URL		文字列
パスワード （セキュリティー）	**必要な** ServiceNow アカウントのパスワード（指定する必要があります）		文字列
proxyPassword (security)	プロキシー認証のパスワード		文字列
proxyUserName (security)	プロキシー認証のユーザー名		文字列
sslContextParameters (security)	SSLContextParameters を使用してセキュリティーを設定します。http://camel.apache.org/camel-configuration-utilities.htmlを参照してください。		SSLContextParameters
userName (security)	**必要な** ServiceNow ユーザーアカウント名、指定する必要があります。		文字列

278.3. HEADERS

Name	Type	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowResource	文字列	-	-	アクセスするリソース

Name	Type	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowAction	文字列	-	-	実行するアクション
CamelServiceNowActionSubject	-	-	文字列	アクションを適用するサブジェクト
CamelServiceNowModel	クラス	-	-	データモデル
CamelServiceNowRequestModel	クラス	-	-	要求データモデル
CamelServiceNowResponseModel	クラス	-	-	レスポンスデータモデル
CamelServiceNowOffsetNext	-	-	-	-
CamelServiceNowOffsetPrev	-	-	-	-
CamelServiceNowOffsetFirst	-	-	-	-

Name	Type	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowOffsetLast	-	-	-	-
CamelServiceNowContentType	-	-	-	-
CamelServiceNowContentEncoding	-	-	-	-
CamelServiceNowContentMeta	-	-	-	-
CamelServiceNowSysId	文字列	sys_id	-	-
CamelServiceNowUserSysId	文字列	user_sysid	-	-
CamelServiceNowUserId	文字列	user_id	-	-
CamelServiceNowCartItemId	文字列	cart_item_id	-	-

Name	Type	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowFileName	文字列	file_name	-	-
CamelServiceNowTable	文字列	table_name	-	-
CamelServiceNowTableSysId	文字列	table_sys_id	-	-
CamelServiceNowEncryptionContext	文字列	encryption_context	-	-
CamelServiceNowCategory	文字列	sysparm_category	-	-
CamelServiceNowType	文字列	sysparm_type	-	-
CamelServiceNowCatalog	文字列	sysparm_catalog	-	-
CamelServiceNowQuery	文字列	sysparm_query	-	-

Name	Type	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowDisplayValue	文字列	sysparm_display_value	displayValue	-
CamelServiceNowInputDisplayValue	ブール値	sysparm_input_display_value	inputDisplayValue	-
CamelServiceNowExcludeReferenceLink	ブール値	sysparm_exclude_reference_link	excludeReferenceLink	-
CamelServiceNowFields	文字列	sysparm_fields	-	-
CamelServiceNowLimit	整数	sysparm_limit	-	-
CamelServiceNowText	文字列	sysparm_text	-	-
CamelServiceNowOffset	整数	sysparm_offset	-	-
CamelServiceNowView	文字列	sysparm_view	-	-

Name	Type	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowSuppressAutoSysField	ブール値	sysparm_suppress_auto_sys_field	suppressAutoSysField	-
CamelServiceNowSuppressPaginationHeader	Boolean	sysparm_suppress_pagination_header	suppressPaginationHeader	-
CamelServiceNowMinFields	文字列	sysparm_min_fields	-	-
CamelServiceNowMaxFields	文字列	sysparm_max_fields	-	-
CamelServiceNowSumFields	文字列	sysparm_sum_fields	-	-
CamelServiceNowAvgFields	文字列	sysparm_avg_fields	-	-
CamelServiceNowCount	ブール値	sysparm_count	-	-
CamelServiceGroupBy	文字列	sysparm_group_by	-	-

Name	Type	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceOrderBy	文字列	sysparm_order_by	-	-
CamelServiceHaving	文字列	sysparm_having	-	-
CamelServiceNowUUID	文字列	sysparm_uuid	-	-
CamelServiceNowBreakdown	文字列	sysparm_breakdown	-	-
CamelServiceNowIncludeScores	ブール値	sysparm_include_scores	includeScores	-
CamelServiceNowIncludeScoreNotes	ブール値	sysparm_include_score_notes	includeScoreNotes	-
CamelServiceNowIncludeAggregates	ブール値	sysparm_include_aggregates	includeAggregates	-
CamelServiceNowIncludeAvailableBreakdowns	ブール値	sysparm_include_available_breakdowns	includeAvailableBreakdowns	-

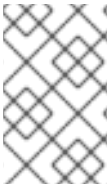
Name	Type	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowIncludeAvailableAggregates	ブール値	sysparm_include_available_aggregates	includeAvailableAggregates	-
CamelServiceNowFavorites	ブール値	sysparm_favorites	お気に入り	-
CamelServiceNowKey	ブール値	sysparm_key	key	-
CamelServiceNowTarget	ブール値	sysparm_target	target	-
CamelServiceNowDisplay	文字列	sysparm_display	display	-
CamelServiceNowPerPage	整数	sysparm_per_page	perPage	-
CamelServiceNowSortBy	文字列	sysparm_sortby	sortBy	-
CamelServiceNowSortDir	文字列	sysparm_sortdit	sortDir	-

Name	Type	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowContains	文字列	sysparm_contains	-	-
CamelServiceNowTags	文字列	sysparm_tags	-	-
CamelServiceNowPage	文字列	sysparm_page	-	-
CamelServiceNowElementsFilter	文字列	sysparm_elements_filter	-	-
CamelServiceNowBreakdownRelation	文字列	sysparm_breakdown_relation	-	-
CamelServiceNowDataSource	文字列	sysparm_data_source	-	-
CamelServiceNowTopLevelOnly	ブール値	sysparm_top_level_only	topLevelOnly	-
CamelServiceNowApiVersion	文字列	-	-	REST API バージョン

Name	Type	ServiceNow API パラメーター	エンドポイントオプション	説明
CamelServiceNowResponseMeta	マップ	-	-	応答と共に提供されるメタデータ

表278.1 API マッピング

CamelServiceNowResource	CamelServiceNowAction	メソッド	API URI
テーブル	RETRIEVE	GET	/api/now/v1/table/{table_name}/{sys_id}
	CREATE	POST	/api/now/v1/table/{table_name}
	MODIFY	PUT	/api/now/v1/table/{table_name}/{sys_id}
	DELETE	DELETE	/api/now/v1/table/{table_name}/{sys_id}
	UPDATE	PATCH	/api/now/v1/table/{table_name}/{sys_id}
AGGREGATE	RETRIEVE	GET	/api/now/v1/stats/{table_name}
IMPORT	RETRIEVE	GET	/api/now/import/{table_name}/{sys_id}
	CREATE	POST	/api/now/import/{table_name}



注記

Fuji REST API ドキュメント

表278.2 API マッピング

Camel ServiceNowResource	Camel ServiceNowAction	Camel ServiceNowActionSubject	メソッド	API URI
テーブル	RETRIEVE		GET	/api/now/v1/table/{table_name}/{sys_id}
	CREATE		POST	/api/now/v1/table/{table_name}
	MODIFY		PUT	/api/now/v1/table/{table_name}/{sys_id}
	DELETE		DELETE	/api/now/v1/table/{table_name}/{sys_id}
	UPDATE		PATCH	/api/now/v1/table/{table_name}/{sys_id}
AGGREGATE	RETRIEVE		GET	/api/now/v1/stats/{table_name}
IMPORT	RETRIEVE		GET	/api/now/import/{table_name}/{sys_id}
	CREATE		POST	/api/now/import/{table_name}
ATTACHMENT	RETRIEVE		GET	/api/now/api/now/attachment/{sys_id}
	コンテンツ		GET	/api/now/attachment/{sys_id}/file
	UPLOAD		POST	/api/now/api/now/attachment/file
	DELETE		DELETE	/api/now/attachment/{sys_id}
スコアカード	RETRIEVE	PERFORMANCE_ANALYTICS	GET	/api/now/pa/scorecards

Camel ServiceNowResource	Camel ServiceNowAction	Camel ServiceNowActionSubject	メソッド	API URI
MISC	RETRIEVE	USER_ROLE_INHERITANCE	GET	/api/global/user_role_inheritance
	CREATE	IDENTIFY_RECONCILE	POST	/api/now/identifyreconcile
SERVICE_CATALOG	RETRIEVE		GET	/sn_sc/servicecatalog/catalogs/{sys_id}
	RETRIEVE	カテゴリー	GET	/sn_sc/servicecatalog/catalogs/{sys_id}/categories
SERVICE_CATALOG_ITEMS	RETRIEVE		GET	/sn_sc/servicecatalog/items/{sys_id}
	RETRIEVE	SUBMIT_GUIDE	POST	/sn_sc/servicecatalog/items/{sys_id}/submit_guide
	RETRIEVE	CHECKOUT_GUIDE	POST	/sn_sc/servicecatalog/items/{sys_id}/checkout_guide
	CREATE	SUBJECT_CART	POST	/sn_sc/servicecatalog/items/{sys_id}/add_to_cart
	CREATE	SUBJECT_PRODUCER	POST	/sn_sc/servicecatalog/items/{sys_id}/submit_producer
SERVICE_CATALOG_CARTS	RETRIEVE		GET	/sn_sc/servicecatalog/cart
	RETRIEVE	DELIVERY_ADDRESS	GET	/sn_sc/servicecatalog/cart/delivery_address/{user_id}

Camel ServiceNowResource	Camel ServiceNowAction	Camel ServiceNowActionSubject	メソッド	API URI
	RETRIEVE	チェックアウト	POST	/sn_sc/servicecatalog/cart/checkout
	UPDATE		POST	/sn_sc/servicecatalog/cart/{cart_item_id}
	UPDATE	チェックアウト	POST	/sn_sc/servicecatalog/cart/submit_order
	DELETE		DELETE	/sn_sc/servicecatalog/cart/{sys_id}/empty
SERVICE_CATALOG_CATEGORIES	RETRIEVE		GET	/sn_sc/servicecatalog/categories/{sys_id}

注記

Helsinki REST API ドキュメント

278.4. 使用例：

10 Incidents を取得

```
context.addRoutes(new RouteBuilder() {
    public void configure() {
        from("direct:servicenow")
            .to("servicenow:{{env:SERVICENOW_INSTANCE}}"
                + "?userName={{env:SERVICENOW_USERNAME}}"
```

```
            + "&password={{env:SERVICENOW_PASSWORD}}"
            + "&oauthClientId={{env:SERVICENOW_OAUTH2_CLIENT_ID}}"
            + "&oauthClientSecret={{env:SERVICENOW_OAUTH2_CLIENT_SECRET}}"
        .to("mock:servicenow");
    }
});

FluentProducerTemplate.on(context)
    .withHeader(ServiceNowConstants.RESOURCE, "table")
    .withHeader(ServiceNowConstants.ACTION, ServiceNowConstants.ACTION_RETRIEVE)
    .withHeader(ServiceNowConstants.SYSPARM_LIMIT.getId(), "10")
    .withHeader(ServiceNowConstants.TABLE, "incident")
    .withHeader(ServiceNowConstants.MODEL, Incident.class)
    .to("direct:servicenow")
    .send();
```

第279章 SERVLET コンポーネント

Camel バージョン 2.0 で利用可能

servlet: コンポーネントは、公開されたサーブレットにバインドされる HTTP エンドポイントに到達する HTTP リクエストを使用するために HTTP ベースのエンドポイントを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-servlet</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

INFO: Stream.Servlet はストリームベースで、受信する入力がストリームとして Camel に送信されます。つまり、は 1 回 のみストリームのコンテンツを読み取ることができます。メッセージボディーが空であるか、データに複数回アクセスする必要がある場合（マルチキャストの実行や再配信エラー処理など）、ストリームキャッシュを使用するか、メッセージボディーを複数回読み取ることのできる String に変換する必要があります。

279.1. URI 形式

```
servlet://relative_path[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

279.2. オプション

Servlet コンポーネントは、以下に挙げる 8 個のオプションをサポートします。

Name	説明	デフォルト	Type
servletName (consumer)	使用するサーブレットのデフォルト名。デフォルトの名前は CamelServlet です。		文字列

Name	説明	デフォルト	Type
httpRegistry (consumer)	カスタムの org.apache.camel.component.servlet.HttpRegistry を使用します。		HttpRegistry
attachmentMultipart Binding （コンシューマー）	マルチパート/フォームデータを Camel Exchange で添付として自動バインドするかどうか。attachmentMultipartBinding=true オプションおよび disableStreamCache=false オプションは併用できません。AttachmentMultipartBinding を使用するには disableStreamCache を削除します。サーブレットの使用時にこの設定を有効にするためにサーブレット固有の設定を必要とする可能性があるため、これはデフォルトで無効になります。	false	boolean
httpBinding (advanced)	カスタムの HttpBinding を使用して Camel メッセージと HttpClient 間のマッピングを制御する場合。		HttpBinding
httpConfiguration (advanced)	共有 HttpConfiguration をベース設定として使用します。		HttpConfiguration
allowJavaSerialized Object (advanced)	リクエストが context-type=application/x-java-serialized-object を使用する場合に java のシリアライズを許可するかどうか。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Servlet エンドポイントは、URI 構文を使用して設定します。

```
servlet:contextPath
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

279.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
contextPath	使用する context-path が **必要です**。		文字列

279.2.2. クエリーパラメーター（21 パラメーター）：

Name	説明	デフォルト	Type
disableStreamCache (common)	Servlet からの raw 入力ストリームがキャッシュされているかどうかを決定します（Camel はストリームをメモリー内/オーバーフローでファイル、ストリームキャッシング）キャッシュに読み取ります。デフォルトでは、Camel は Servlet 入力ストリームをキャッシュして、複数回ロードし、Camel がストリームからすべてのデータを取得できるようにします。ただし、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。たとえば、ファイルまたは他の永続ストアに直接ストリーミングする場合などに、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。DefaultHttpBinding は、ストリームの読み取りを複数回サポートするために、このオプションが false の場合は、リクエスト入力ストリームをストリームキャッシュにコピーし、メッセージボディーに配置します。Servlet を使用してエンドポイントをブリッジ/プロキシーする場合、このオプションを有効にしてパフォーマンスを向上することを検討してください。メッセージペイロードを複数回読み取る必要がない場合は、このオプションを有効にします。http/http4 プロデューサーは、デフォルトでは応答本体ストリームをキャッシュします。このオプションを true に設定すると、プロデューサーは応答ボディーストリームをキャッシュしませんが、応答ストリームをメッセージボディーとして使用します。	false	boolean
headerFilterStrategy (common)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
httpBinding (common)	カスタムの HttpBinding を使用して Camel メッセージと HttpClient 間のマッピングを制御する場合。		HttpBinding
非同期 （コンシューマー）	非同期モードで動作するようにコンシューマーを設定する	false	boolean

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
チャンク （コンシューマー）	このオプションが false の場合、サーブレットは HTTP ストリーミングを無効にし、応答に content-length ヘッダーを設定します。	true	boolean
httpMethodRestrict (consumer)	HttpMethod が一致する場合にのみ消費（GET/POST/PUT など）を許可するために使用されます。複数のメソッドはカンマで区切って指定できます。		文字列
matchOnUriPrefix (consumer)	完全一致がない場合、コンシューマーが URI プレフィックスに一致することでターゲットコンシューマーの検索を試行するかどうか。	false	boolean
responseBufferSize (consumer)	javax.servlet.ServletResponse でカスタムバッファーサイズを使用します。		整数
servletName (consumer)	使用するサーブレットの名前	Camel Servlet	文字列
transferException (consumer)	有効で Exchange がコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object のコンテンツタイプとして応答でシリアライズされたかどうか。プロデューサー側では、例外は HttpOperationFailedException ではなくデシリアライズされ、そのままスローされます。原因となる例外はシリアライズされている必要があります。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean

Name	説明	デフォルト	Type
attachmentMultipartBinding (consumer)	マルチパート/フォームデータを Camel Exchange で添付として自動バインドするかどうか。attachmentMultipartBinding=true オプションおよび disableStreamCache=false オプションは併用できません。AttachmentMultipartBinding を使用するには disableStreamCache を削除します。サーブレットの使用時にこの設定を有効にするためにサーブレット固有の設定を必要とする可能性があるため、これはデフォルトで無効になります。	false	boolean
eagerCheckContentAvailable (consumer)	content-length ヘッダーが 0 の場合に、HTTP リクエストにコンテンツがあるかどうかをアクティブにチェックするかどうか。これは、HTTP クライアントがストリーミングデータを送信しない場合に有効にすることができます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
optionsEnabled (consumer)	このサーブレットコンシューマーに HTTP OPTIONS を有効にするかどうかを指定します。デフォルトでは OPTIONS はオフになっています。	false	boolean
traceEnabled (consumer)	このサーブレットコンシューマーに対して HTTP TRACE を有効にするかどうかを指定します。デフォルトでは、TRACE はオフになっています。	false	boolean
mapHttpMessageBody (advanced)	このオプションが true の場合、エクスチェンジの IN エクスチェンジボディーは HTTP ボディーにマッピングされます。false に設定すると HTTP マッピングが回避されます。	true	boolean
mapHttpMessageFormUrlEncodedBody (advanced)	このオプションが true の場合、エクスチェンジの IN Exchange Form Encoded body が HTTP にマッピングされます。これを false に設定すると、HTTP Form Encoded body マッピングを回避します。	true	boolean

Name	説明	デフォルト	Type
mapHttpMessageHeaders (advanced)	このオプションが true の場合、エクスチェンジの IN エクスチェンジヘッダーは HTTP ヘッダーにマッピングされます。false に設定すると、HTTP ヘッダーのマッピングが回避されます。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

279.3. メッセージヘッダー

Camel は、HTTP コンポーネントと同じ Message Headers を適用します。

Camel は すべて の request.parameter および request.headers も設定します。たとえば、クライアントリクエストに URL http://myserver/myserver?orderid=123 がある場合、エクスチェンジには値 123 の orderid という名前のヘッダーが含まれます。

279.4. 用途

Servlet コンポーネントによって生成されたエンドポイントからのみ消費できます。そのため、Camel ルートへの入力としてのみ使用してください。他の HTTP エンドポイントに対して HTTP 要求を発行するには、HTTP コンポーネントを使用します。

279.5. アプリケーションサーバーブートクラスパスへの CAMEL JAR の配置

camel-core、camel-servlet など、アプリケーションサーバーのブートクラスパス（通常は lib ディレクトリーなど）に Camel JAR を配置する場合、サーブレットマッピングリストは、アプリケーションサーバーで複数のデプロイされた Camel アプリケーション間で共有されることに注意してください。

通常、アプリケーションサーバーのブートクラスパスに Camel JAR を配置することはベストプラクティスではありません。

そのため、web.xml などのように、各 Camel アプリケーションでカスタムおよび一意なサーブレット名を定義する 必要があり ます。

```
<servlet>
```

```
  <servlet-name>MyServlet</servlet-name>
  <servlet-class>org.apache.camel.component.servlet.CamelHttpTransportServlet</servlet-
class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>MyServlet</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Camel エンドポイントにはサーブレット名も含まれる

```
<route>
  <from uri="servlet://foo?servletName=MyServlet"/>
  ...
</route>
```

Camel 2.11 以降では、Camel はこの重複を検出し、アプリケーションの起動に失敗します。以下のようにサーブレット init-parameter ignoreDuplicateServletName を true に設定すると、この重複を無視するように制御できます。

```
  <servlet>
    <servlet-name>CamelServlet</servlet-name>
    <display-name>Camel Http Transport Servlet</display-name>
    <servlet-class>org.apache.camel.component.servlet.CamelHttpTransportServlet</servlet-
class>
    <init-param>
      <param-name>ignoreDuplicateServletName</param-name>
      <param-value>true</param-value>
    </init-param>
  </servlet>
```

しかし、この重複クラッシュを回避するために、各 Camel アプリケーションに一意の servlet-name を使用することを 強く推奨 します。

279.6. 例

INFO: Camel 2.7 以降では、Spring Web アプリケーションで サーブレット を簡単に使用できます。詳細は、「サーブレット Tomcat の例」を参照してください。

この例では、http://localhost:8080/camel/services/hello で HTTP サービスを公開するルートを定義します。
まず、通常の Web コンテナーまたは OSGi サービスを介して CamelHttpTransportServlet を公開する

必要があります。
Web.xml ファイルを使用して、以下のように CamelHttpTransportServlet を公開します。

次に、以下のようにルートを定義できます。

注記

camel-servlet エンドポイントの相対パスを指定 します。Http トランスポートをパブリッシュされたサーブレットでバインディングし、サーブレットのアプリケーションコンテキストパスを認識しないため、camel-servlet エンドポイントは相対パスを使用してエンドポイントの URL を指定します。クライアントはサーブレット公開アドレス ("http://localhost:8080/camel/services")+ RELATIVE_PATH("/hello")を介して camel-servlet エンドポイントにアクセスできます。

279.6.1. Spring 3.x を使用する場合の例

「Servlet Tomcat Example」を参照してください。

279.6.2. Spring 2.x を使用する場合の例

Camel/Spring アプリケーションで Servlet コンポーネントを使用する場合、Servlet コンポーネントの起動後 に Spring ApplicationContext をロードする必要があることがよくあります。これは、ContextLoaderListener の代わりに Spring の ContextLoaderServlet を使用して実行できます。この場合、以下のように CamelHttpTransportServlet の後に ContextLoaderServlet を起動する必要があります。

```
<web-app>
  <servlet>
      <servlet-name>CamelServlet</servlet-name>
      <servlet-class>
         org.apache.camel.component.servlet.CamelHttpTransportServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
      <servlet-name>SpringApplicationContext</servlet-name>
      <servlet-class>
         org.springframework.web.context.ContextLoaderServlet
      </servlet-class>
      <load-on-startup>2</load-on-startup>
  </servlet>
<web-app>
```

279.6.3. OSGi を使用する場合の例

Camel 2.6.0 から、このように SpringDM を利用して CamelHttpTransportServlet を OSGi サービスとして公開できます。

次に、以下のように Camel ルートでこのサービスを使用します。

Camel 2.6 より前のバージョンでは、Actorivator を 使用 して OSGi プラットフォームで CamelHttpTransportServlet を公開することができます。

279.6.4. Spring Boot での使用

Camel 2.19.0 以降、camel-servlet-starter ライブラリーは、'/camel/'' コンテキストパス下の REST エンドポイントをすべて自動的にバインドします。以下の表は、camel-servlet-starter ライブラリーで利用可能なその他の設定プロパティーをまとめたものです。Camel サーブレットの自動マッピングも無効にできます。*

Spring Boot プロパティー	デフォルト	説明
camel.component.servlet.mapping.enabled	**true**	Servlet コンポーネントと Spring Web コンテキストへの自動マッピングを有効にします。
camel.component.servlet.mapping.context-path	**/camel/***	自動マッピングのサーブレットコンポーネントによって使用されるコンテキストパス
camel.component.servlet.mapping.servlet-name	**CamelServlet**	Camel サーブレットの名前

279.7. 関連項目

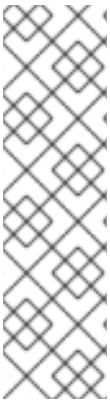
- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

- サーブレット Tomcat の例
- Servlet Tomcat No Spring Example
- HTTP
- Jetty

279.8. SERVLETLISTENER COMPONENT

Camel 2.11 から利用可能

このコンポーネントは、Web アプリケーションで Camel アプリケーションのブートストラップに使用されます。たとえば、前者は Camel をブートストラップする独自の方法を見つけるか、Spring などのサードパーティーフレームワークに依存してこれを行う必要があります。



注記

サイドバー のこのコンポーネントは Servlet 2.x 以降をサポートします。つまり、古い Web コンテナーでも機能します。これは、このコンポーネントの目的です。Servlet 2.x は web.xml ファイルを設定として使用する必要があります。Servlet 3.x コンテナーでは、@WebListener を使用してアノテーション駆動設定を使用して Camel をブートストラップし、独自のクラスを実装できます。ここで Camel をブートストラップできます。これにより、エンドユーザーが Camel を簡単に設定できるという課題があります。これにより、古い授業の web.xml ファイルで無料になります。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-servletlistener</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

279.8.1. 使用

抽象クラス org.apache.camel.component.servletlistener.CamelServletContextListener の実装のいずれかを選択する必要があります。

- *JndiRegistry を使用してレジストリーに JNDI を利用する JndiCamelServletContextListener。*
- *SimpleRegistry を使用して java.util.Map をレジストリーとして活用する SimpleCamelServletContextListener。*

これを使用するには、以下のように WEB-INF/web.xml ファイルで org.apache.camel.component.servletlistener.CamelServletContextListener を設定する必要があります。

279.8.2. オプション

org.apache.camel.component.servletlistener.CamelServletContextListener は、web.xml ファイルで context-param として設定できる以下のオプションをサポートします。

オプション	型	説明
propertyPlaceholder.XXX		Camel でプロパティープレースホルダーを設定します。ロケーションを設定するには、オプションの前に「propertyPlaceholder.」を追加する必要があります。たとえば、name には propertyPlaceholder.location を使用します。Properties コンポーネントからすべてのオプションを設定できます。
jmx.XXX		JMX を設定します。JMX を無効にする場合などに、オプションの前に「jmx.」を追加する必要があります。jmx.disabled を名前として使用します。**org.apache.camel.spi.ManagementAgent** からすべてのオプションを設定できます。JMX ページに記載されるオプション。
name	**文字列**	CamelContext の名前を設定します。
messageHistory	**ブール値**	**Camel 2.12.2:** メッセージ履歴を有効または無効にするかどうか（デフォルトでは有効）。
streamCache	**ブール値**	Stream キャッシュを有効にするかどうか。
trace	**ブール値**	トレーサーを有効にするかどうか。
delayer	**Long**	Delay Interceptor に delay の値を設定します。

オプション	型	説明
handleFault	**ブール値**	処理の失敗を有効にするかどうか。
errorHandlerRef	**文字列**	使用するコンテキストスコープのエラーハンドラーを参照します。
autoStartup	**ブール値**	Camel の起動時にすべてのルートを開始するかどうか。
useMDCLogging	**ブール値**	MDC ロギングを使用するかどうか。
useBreadcrumb	**ブール値**	breadcrumb を使用するかどうか。
managementNamePattern	**文字列**	JMX MBean にカスタム命名パターンを設定します。
threadNamePattern	**文字列**	スレッドにカスタム命名パターンを設定します。
properties.XXX		**CamelContext.getProperties** でカスタムプロパティーを設定するには、以下を行います。これは使用中のseldom です。
routebuilder.XXX		使用するルートを設定します。詳細は、以下を参照してください。
CamelContextLifecycle		**org.apache.camel.component.servletlistener.CamelContextLifecycle** の実装の FQN クラス名を参照します。これにより、CamelContext の起動または停止の前後にカスタムコードを実行できます。詳細は、以下を参照してください。
XXX		CamelContext で任意のオプションを設定します。

279.8.3. 例

「Servlet Tomcat No Spring Example」を参照してください。

279.8.4. 作成した CamelContext へのアクセス

Camel 2.14/2.13.3/2.12.5 から利用可能

作成された CamelContext は、キー "CamelContext" を持つ属性として ServletContext に格納されます。以下のように ServletContext を保持できる場合は、CamelContext を保持することができます。

```
ServletContext sc = ...
CamelContext camel = (CamelContext) sc.getAttribute("CamelContext");
```

279.8.5. ルートの設定

web.xml ファイルで使用するルートを設定する必要があります。これは複数の方法で実行できますが、すべてのパラメーターの前に "routeBuilder" を付ける必要があります。

279.8.5.1. RouteBuilder クラスの使用

デフォルトでは、Camel は以下のように param-value が Camel RouteBuilder クラスの FQN クラス名であることを前提としています。

```
<context-param>
  <param-name>routeBuilder-MyRoute</param-name>
  <param-value>org.apache.camel.component.servletlistener.MyRoute</param-value>
</context-param>
```

以下に示すように、同じパラメーター値に複数のクラスを指定できます。

```
<context-param>
  <param-name>routeBuilder-routes</param-name>
  <!-- we can define multiple values separated by comma -->
  <param-value>
    org.apache.camel.component.servletlistener.MyRoute,
    org.apache.camel.component.servletlistener.routes.BarRouteBuilder
  </param-value>
</context-param>
```

実行時に、パラメーターの名前には意味がありません。"routeBuilder" で開始する必要があるのは一意でなければいけません。上記の例では、「routeBuilder-routes」があります。ただし、

「routeBuilder.foo」という名前を付けることもできます。

279.8.5.2. パッケージスキャンの使用

パッケージスキャンを使用するように Camel に指示することもできます。つまり、RouteBuilder タイプのすべてのクラスに対して指定のパッケージを検索し、Camel ルートとして自動的に追加されます。これを実行するには、以下のように値を "packagescan:" のプレフィックスにする必要があります。

```
<context-param>
  <param-name>routeBuilder-MyRoute</param-name>
  <!-- define the routes using package scanning by prefixing with packagescan: -->
  <param-value>packagescan:org.apache.camel.component.servletlistener.routes</param-value>
</context-param>
```

279.8.5.3. XML ファイルの使用

XML DSL を使用して Camel ルートを定義することもできますが、Spring または Blueprint を使用しないため、XML ファイルには Camel ルートのみを含めることができます。

web.xml では、以下のように "classpath"、"file"、または "http" の URL から指定できる XML ファイルを参照します。

```
<context-param>
  <param-name>routeBuilder-MyRoute</param-name>
  <param-value>classpath:routes/myRoutes.xml</param-value>
</context-param>
```

XML ファイルは以下のようになります。

routes/myRoutes.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- the xmlns="http://camel.apache.org/schema/spring" is needed -->
<routes xmlns="http://camel.apache.org/schema/spring">

  <route id="foo">
    <from uri="direct:foo"/>
    <to uri="mock:foo"/>
  </route>

  <route id="bar">
```

```
    <from uri="direct:bar"/>
    <to uri="mock:bar"/>
  </route>

</routes>
```

XML ファイルでは、root タグは <routes> で、名前空間 "http://camel.apache.org/schema/spring" を使用する必要があります。この名前空間には名前に spring がありますが、過去の理由により、Spring は最初であり、XML DSL は時間内に戻るだけなので、過去の理由によるものです。実行時に Spring JAR は必要ありません。Camel 3.0 では、namespace の名前を汎用名に変更できます。

279.8.5.4. 適切なプレースホルダーの設定

以下は、クラスパスから myproperties.properties をロードするプロパティープレースホルダーを設定するための web.xml 設定のスニペットです。

```
  <!-- setup property placeholder to load properties from classpath -->
  <!-- we do this by setting the param-name with propertyPlaceholder. as prefix and then any
options such as location, cache etc -->
  <context-param>
    <param-name>propertyPlaceholder.location</param-name>
    <param-value>classpath:myproperties.properties</param-value>
  </context-param>
  <!-- for example to disable cache on properties component, you do -->
  <context-param>
    <param-name>propertyPlaceholder.cache</param-name>
    <param-value>false</param-value>
  </context-param>
```

279.8.5.5. JMX の設定

以下は、JMX の無効化など、JMX を設定するための web.xml 設定のスニペットです。

```
  <!-- configure JMX by using names that is prefixed with jmx. -->
  <!-- in this example we disable JMX -->
  <context-param>
    <param-name>jmx.disabled</param-name>
    <param-value>true</param-value>
  </context-param>
```

JNDI または Camel Registry ^^^^^^^^^^^^^^^^

このコンポーネントは、JNDI または Simple をレジストリーとして使用します。
これにより、JNDI で Beanおよびその他のサービスを検索したり、独自の Beanをバインドおよびバイ

ンド解除したりすることができます。

これは、org.apache.camel.component.servletlistener.CamelContextLifecycle を実装して Java コードから実行されます。

279.8.5.6. カスタム CamelContextLifecycle の使用

以下のコードでは、beforeStart および afterStop のコールバックを使用して Simple Registry にカスタム Bean を表示し、停止時にクリーンアップします。

次に、パラメーター名 "CamelContextLifecycle" を使用して、web.xml ファイルにこのクラスを登録する必要があります。この値は、org.apache.camel.component.servletlistener.CamelContextLifecycle インターフェースを実装するクラスを参照する FQN である必要があります。

```
<context-param>
  <param-name>CamelContextLifecycle</param-name>
  <param-value>org.apache.camel.component.servletlistener.MyLifecycle</param-value>
</context-param>
```

「my Bean」という名前を使用して HelloBean Bean を登録すると、以下のように Camel ルートでこの Bean を参照できます。

```
public class MyBeanRoute extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("seda:foo").routeId("foo")
            .to("bean:myBean")
            .to("mock:foo");
    }
}
```

重要： org.apache.camel.component.servletlistener.JndiCamelServletContextListener を使用する場合、CamelContextLifecycle も JndiRegistry を使用する必要があります。サーブレットが org.apache.camel.component.servletlistener.SimpleCamelServletContextListener の場合には、CamelContextLifecycle が SimpleRegistryを使用する必要があります。

279.8.6. 関連項目

- SERVLET

- *サーブレット Tomcat の例*

- *Servlet Tomcat No Spring Example*

第280章 SFTP コンポーネント

Camel バージョン 1.1 で利用可能

このコンポーネントは、FTP プロトコルおよび SFTP プロトコルを介してリモートファイルシステムへのアクセスを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ftp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

詳細は、FTP コンポーネントを参照してください。

280.1. URI オプション

以下のオプションは、FTPS コンポーネント専用です。

SFTP コンポーネントにはオプションがありません。

SFTP エンドポイントは、URI 構文を使用して設定されます。

```
sftp:host:port/directoryName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

280.1.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
host	FTP サーバーに **必要な** ホスト名		文字列

Name	説明	デフォルト	Type
port	FTP サーバーのポート		int
directoryName	起動ディレクトリー		文字列

280.1.2. クエリーパラメーター（111 パラメーター）：

Name	説明	デフォルト	Type
charset (common)	このオプションは、ファイルのエンコーディングを指定するために使用されます。これはコンシューマーで使用して、ファイルのエンコーディングを指定できます。これにより、Camel はファイルの内容にアクセスした場合にファイルの内容を読み込む必要があります。ファイルを書き込む場合も同様に、このオプションを使用して、ファイルを書き込む文字を指定することもできます。ファイルを書くときには、メッセージの内容をメモリーに読み込んで、データを設定済みの文字セットに変換しなければならない場合があります。そのため、大きなメッセージがある場合は使用しないでください。		文字列
切断 （共通）	使用後にリモートの FTP サーバーから適切な接続を解除するかどうか。切断すると、現在の FTP サーバーへの接続が切断されるだけです。停止するコンシューマーがある場合は、代わりにコンシューマー/ルートを停止する必要があります。	false	boolean
doneFileName (common)	プロデューサー： 指定された場合、元のファイルが書き込まれると、Camel は 2 番目に完了したファイルを書き込みます。完了ファイルは空になります。このオプションは、使用するファイル名を設定します。固定名を指定できます。または、動的プレースホルダーを使用することもできます。done ファイルは、常に元のファイルと同じフォルダーに書き込まれます。コンシューマー： 提供された場合、Camel は完了したファイルが存在する場合にのみファイルを消費します。このオプションは、使用するファイル名を設定します。固定名を指定できます。または、動的プレースホルダーを使用することもできます。完成したファイルは、常に元のファイルと同じフォルダーに期待されます。$file.name および $file.name.noext のみが動的プレースホルダーとしてサポートされます。		文字列

Name	説明	デフォルト	Type
fileName (common)	File Language などの式を使用して、ファイル名を動的に設定します。コンシューマーの場合は、ファイル名フィルターとして使用されます。プロデューサーの場合、書き込みするファイル名を評価するために使用されます。式が設定されている場合は、CamelFileName ヘッダーよりも優先されます。（注記： ヘッダー自体は式にすることもできます。式オプションは String タイプと Expression 型の両方をサポートします。式が String タイプである場合、これは常に File 言語を使用して評価されます。式が Expression タイプである場合、指定された式タイプが使用されます。たとえば、OGNL 式を使用できます。コンシューマーの場合は、これを使用してファイル名を絞り込むことができます。そのため、インスタンスは File Language 構文 mydata-$date:yyyyMMdd.txt を使用して現在のファイルを使用できます。プロデューサーは、既存の CamelFileName ヘッダーよりも優先される CamelOverruleFileName ヘッダーをサポートします。CamelOverruleFileName は 1 度だけ使用されるヘッダーで、CamelFileName を一時的に保存し、後で復元する必要があるため、より簡単に使用できます。		文字列
jschLoggingLevel (common)	JSCH アクティビティーロギングに使用するロギングレベル。JSCH はデフォルトで INFO レベルで詳細なため、しきい値はデフォルトで WARN になります。	WARN	LoggingLevel
セパレーター （共通）	使用するパス区切り文字を設定します。UNIX = Uses unix スタイルの path separator Windows = Uses windows style path separator Auto =(is default)Use existing path separator in file name	UNIX	PathSeparator
fastExistsCheck (common)	このオプションを true に設定すると、camel-ftp はリストファイルを直接使用してファイルが存在するかどうかを確認します。一部の FTP サーバーはファイルを直接一覧表示できない可能性があるため、オプションが false の場合、camel-ftp は古い方法でディレクトリーを一覧表示し、ファイルが存在するかどうかを確認します。また、このオプションは readLock=changed にも影響し、ファイル情報を更新する高速チェックを実行するかどうかを制御します。FTP サーバーに多くのファイルがある場合に、このプロセスを迅速化するために使用できます。	false	boolean

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
Delete （コンシューマー）	true の場合、ファイルは正常に処理された後に削除されます。	false	boolean
moveFailed (consumer)	Simple 言語に基づいて移動失敗式を設定します。たとえば、.error サブディレクトリーにファイルを移動するには、.error を使用します。注記： 失敗した場所にファイルを移動すると Camel はエラーを処理し、再度ファイルを取得しません。		文字列
noop （コンシューマー）	true の場合、ファイルは移動または削除されません。このオプションは、読み取り専用データまたは ETL タイプの要件に適しています。noop=true の場合、Camel はべき等=true も設定し、同じファイルを何度も消費しないようにします。	false	boolean
preMove (consumer)	処理前にファイル名を動的に設定するために使用される式（ファイル言語など）。たとえば、進行中のファイルを order ディレクトリーに移動するには、この値を order に設定します。		文字列
preSort (consumer)	pre-sort を有効にすると、コンシューマーはポーリング中にファイルとディレクトリー名をソートし、ファイルシステムから取得されたものになります。これは、ソートされた順序でファイルで操作する必要がある場合があります。pre-sort は、コンシューマーがフィルターをかけ、Camel が処理するファイルを受け入れる前に実行されます。このオプションは、無効を意味する default=false です。	false	boolean
再帰的 （コンシューマー）	ディレクトリーの場合は、すべてのサブディレクトリー内のファイルも検索します。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

Name	説明	デフォルト	Type
streamDownload (consumer)	ローカルの作業ディレクトリーを使用しない場合に使用するダウンロード方法を設定します。true に設定すると、リモートファイルは読み取り時にルートにストリーミングされます。false に設定すると、リモートファイルはルートに送信される前にメモリーに読み込まれます。	false	boolean
directoryMustExist (consumer)	startingDirectoryMustExist と同様ですが、再帰的なサブディレクトリーのポーリング時に適用されます。	false	boolean
download (consumer)	FTP コンシューマーがファイルをダウンロードするかどうか。このオプションを false に設定すると、メッセージボディーは null になりますが、コンシューマーはファイル名、ファイルサイズなどのファイルの詳細が含まれる Camel Exchange のトリガーになります。ファイルはダウンロードされないだけです。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
ignoreFileNotFoundOrPermissionError (consumer)	いつ無視するか（ディレクトリー内のファイルを一覧表示しようとするか、またはファイルをダウンロードする際に）無視するかどうか。これは存在しないか、パーミッションエラーが原因です。デフォルトでは、ディレクトリーまたはファイルが存在しないか、パーミッションが不十分な場合に、例外が発生します。このオプションを true に設定すると、代わりにこれを無視することができます。	false	boolean
inProgressRepository (consumer)	プラグ可能な in-progress リポジトリー org.apache.camel.spi.IdempotentRepository。進行中のリポジトリーは、現在使用中の進行中のファイルを考慮するために使用されます。デフォルトでは、メモリーベースのリポジトリーが使用されます。		String>

Name	説明	デフォルト	Type
localWorkDirectory (consumer)	使用する場合は、ローカルの作業ディレクトリーを使用してリモートファイルの内容を直接ローカルファイルに保存し、コンテンツをメモリーにロードしないようにできます。これは、非常に大きなリモートファイルを使用しているため、メモリーを節約できることに有益です。		文字列
onCompletionException Handler (consumer)	カスタムの org.apache.camel.spi.ExceptionHandler を使用して、コンシューマーがコミットまたはロールバックを行う完了プロセス時に発生する例外を処理します。デフォルトの実装は、WARN レベルですべての例外をログに記録し、無視されます。		ExceptionHandler
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processStrategy (consumer)	プラグ可能な org.apache.camel.component.file.GenericFileProcessStrategy を使用すると、独自の readLock オプションまたは同様のものを実装できます。また、特別な対応ファイルが存在するなど、ファイルを使用する前に特別な条件が満たされなければならない場合にも使用できます。このオプションを設定すると、readLock オプションが適用されません。		GenericFileProcess s Strategy<T>
startingDirectoryMustExist (consumer)	開始ディレクトリーが存在するかどうか。autoCreate オプションはデフォルトで有効になっています。つまり、開始ディレクトリーが存在しない場合には、通常、起動ディレクトリーが作成されます。autoCreate を無効にして、これを有効にして、開始ディレクトリーが存在することを確認できます。ディレクトリーが存在しない場合は例外が発生します。	false	boolean
useList (consumer)	ファイルのダウンロード時に LIST コマンドの使用を許可するかどうか。デフォルトは true です。ユースケースによっては、特定のファイルをダウンロードし、LIST コマンドを使用できないことがあるため、このオプションを false に設定します。このオプションを使用すると、ダウンロードする特定のファイルには、ファイルサイズ、タイムスタンプ、パーミッションなどのメタデータ情報が含まれません。これらの情報は、LIST コマンドが使用されている場合のみ取得できるためです。	true	boolean

Name	説明	デフォルト	Type
fileExist (producer)	同じ名前のファイルがすでに存在する場合のアクション。デフォルトのオーバーライドは、既存のファイルを置き換えます。append: 既存のファイルにコンテンツを追加します。fail: GenericFileOperationException をスローし、既存のファイルがあることを示します。ignore: 問題を警告なしで無視し、既存のファイルは上書きしませんが、すべてが問題であることを前提としています。move - オプションでは、moveExisting オプションも設定する必要があります。eagerDeleteTargetFile オプションを使用して、ファイルの移動や既存のファイルがすでに存在する場合に実行内容を制御できます。それ以外の場合は、移動操作が失敗します。Move オプションは、ターゲットファイルを書き込む前に既存のファイルを移動します。TryRename は、tempFileName オプションが使用されている場合にのみ適用されます。これにより、存在チェックを実行せずに、一時的な名前から実際の名前へのファイルの名前変更を試すことができます。一部のファイルシステムや、特に FTP サーバーでは、このチェックの方が高速である場合があります。	オーバーライド	GenericFileExist
flatten (producer)	フラット化は、ファイル名パスをフラット化して先頭のパスを削除するために使用されるので、ファイル名だけになります。これにより、サブディレクトリーに再帰的に消費できますが、たとえば別のディレクトリーにファイルを書き込むと、それらのファイルは単一のディレクトリーに書き込まれます。これをプロデューサーで true に設定すると、CamelFileName ヘッダーのファイル名が任意の先頭パスに対して削除されます。	false	boolean
moveExisting (producer)	fileExist=Move が設定されている場合に使用するファイル名の計算に使用される式（ファイル言語など）。ファイルをバックアップサブディレクトリーに移動するには、バックアップを入力します。このオプションは、file:name、file:name.ext、file:name.noext、file:onlyname、file:onlyname.noext、file:ext、および file:parent の File Language トークンのみをサポートします。FTP コンポーネントは既存のファイルをベースとして、相対ディレクトリーにのみ移動できるため、file:parent は FTP コンポーネントではサポートされないことに注意してください。		文字列
tempFileName (producer)	tempPrefix オプションと同じですが、File 言語を使用する一時的なファイル名の命名をより細かく制御できます。		文字列

Name	説明	デフォルト	Type
tempPrefix (producer)	このオプションは、一時的な名前を使用してファイルを書き込むために使用されます。次に、書き込みが完了したら、その名前を変更します。書き込まれているファイルを特定し、進行中のファイルで読み取るコンシューマー（排他的読み取りロックを使用しない）も回避できます。大容量のファイルをアップロードする場合に FTP が使用することがよくあります。		文字列
allowNullBody (producer)	ファイルの書き込み中に null ボディーを許可するかどうかを指定するのに使用します。true に設定すると、空のファイルが作成され、false に設定され、null ボディーを file コンポーネントに送信しようとすると、'Cannot write null body to file.' の GenericFileWriteException がスローされます。fileExist オプションを 'Override' に設定すると、ファイルが切り捨てられます。追加する場合は、ファイルは変更されません。	false	boolean
chmod （プロデューサー）	保存したファイルで chmod を設定できます。例：chmod=640		文字列
disconnectOnBatchComplete (producer)	Batch アップロードの完了後にリモート FTP サーバーから右に切断するかどうか。disconnectOnBatchComplete は、現在の FTP サーバーの接続のみを切断します。	false	boolean
eagerDeleteTargetFile (producer)	既存のターゲットファイルを活発に削除するかどうか。このオプションは、fileExists=Override オプションおよび tempFileName オプションを使用している場合にのみ適用されます。このパラメーターを使用して、一時ファイルが書き込まれる前にターゲットファイルを削除する（false に設定します）ことができます。たとえば、大きなファイルを作成し、一時ファイルが書き込まれている間にターゲットファイルを存在させたいとします。これにより、一時ファイルの名前がターゲットファイル名に変更される直前に、ターゲットファイルが最後の時点までのみ削除されます。また、このオプションは、fileExist=Move が有効で、既存のファイルが存在すると、既存のファイルを削除するかどうかを制御することもできます。このオプションの copyAndDeleteOnRenameFails false を指定した場合、既存のファイルが存在する場合は、移動操作の前に既存のファイルが削除されます。	true	boolean

Name	説明	デフォルト	Type
keepLastModified (producer)	ソースファイルから最後に変更したタイムスタンプを保持します（存在する場合）。Exchange.FILE_LAST_MODIFIED ヘッダーを使用してタイムスタンプを見つけます。このヘッダーには java.util.Date またはタイムスタンプの long を含めることができます。タイムスタンプが存在し、オプションが有効な場合は、書き込まれたファイルにこのタイムスタンプを設定します。注記： このオプションは、ファイルプロデューサーにのみ適用されます。このオプションは、ftp プロデューサーでは使用できません。	false	boolean
sendNoop (producer)	FTP サーバーにファイルをアップロードする前に、事前書き込みチェックとして noop コマンドを送信するかどうか。これは、接続の検証がまだ有効であるため、デフォルトでは、ファイルのアップロードを警告なしで再接続できます。ただし、これにより問題が発生する場合は、このオプションをオフにすることができます。	true	boolean
autocreate(advanced)	ファイルのパス名に不足しているディレクトリーを自動的に作成します。ファイルコンシューマーの場合は、開始ディレクトリーを作成することを意味します。ファイルプロデューサーの場合、ファイルが書き込まれるディレクトリーを意味します。	true	boolean
bufferSize (advanced)	バイト単位で書き込みバッファーサイズを書き込みます。	131072	int
bulkRequests (advanced)	一度に処理できるリクエスト数を指定します。この値を増やすと、ファイル転送速度が若干向上しますが、メモリー使用量が増えます。		整数
圧縮 （詳細）	圧縮を使用するには、以下を行います。1 から 10 にレベルを指定します。重要： 圧縮サポートのために、必要な JSCH zlib JAR を手動でクラスパスに追加する必要があります。		int
connectTimeout (advanced)	FTPClient と JSCH の両方によって接続が確立されるまでの接続タイムアウトを設定します。	10000	int
maximumReconnectAttempts (advanced)	リモート FTP サーバーに接続しようとすると Camel が実行を試行する最大再接続試行を指定します。この動作を無効にするには 0 を使用します。		int

Name	説明	デフォルト	Type
proxy (advanced)	カスタムに設定された com.j Creation.jsch.Proxy を使用します。このプロキシーは、ターゲット SFTP ホストからメッセージを消費/送信するために使用されます。		Proxy
reconnectDelay (advanced)	Camel は再接続を試みる前に待機します。		Long
serverAliveCountMax (advanced)	sftp セッションの serverAliveCountMax を設定できます。	1	int
serverAliveInterval (advanced)	sftp セッションの serverAliveInterval を設定できます。		int
soTimeout (advanced)	FTPClient でのみ使用されるようにタイムアウトを設定します。	300000	int
stepwise (advanced)	ファイルのダウンロード時、またはファイルのダウンロード時、またはディレクトリーにファイルをアップロードする場合に、ディレクトリーをステップ的に変更するかどうかを設定します。たとえば、セキュリティー上の理由から FTP サーバーのディレクトリーを変更できない場合は、これを無効にすることができます。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
throwExceptionOnConnect Failed (advanced)	接続に失敗した場合（すべて）に例外がスローされるかどうか。デフォルトでは例外はスローされず、WARN がログに記録されます。これを使用して、例外がスローされ、org.apache.camel.spi.PollingConsumerPollStrategy ロールバックメソッドから発生した例外を処理できます。	false	boolean
タイムアウト （詳細）	FTPClient のみで使用された応答を待機するデータタイムアウトを設定します。	30000	int
antExclude (filter)	ant スタイルのフィルターの除外。antInclude と antExclude の両方を使用する場合は、antInclude よりも antExclude が優先されます。複数の除外はコンマ区切りの形式で指定できます。		文字列

Name	説明	デフォルト	Type
antFilterCaseSensitive (filter)	付与フィルターに大文字と小文字を区別するフラグを設定します。	true	boolean
antInclude (filter)	ant スタイルのフィルターが含まれます。複数の包含は、コンマ区切りの形式で指定できます。		文字列
eagerMaxMessagesPerPoll (filter)	maxMessagesPerPoll の制限が Eager かどうかを制御できます。Eager が Eager の場合は、制限がファイルのスキャン中になります。false の場合、すべてのファイルをスキャンし、並び替えを実行します。このオプションを false に設定すると、すべてのファイルを最初にソートし、ポーリングを制限できます。ソートを実行するには、すべてのファイルの詳細がメモリー内にあるため、メモリー使用量を高くする必要があることに注意してください。	true	boolean
exclude （フィルター）	ファイルを除外するために使用されます（ファイル名が正規表現パターンと一致する場合）。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW（）構文を使用してこれを設定する必要があります。エンドポイント URI の設定に関する詳細は、「エンドポイント URI の設定」を参照してください。		文字列
Filter （フィルター）	プラグ可能なフィルター（org.apache.camel.component.file.GenericFileFilter クラス）フィルターが accept（）メソッドで false を返すと、ファイルをスキップします。		GenericFileFilter<T>
filterDirectory (filter)	Simple 言語に基づいてディレクトリーをフィルタリングします。たとえば、現在の日付でフィルタリングするには、$date:now:yyMMdd などの単純な日付パターンを使用できます。		文字列
filterFile (filter)	Simple 言語に基づいてファイルをフィルタリングします。たとえば、ファイルサイズで絞り込むには、$file:size 5000 を使用できます。		文字列
idempotent (filter)	Idempotent Consumer EIP パターンを使用して、Camel がすでに処理されたファイルをスキップするオプション。デフォルトでは、1000 エントリーを保持するメモリーベースの LRUCache を使用します。noop=true の場合は、同じファイルを何度も使用することを回避するため、べき等性も有効になります。	false	ブール値

Name	説明	デフォルト	Type
idempotentKey (filter)	カスタムのべき等キーを使用するには、以下を行います。デフォルトでは、ファイルの絶対パスが使用されます。File 言語を使用すると、ファイル名とファイルサイズを使用できます (idempotentKey=$file:name-$file:size)。		文字列
idempotentRepository (filter)	何も指定されておらず、べき等性が true の場合、デフォルトでは MemoryMessageIdRepository を使用するプラグ可能なリポジトリー org.apache.camel.spi.IdempotentRepository。		String>
include （フィルター）	ファイルを含めるために使用されます。ファイル名が正規表現パターンと一致する場合（一致する場合は大文字と小文字が区別されます）。プラス記号などのシンボルを使用する場合は、エンドポイント URI としてこれを設定する場合は RAW（）構文を使用してこれを設定する必要があります。エンドポイント URI の設定に関する詳細は、「エンドポイント URI の設定」を参照してください。		文字列
maxDepth (filter)	ディレクトリーを再帰的に処理する際に通過する最大深度。	2147483647	int
maxMessagesPerPoll (filter)	ポーリングごとに収集する最大メッセージを定義します。デフォルトでは最大値は設定されません。1000 などの制限を設定して、数千のファイルがあるサーバーを起動すると回避できます。無効にするには、0 または negative の値を設定します。注記： このオプションが使用されている場合、File および FTP コンポーネントはソート前に制限されます。たとえば、100000 ファイルがあり、maxMessagesPerPoll=500 を使用する場合は、最初の 500 ファイルのみが選択され、その後にソートされます。eagerMaxMessagesPerPoll オプションを使用して、これを false に設定すると、最初にすべてのファイルをスキャンし、後でソートできます。		int
minDepth (filter)	ディレクトリーを再帰的に処理する際に処理を開始する最小深度。minDepth=1 を使用すると、ベースディレクトリーを意味します。minDepth=2 を使用すると、最初のサブディレクトリーを意味します。		int
Move （フィルター）	処理後にファイル名を動的に設定するために使用される式（Simple Language など）。ファイルを .done サブディレクトリーに移動するには、.done と入力します。		文字列

Name	説明	デフォルト	Type
exclusiveReadLockStrategy (lock)	プラグ可能な read-lock を org.apache.camel.component.file.GenericFileExclusiveReadLockStrategy 実装とする。		GenericFileExclusive ReadLockStrategy <T>
readLock (lock)	ファイルに read-lock が排他的な場合にのみファイルをポーリングするために、コンシューマーによって使用されます（ファイルが進行中の場合や書き込みされていません）。Camel はファイルロックが許可されるまで待機します。このオプションは、ストラテジーでビルドを提供します。none - No read lock is in use markerFile - Camel はマーカーファイル (fileName.camelLock)を作成し、そのロックを保持します。このオプションは、FTP コンポーネントの変更には使用できません。Changed は、ファイルの長さ/変更タイムスタンプを使用して、ファイルがすでにコピーされているかどうかを検出します。少なくとも 1 つのセクションを使用してこれを判断するため、このオプションはファイルを他のプロセスとしてすぐに消費できませんが、JDK IO API はファイルを別のプロセスで現在使用しているかどうかを判断することができないため、信頼性が高まります。readLockCheckInterval オプションを使用してチェック頻度を設定できます。fileLock - は java.nio.channels.FileLock 用です。このオプションは FTP コンポーネントでは使用できません。この方法は、ファイルシステムが分散ファイルロックに対応している場合を除き、マウント/共有経由でリモートファイルシステムにアクセスするときに回避する必要があります。名前変更は、読み取り専用で読み取りロックを取得できる場合に、テストとしてファイルの名前を変更するためのものです。べき等性 -（ファイルコンポーネントのみ）べき等性は、idempotentRepository を read-lock として使用することです。これにより、べき等リポジトリーの実装がサポートする場合、クラスタリングをサポートする読み取りロックを使用できます。idempotent-changed -（ファイルコンポーネントのみ）idempotent-changed はidempotentRepository を使用し、結合された read-lock として変更されます。これにより、べき等リポジトリー実装がこれをサポートする場合、クラスタリングをサポートする読み取りロックを使用できます。idempotent-rename -（ファイルコンポーネントのみ）idempotent-rename はidempotentRepository を使用し、結合された read-lock として名前を変更します。これにより、べき等リポジトリーの実装がサポートしている場合、クラスタリングをサポートする読み取りロックを使用できます。注記： 各種の読み取りロックは、クラスターモードですべて機能する訳ではありません。この場合、異なるノードの同時コンシューマーは共有ファイルシステムの同じ	none	文字列

Name	説明	デフォルト	Type
	ファイルに対して競合しています。atomic 操作を使用して空のマーカーファイルを作成しますが、クラスター内での動作が保証されません。fileLock の方が良好に機能しますが、ファイルシステムは分散ファイルロックなどに対応する必要があります。べき等リポジトリーが Hazelcast コンポーネントや Infinispan などのクラスタリングに対応している場合、べき等リポジトリーがクラスタリングをサポートする場合は、べき等読み取りロックを使用できます。		
readLockCheckInterval (lock)	読み取りロックでサポートされている場合は、readLock の間隔（ミリ秒単位）。この間隔は、読み取りロックの取得を試みる間にスリープするために使用されます。たとえば、変更した読み取りロックを使用する場合は、低速な書き込みのために、間隔を cater に設定できます。デフォルトの 1 秒。プロデューサーがファイルを書き込む速度が非常に遅い場合は、速度が長すぎる可能性があります。注記：FTP の場合、デフォルトの readLockCheckInterval は 5000 です。readLockTimeout の値は readLockCheckInterval よりも大きくなければなりませんが、thumb のルールは readLockCheckInterval よりも少なくとも 2 倍以上タイムアウトになるようにします。これは、読み取りロックプロセスでタイムアウトがヒットするまでにロックの取得を試行するために バブル時間が許可されるようにするために必要です。	1000	Long
readLockDeleteOrphanLock Files (lock)	Camel が適切にシャットダウンされない場合（JVM クラッシュなど）、ファイルシステム上に残っている可能性のある孤立した読み取りロックファイルを削除する際にマーカーファイルを使用したロックを削除するかどうか。このオプションを false に指定すると、孤立したロックファイルがあると、Camel がそのファイルを取得しようとしません。また、別のノードが同じ共有ディレクトリーからファイルを同時に読み取ることが原因として考えられます。	true	boolean
readLockLoggingLevel (lock)	読み取りロックを取得できなかったときに使用されるロギングレベル。デフォルトでは、WARN がログに記録されます。このレベルは OFF でロギングを持たずに変更できます。このオプションは、type の readLock（changed、fileLock、idempotent、idempotent-changed、idempotent-rename、rename）にのみ適用できます。	DEBUG	LoggingLevel

Name	説明	デフォルト	Type
readLockMarkerFile (lock)	変更、名前変更、または排他的読み取りロックタイプでマーカーファイルを使用するかどうか。デフォルトでは、マーカーファイルが使用されるだけでなく、同じファイルを選択する他のプロセスに対して保護されます。このオプションを false に設定すると、この動作をオフにすることができます。たとえば、Camel アプリケーションによってマーカーファイルをファイルシステムに書き込みます。	true	boolean
readLockMinAge (lock)	このオプションは、readLock=change にのみ適用されます。このオプションでは、読み取りロックの取得を試行する前にファイルに必要な最小期間を指定できます。たとえば、readLockMinAge=300s を使用して、ファイルを最後の 5 分前とする必要があります。これにより、指定された期間以上のファイルの取得を試みるため、変更した読み取りロックが高速化されます。	0	Long
readLockMinLength (lock)	このオプションは、readLock=changed にのみ適用されます。このオプションを使用すると、最小限のファイルの長さを設定できます。デフォルトでは、Camel はファイルにデータが含まれることを想定するため、デフォルト値は 1 です。このオプションをゼロに設定するには、ゼロ長のファイルを使用できます。	1	Long
readLockRemoveOnCommit (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションでは、ファイルの処理に成功し、コミットが行われるときに、べき等リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。デフォルトでは、このファイルは削除されません。これにより、競合状態が発生しないため、別のアクティブなノードがファイルを取得できなくなります。代わりに、べき等リポジトリーは、X 分後にファイル名のエントリーをエビクトするように設定するエビクションストラテジーをサポートする可能性があります。これにより、競合状態に関する問題がなくなります。	false	boolean
readLockRemoveOnRollback (lock)	このオプションは、readLock=idempotent にのみ適用されます。このオプションでは、ファイルの処理に失敗し、ロールバックが行われるときに、べき等リポジトリーからファイル名のエントリーを削除するかどうかを指定できます。このオプションが false の場合、ファイル名のエントリーが確認されます（ファイルがコミットしたかのように）。	true	boolean

Name	説明	デフォルト	Type
readLockTimeout (lock)	read-lock で対応している場合は、read-lock のオプションのタイムアウト（ミリ秒単位）。read-lock が許可されず、タイムアウトが発生すると、Camel はファイルをスキップします。次回のポーリング Camel はファイルを再度試行し、今回は読み取りロックが付与される可能性があります。0 以下の値を使用して、永久に指定します。現在、fileLock、change、および rename はタイムアウトに対応しています。注記： FTP の場合、デフォルトの readLockTimeout 値は 10000 ではなく 20000 です。readLockTimeout の値は readLockCheckInterval よりも大きくなければなりませんが、thumb のルールは readLockCheckInterval よりも少なくとも 2 倍以上タイムアウトになるようにします。これは、読み取りロックプロセスでタイムアウトがヒットするまでにロックの取得を試行するために バブル時間が許可されるようにするために必要です。	10000	Long
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long

Name	説明	デフォルト	Type
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
shuffle (sort)	ファイルの一覧をシャッフルする方法（ランダムな順序など）	false	boolean
sortBy (sort)	File 言語を使用した組み込みソート。入れ子のソートをサポートするため、ファイル名でソートされ、2 番目にグループの種類を変更した日付でソートできます。		文字列
sorter (sort)	java.util.Comparator クラスとしてのプラグ可能なソーター。		GenericFile<T>>
暗号 化（セキュリティー）	優先順に使用する暗号のコンマ区切りリストを設定します。可能な暗号名は JCraft JSCH で定義されます。例として、aes128-ctr,aes128-cbc,3des-ctr,3des-cbc,blowfish-cbc,aes192-cbc,aes256-cbc,aes256-cbc などがあります。指定されていない場合は、JSCH のデフォルト一覧が使用されます。		文字列
keyPair (security)	SFTP エンドポイントが公開鍵と秘密鍵の検証を実行できるように、公開鍵と秘密鍵のペアを設定します。		KeyPair

Name	説明	デフォルト	Type
knownHosts (security)	SFTP エンドポイントがホストキーの検証を実行できるように、バイトアレイの known_hosts を設定します。		byte[]
knownHostsFile (security)	SFTP エンドポイントが鍵の検証をホストできるように known_hosts ファイルを設定します。		文字列
knownHostsUri (security)	SFTP エンドポイントが鍵の検証をホストできるように known_hosts ファイル（デフォルトではクラスパスからロードされる）を設定します。		文字列
パスワード （セキュリティー）	ログインに使用するパスワード		文字列
preferredAuthentications (security)	SFTP エンドポイントが使用される優先認証を設定します。例としては、password、publickey などがあります。指定されていない場合は、JSCH のデフォルト一覧が使用されます。		文字列
privateKey (security)	SFTP エンドポイントが秘密鍵の検証を実行できるように、秘密鍵をバイトとして設定します。		byte[]
privateKeyFile (security)	SFTP エンドポイントが秘密鍵の検証を実行できるように、秘密鍵ファイルを設定します。		文字列
privateKeyPassphrase (security)	SFTP エンドポイントが秘密鍵の検証を実行できるように、秘密鍵ファイルのパスフレーズを設定します。		文字列
privateKeyUri (security)	SFTP エンドポイントが秘密鍵の検証を実行できるように、秘密鍵ファイル（デフォルトではクラスパスからロードされる）を設定します。		文字列
strictHostKeyChecking (security)	厳密なホストキーチェックを使用するかどうかを設定します。	いいえ	文字列
ユーザー名 （セキュリティー）	ログインに使用するユーザー名		文字列
useUserKnownHostsFile (security)	knownHostFile が明示的に設定されていない場合は、System.getProperty(user.home)/.ssh/known_hosts のホストファイルを使用します。	true	boolean

第281章 SHIRO セキュリティーコンポーネント

Camel 2.5 で利用可能

Camel の shiro-security コンポーネントは、Apache Shiro セキュリティープロジェクトをベースとするセキュリティーフォーカスのコンポーネントです。

Apache Shiro は、認証、承認、エンタープライズセッション管理、暗号化を適切に処理する強力かつ柔軟なオープンソースのセキュリティーフレームワークです。Apache Shiro プロジェクトの目的は、最も堅牢で包括的なアプリケーションセキュリティーフレームワークを提供することです。また、理解が非常に簡単で、非常に簡単です。

この camel shiro-security コンポーネントを使用すると、認証および承認サポートを camel ルートの異なるセグメントに適用できます。

Shiro セキュリティーは Camel ポリシーを使用してルートに適用されます。Camel の Policy は、インターセプターを Camel プロセッサーに適用するストラテジーパターンを使用します。これは、Camel ルートのセクション/セグメントにクロスピングに関する懸念（セキュリティー、トランザクションなど）を適用する機能を提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-shiro</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

281.1. SHIRO セキュリティーの基本

camel ルートで Shiro セキュリティーを使用するには、ShiroSecurityPolicy オブジェクトをセキュリティー設定の詳細（ユーザー、パスワード、ロールなど）でインスタンス化する必要があります。このオブジェクトは Camel ルートに適用する必要があります。この ShiroSecurityPolicy オブジェクトは Camel レジストリー（JNDI または ApplicationContextRegistry）に登録してから、Camel コンテキストの他のルートで使用できます。

設定の詳細は、Ini ファイル（プロパティーファイル）または Ini オブジェクトを使用して ShiroSecurityPolicy に提供されます。Ini ファイルは、以下に示すようにユーザー/ロールの詳細を含

む標準の Shiro 設定ファイルです。

```
[users]
# user 'ringo' with password 'starr' and the 'sec-level1' role
ringo = starr, sec-level1
george = harrison, sec-level2
john = lennon, sec-level3
paul = mccartney, sec-level3

[roles]
# 'sec-level3' role has all permissions, indicated by the
# wildcard '*'
sec-level3 = *

# The 'sec-level2' role can do anything with access of permission
# readonly (*) to help
sec-level2 = zone1:*

# The 'sec-level1' role can do anything with access of permission
# readonly
sec-level1 = zone1:readonly:*
```

281.2. SHIROSECURITYPOLICY オブジェクトのインスタンス化

ShiroSecurityPolicy オブジェクトは以下のようにインスタンス化されます。

```
    private final String iniResourcePath = "classpath:shiro.ini";
    private final byte[] passPhrase = {
       (byte) 0x08, (byte) 0x09, (byte) 0x0A, (byte) 0x0B,
       (byte) 0x0C, (byte) 0x0D, (byte) 0x0E, (byte) 0x0F,
       (byte) 0x10, (byte) 0x11, (byte) 0x12, (byte) 0x13,
       (byte) 0x14, (byte) 0x15, (byte) 0x16, (byte) 0x17};
    List<permission> permissionsList = new ArrayList<permission>();
    Permission permission = new WildcardPermission("zone1:readwrite:*");
    permissionsList.add(permission);

    final ShiroSecurityPolicy securityPolicy =
       new ShiroSecurityPolicy(iniResourcePath, passPhrase, true, permissionsList);
```

281.3. SHIROSECURITYPOLICY オプション

Name	デフォルト値	型	説明
iniResourcePath または ini	**none**	リソース文字列または Ini オブジェクト	iniResourcePath または Ini オブジェクトのインスタンスに必要なリソース文字列は、セキュリティーポリシーに渡す必要があります。リソースは、それぞれ「file:、classpath:、または url:」のプレフィックスが付けられたファイルシステム、クラスパス、または URL から取得できます。例：「classpath:shiro.ini」
passPhrase	**AES 128 ベースのキー**	byte[]	メッセージエクスチェンジとともに送信される ShiroSecurityToken を復号化する passPhrase
alwaysReauthenticate	**true**	boolean	個別の要求で再認証するように設定。false に設定すると、同じユーザーの要求のみが認証され、認証されるため、ユーザーは認証されロックされます。
permissionsList	**none**	List<Permission>	認証されたユーザーが追加のアクションを実行するために必要なパーミッションの一覧。これはルートでさらに継続します。パーミッションリストまたはロール一覧（以下を参照）が ShiroSecurityPolicy オブジェクトに提供されている場合、承認は必要ではないと見なされます。デフォルトでは、リスト内のパーミッションオブジェクトのいずれかが適用されると承認が付与されることに注意してください。
rolesList	**none**	list<String>	**Camel 2.13:** 認証されたユーザーが追加のアクションの実行を承認するために必要なロールの一覧（ルートでさらに継続）ShiroSecurityPolicy オブジェクトにロールリストやパーミッションリストがない場合（上記を参照）、承認は必要ではないと見なされます。デフォルトでは、リスト内のいずれかのロールが適用されると承認が付与される点に注意してください。
cipherService	**AES**	org.apache.shiro.crypto.CipherService	Shiro には AES および Blowfish ベースの CipherServices が同梱されています。これらを使用するか、独自の暗号実装を渡すことができます。
base64	**false**	**boolean**	**Camel 2.12:** JMS などのヘッダーの転送を可能にするセキュリティートークンヘッダーに base64 エンコーディングを使用するには、以下を行います。このオプションは **ShiroSecurityTokenInjector** でも設定する必要があります。

Name	デフォルト値	型	説明
allPermissionsRequired	**false**	**boolean**	Camel 2.13: デフォルトでは、permissionList パラメーターの Permission Objects が適用されると承認が付与されます。これは、すべてのパーミッションが満たされるようにするには true に設定します。
allRolesRequired	**false**	**boolean**	Camel 2.13: デフォルトでは、rolesList パラメーターのロールのいずれかが適用されると承認が付与されます。すべてのロールを満たすには、このパラメーターを true に設定します。

281.4. CAMEL ルートでの SHIRO 認証の適用

ShiroSecurityPolicy で、Message Header に暗号化された SecurityToken を含む受信メッセージエクスチェンジをテストし、許可し、さらに適切な認証に従います。SecurityToken オブジェクトには、ユーザーが有効なユーザーである場所を決定するために使用される Username/Password の詳細が含まれます。

```
protected RouteBuilder createRouteBuilder() throws Exception {
    final ShiroSecurityPolicy securityPolicy =
        new ShiroSecurityPolicy("classpath:shiro.ini", passPhrase);

    return new RouteBuilder() {
        public void configure() {
            onException(UnknownAccountException.class).
                to("mock:authenticationException");
            onException(IncorrectCredentialsException.class).
                to("mock:authenticationException");
            onException(LockedAccountException.class).
                to("mock:authenticationException");
            onException(AuthenticationException.class).
                to("mock:authenticationException");

            from("direct:secureEndpoint").
                to("log:incoming payload").
                policy(securityPolicy).
                to("mock:success");
        }
    };
}
```

281.5. CAMEL ルートでの SHIRO 承認の適用

Permissions List を ShiroSecurityPolicy に関連付けると、承認を camel ルートに適用できます。Permissions List は、ユーザーがルートセグメントの実行を続行するために必要なパーミッションを指定します。ユーザーに適切なパーミッションが設定されていない場合、リクエストに対する承認は追加で続行されません。

```
protected RouteBuilder createRouteBuilder() throws Exception {
    final ShiroSecurityPolicy securityPolicy =
        new ShiroSecurityPolicy("./src/test/resources/securityconfig.ini", passPhrase);

    return new RouteBuilder() {
        public void configure() {
            onException(UnknownAccountException.class).
                to("mock:authenticationException");
            onException(IncorrectCredentialsException.class).
                to("mock:authenticationException");
            onException(LockedAccountException.class).
                to("mock:authenticationException");
            onException(AuthenticationException.class).
                to("mock:authenticationException");

            from("direct:secureEndpoint").
                to("log:incoming payload").
                policy(securityPolicy).
                to("mock:success");
        }
    };
}
```

281.6. SHIROSECURITYTOKEN を作成してメッセージ EXCHANGE にインジェクトする

ShiroSecurityToken オブジェクトを作成し、ShiroSecurityTokenInjector と呼ばれる Shiro プロセッサーを使用して Message Exchange にインジェクトできます。クライアントの ShiroSecurityTokenInjector を使用して ShiroSecurityToken を注入する例は次のとおりです。

```
ShiroSecurityToken shiroSecurityToken = new ShiroSecurityToken("ringo", "starr");
ShiroSecurityTokenInjector shiroSecurityTokenInjector =
    new ShiroSecurityTokenInjector(shiroSecurityToken, passPhrase);

from("direct:client").
    process(shiroSecurityTokenInjector).
    to("direct:secureEndpoint");
```

281.7. SHIROSECURITYPOLICY によってセキュア化されるルートへのメッセージの送信

セキュリティーポリシーの適用先の camel ルートとともに送信されるメッセージおよびメッセージエクスチェンジは、Exchange ヘッダーの SecurityToken を反映する必要があります。SecurityToken は、Username および Password を保持する暗号化オブジェクトです。

SecurityToken はデフォルトで AES 128 ビットセキュリティーを使用して暗号化され、任意の暗号に変更できます。

以下は、Camel の ProducerTemplate と SecurityToken を使用してリクエストを送信する方法の例を示しています。

```
@Test
public void testSuccessfulShiroAuthenticationWithNoAuthorization() throws Exception {
    //Incorrect password
    ShiroSecurityToken shiroSecurityToken = new ShiroSecurityToken("ringo", "stirr");

    // TestShiroSecurityTokenInjector extends ShiroSecurityTokenInjector
    TestShiroSecurityTokenInjector shiroSecurityTokenInjector =
        new TestShiroSecurityTokenInjector(shiroSecurityToken, passPhrase);

    successEndpoint.expectedMessageCount(1);
    failureEndpoint.expectedMessageCount(0);

    template.send("direct:secureEndpoint", shiroSecurityTokenInjector);

    successEndpoint.assertIsSatisfied();
    failureEndpoint.assertIsSatisfied();
}
```

281.8. SHIROSECURITYPOLICY で保護されたルートへのメッセージの送信（CAMEL 2.12 以降）

Camel 2.12 以降では、以下の 2 つの方法でサブジェクトを提供できるため、より容易になりました。

281.8.1. Using ShiroSecurityToken

ユーザー名とパスワードが含まれる org.apache.camel.component.shiro.security.Shiro.security.Shiro.security.ShiroSecurityToken キーのヘッダー ShiroSecurityConstants. SHIRO_SECURITY_TOKEN を使用して、Camel ルートにメッセージを送信できます。以下に例を示します。

```
    ShiroSecurityToken shiroSecurityToken = new ShiroSecurityToken("ringo", "starr");

    template.sendBodyAndHeader("direct:secureEndpoint", "Beatle Mania",
ShiroSecurityConstants.SHIRO_SECURITY_TOKEN, shiroSecurityToken);
```

また、以下のように 2 つの異なるヘッダーでユーザー名とパスワードを指定することもできます。

```
    Map<String, Object> headers = new HashMap<String, Object>();
    headers.put(ShiroSecurityConstants.SHIRO_SECURITY_USERNAME, "ringo");
```

```
headers.put(ShiroSecurityConstants.SHIRO_SECURITY_PASSWORD, "starr");
template.sendBodyAndHeaders("direct:secureEndpoint", "Beatle Mania", headers);
```

ユーザー名とパスワードのヘッダーを使用すると、Camel ルートの ShiroSecurityPolicy はトークンを持つ ShiroSecurityConstants.SHIRO_SECURITY_TOKEN キーを持つ単一のヘッダーに自動的に変換されます。次に、トークンは ShiroSecurityToken インスタンスまたは base64 表現のいずれかを String として指定します（後者は base64=true を設定した場合）。

第282章 SIMPLE 言語

Camel バージョン 1.1 で利用可能

Simple Expression Language は、作成時に非常にシンプルな言語でしたが、より強力になりました。これは主に、XPath に新しい依存関係や知識を必要とせずに式と述語を評価するための非常に小さい言語であることが意図されているため、camel-core でのテストに適しています。この概念は、Camel ルート内に式ベースのスクリプトを必要とする場合、一般的なユースケースの 95% に対応しています。

ただし、より複雑なユースケースでは、以下を含むより表現的で強力な言語を選択することが推奨されます。

- SpEL
- MVEL
- Groovy
- JavaScript
- OGNL
- サポートされる スクリプト言語の 1 つ

Simple 言語は、式に定数リテラルが含まれる複雑な式に ${body} プレースホルダーを使用します。式自体がトークン自体である場合、$\{ } プレースホルダーは省略できます。

ヒント

Camel 2.5 以降では、$simple{ } をプレースホルダーとして使用する別の構文を使用することもできます。これは、たとえば Spring プロパティープレースホルダーを Camel と併用する場合のクラッシュを回避するために使用できます。

282.1. CAMEL 2.9 以降の SIMPLE 言語の変更

Simple 言語は Camel 2.9 以降では改善され、正確なエラーメッセージをインデックス化できるより優れた構文パーサーが使用されるため、何が適切で問題なのか、問題がある場所を正確に把握することができます。たとえば、演算子のいずれかにタイプミスがあった場合、これまでパーサーはこれを検出できず、評価が true になりました。構文には後方互換性がないため、いくつかの変更があります。Simple 言語を述語として使用する場合は、リテラルテキストを一重引用符または二重引用符で囲む 必要 があります。例： "${body} == 'Camel'"リテラルの一重引用符がどのように存在するかに注目してください。メッセージのボディーとヘッダーを参照するために "body" および "header.foo" を使用する古いスタイルは @deprecated です。組み込み機能には常に $\{ } トークンを使用することが推奨されます。範囲演算子では、範囲を一重引用符で指定する必要があり、'30000..39999' の間の " "${header.zip} が表示されます。

message: "body"、または "in.body" または "${body}" のボディーを取得するには、次のコマンドを実行します。

複雑な式は、"Hello ${in.header.name} how are you?" などの $\{ } プレースホルダーを使用する必要があります。

同じ式 「Hello ${in.header.name}」には複数の関数を持つことができます。これは ${in.header.me} talking" です。
しかし、Camel 2.8.x 以前では、関数をネストでき ません （つまり、既存の $\{ } プレースホルダーを使用することはできません）。
Camel 2.9 以降では、機能をネスト化できます。

282.2. SIMPLE 言語オプション

Simple 言語は、以下に示す 2 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
resultType		文字列	結果タイプのクラス名を設定します（出力のタイプ）。
trim	true	ブール値	値をトリミングして先頭および末尾の空白と改行を削除するかどうか。

282.3. 変数

変数	型	説明
camelId	文字列	**Camel 2.10:** CamelContext 名
camelContext.**OGNL**	オブジェクト	**Camel 2.11:** Camel OGNL 式を使用して呼び出される CamelContext。
exchange	エクスチェンジ	**Camel 2.16:** the Exchange
Exchanges.**OGNL**	オブジェクト	**Camel 2.16:** Camel OGNL 式を使用して呼び出されるエクスチェンジ。
exchangeId	文字列	**Camel 2.3:** エクスチェンジ ID
id	文字列	入力メッセージ ID
ボディー	オブジェクト	入力ボディー
in.body	オブジェクト	入力ボディー
本文。**OGNL**	オブジェクト	**Camel 2.3:** Camel OGNL 式を使用して呼び出される入力ボディー。
In.body.**OGNL**	オブジェクト	**Camel 2.3:** Camel OGNL 式を使用して呼び出される入力ボディー。
bodyAs(**type**)	Type	**Camel 2.3:** ボディーをクラス名で決定される指定の型に変換します。変換された本文は null にすることができます。
bodyAs(**type**).**OGNL**	オブジェクト	**Camel 2.18:** クラス名によって決定される指定のタイプにボディーを変換し、Camel OGNL 式を使用してメソッドを呼び出します。変換された本文は null にすることができます。
mandatoryBodyAs(**type**)	Type	**camel 2.5:** ボディーをクラス名で決定される指定の型に変換し、本文が null ではないことを想定します。

変数	型	説明
mandatoryBodyAs(**type**).**OGNL**	オブジェクト	**Camel 2.18:** クラス名によって決定される指定のタイプにボディーを変換し、Camel OGNL 式を使用してメソッドを呼び出します。
out.body	オブジェクト	出力ボディー
header.foo	オブジェクト	入力 foo ヘッダーを参照します。
header[foo]	オブジェクト	**Camel 2.9.2:** 入力 foo ヘッダーを参照します。
headers.foo	オブジェクト	入力 foo ヘッダーを参照します。
headers[foo]	オブジェクト	**Camel 2.9.2:** 入力 foo ヘッダーを参照します。
in.header.foo	オブジェクト	入力 foo ヘッダーを参照します。
in.header[foo]	オブジェクト	**Camel 2.9.2:** 入力 foo ヘッダーを参照します。
in.headers.foo	オブジェクト	入力 foo ヘッダーを参照します。
in.headers[foo]	オブジェクト	**Camel 2.9.2:** 入力 foo ヘッダーを参照します。
header.foo[bar]	オブジェクト	**Camel 2.3:** 入力 foo ヘッダーをマップとして考慮し、バーがキーとしてマップでルックアップを実行します。
in.header.foo[bar]	オブジェクト	**Camel 2.3:** 入力 foo ヘッダーをマップとして考慮し、バーがキーとしてマップでルックアップを実行します。

変数	型	説明
in.headers.foo[bar]	オブジェクト	**Camel 2.3:** 入力 foo ヘッダーをマップとして考慮し、バーがキーとしてマップでルックアップを実行します。
header.foo.**OGNL**	オブジェクト	**Camel 2.3:** 入力 foo ヘッダーを参照し、Camel OGNL 式を使用してその値を呼び出します。
In.header.foo.**OGNL**	オブジェクト	**Camel 2.3:** 入力 foo ヘッダーを参照し、Camel OGNL 式を使用してその値を呼び出します。
In.headers.foo.**OGNL**	オブジェクト	**Camel 2.3:** 入力 foo ヘッダーを参照し、Camel OGNL 式を使用してその値を呼び出します。
out.header.foo	オブジェクト	out ヘッダー foo を参照してください。
out.header[foo]	オブジェクト	**Camel 2.9.2:** out ヘッダー foo を参照してください。
out.headers.foo	オブジェクト	out ヘッダー foo を参照してください。
out.headers[foo]	オブジェクト	**Camel 2.9.2:** out ヘッダー foo を参照してください。
headerAs(**key**, **type**)	Type	**Camel 2.5:** ヘッダーをクラス名で決定される指定の型に変換します。
ヘッダー	マップ	**Camel 2.9:** 入力ヘッダーの参照
in.headers	マップ	**Camel 2.9:** 入力ヘッダーの参照
property.foo	オブジェクト	**非推奨：** エクスチェンジの foo プロパティーを参照します。

変数	型	説明
exchangeProperty.foo	オブジェクト	**Camel 2.15:** エクスチェンジの foo プロパティーを参照します。
property[foo]	オブジェクト	**非推奨：** エクスチェンジの foo プロパティーを参照します。
exchangeProperty[foo]	オブジェクト	**Camel 2.15:** エクスチェンジの foo プロパティーを参照します。
Property.foo.**OGNL**	オブジェクト	**非推奨：** エクスチェンジの foo プロパティーを参照し、Camel OGNL 式を使用してその値を呼び出します。
exchangeProperty.foo.**OGNL**	オブジェクト	**Camel 2.15:** エクスチェンジの foo プロパティーを参照し、Camel OGNL 式を使用してその値を呼び出します。
sys.foo	文字列	システムプロパティーを参照してください。
sysenv.foo	文字列	**Camel 2.3:** システム環境を参照する
exception	オブジェクト	**Camel 2.4:** エクスチェンジの例外オブジェクトを参照します。エクスチェンジに例外が設定されていない場合は **null** になります。Exchange に該当する場合は、フォールバックして取得された例外(**Exchange.EXCEPTION_CAUGHT**)になります。
例外。**OGNL**	オブジェクト	**Camel 2.4:** Camel OGNL 式オブジェクトを使用して呼び出されるエクスチェンジ例外を参照します。
exception.message	文字列	エクスチェンジに例外が設定されていない場合は、エクスチェンジ **の** exception.message を参照してください。Exchange に該当する場合は、フォールバックして取得された例外(**Exchange.EXCEPTION_CAUGHT**)になります。
exception.stacktrace	文字列	**Camel 2.6.**エクスチェンジに例外が設定されていない場合は、エクスチェンジの exception . stracktrace を参照してください。Exchange に該当する場合は、フォールバックして取得された例外(**Exchange.EXCEPTION_CAUGHT**)になります。

変数	型	説明
date:_command_	Date	Date オブジェクトに評価します。サポートされるコマンド： 現在のタイムスタンプの **in.header.xxx** または **header.xxx** で、キー xxx を持つ IN ヘッダーの Date オブジェクトを使用する **よう** になりました。**out.header.xxx**: キー xxx と共に OUT ヘッダーの Date オブジェクトを使用します。**property.xxx**: キー xxx でエクスチェンジプロパティーの Date オブジェクトを使用します。**ファイル** の最後の変更されたタイムスタンプのファイル（ファイルコンシューマーで利用可能）。コマンドは、on - **24h、in.header.xxx+1h、または now+1h 30m-100 などのオフセットを受け入れます。**
date:_command:pattern_	文字列	**java.text.SimpleDataFormat** パターンを使用した日付形式。
date-with-timezone:_command:timezone:pattern_	文字列	**java.text.SimpleDataFormat** のタイムゾーンおよびパターンを使用した日付形式。
bean:_bean expression_	オブジェクト	Bean 言語を使用した Bean 式の呼び出し。区切り文字としてドットを使用する必要があるメソッド名を指定する。Bean コンポーネントが使用する ?method=methodname 構文もサポートしています。
properties:_locations:key_	文字列	**非推奨（代わりに properties-location を使用）Camel 2.3:** 指定のキーでプロパティーを検索します。**locations** オプションは任意です。詳細は「PropertyPlaceholder の使用」を参照してください。
properties-location:_http://locationskey[locations:key]_	文字列	**Camel 2.14.1:** 指定のキーでプロパティーを検索します。**locations** オプションは任意です。詳細は「PropertyPlaceholder の使用」を参照してください。
properties:key:default	文字列	**Camel 2.14.1**: 指定のキーでプロパティーを検索します。キーが存在しないか、または値がない場合は、任意のデフォルト値を指定できます。
routeId	文字列	**Camel 2.11:** Exchange がルーティングされている現在のルートの ID を返します。
threadName	文字列	**Camel 2.3:** 現在のスレッドの名前を返します。ロギングの目的で使用できます。

変数	型	説明
ref:xxx	オブジェクト	**Camel 2.6:** 指定の ID でレジストリーから Bean を検索する。
type:name.field	オブジェクト	**Camel 2.11:** FQN 名でタイプまたはフィールドを参照します。フィールドを参照するには、.FIELD_NAME を追加できます。たとえば、Exchange からの定数フィールドを **org.apache.camel.Exchange.FILE_NAME**として参照できます。
null	null	**Camel 2.12.3: null**を表します。
random_(value)_	整数	*Camel 2.16.0:* 0(included)と **値** （除外）の間でランダムな整数を返す
random_(min, max)_	整数	*Camel 2.16.0:* **min** (included)と **max** (excluded)の間でランダムな整数を返す
collate(group)	リスト	**Camel 2.17:** 照合関数はメッセージのボディーを繰り返し処理し、データを指定されたサイズのサブリストにグループ化します。Splitter EIP で使用して、メッセージボディーを分割し、分割されたサブメッセージを N サブリストのグループに分割することができます。このメソッドは、Groovy のコンケーターメソッドと同様に機能します。
skip(number)	Iterator	**Camel 2.19:** skip 関数はメッセージボディーをイテレートし、最初の項目数をスキップします。Splitter EIP と併用することで、メッセージボディーを分割し、最初の N 項目数をスキップすることができます。
messageHistory	文字列	**Camel 2.17:** 現在のエクスチェンジのルーティング方法のメッセージ履歴。これは、未処理の例外の場合にエラーハンドラーのログである route stack-trace メッセージ履歴と似ています。
messageHistory(false)	文字列	**Camel 2.17:** メッセージ履歴。エクスチェンジの詳細なし（ルート strack-trace のみが含まれます）。メッセージ自体から機密データをログに記録したくない場合は、これを使用できます。

282.4. OGNL 式のサポート

Camel 2.3 の時点で利用可能

INFO:Camel の OGNL サポート はメソッドのみを呼び出すためのものです。フィールドにアクセスできません。Camel 2.11.1 以降では、Java 配列の length フィールドにアクセスするための特別なサポートを追加しました。

Simple 言語および Bean 言語は、チェーンで Bean を呼び出す Camel OGNL 表記をサポートするようになりました。Message IN ボディーに getAddress（）メソッドを持つ POJO が含まれるとします。

その後、Camel OGNL 表記を使用してアドレスオブジェクトにアクセスできます。

```
simple("${body.address}")
simple("${body.address.street}")
simple("${body.address.zip}")
```

Camel はゲッターの短縮名を認識しますが、以下のような実際の名前を呼び出すか、または任意の名前を使用することができます。

```
simple("${body.address}")
simple("${body.getAddress.getStreet}")
simple("${body.address.getZip}")
simple("${body.doSomething}")
```

本文にアドレスがない場合は、null セーフ演算子(?.)を使用して NPE を回避することもできます。

```
simple("${body?.address?.street}")
```

Map または List タイプでインデックス化することもできます。そのため、以下を実行できます。

```
simple("${body[foo].name}")
```

ボディーが Map ベースで、foo が key として値を検索し、その値で getName メソッドを呼び出します。

キーにスペースがある場合は、キーを引用符で囲む 必要 があります（例： 'foo bar'）：

```
simple("${body['foo bar'].name}")
```

Map または List オブジェクトは、それらのキー名（ドットありまたはなし）を使用して直接アクセスできます。

```
simple("${body[foo]}")
simple("${body[this.is.foo]}")
```

キー foo に値がなければ、null セーフ演算子を使用して NPE を回避することができます。

```
simple("${body[foo]?.name}")
```

List タイプにアクセスすることもできます。たとえば、以下のようなアドレスから行を取得することもできます。

```
simple("${body.address.lines[0]}")
simple("${body.address.lines[1]}")
simple("${body.address.lines[2]}")
```

リストから 最後 の値を取得するのに使用できる特別な last キーワードがあります。

```
simple("${body.address.lines[last]}")
```

最後に 2 番目に数字を取り除くため、last-1 を使用して以下を指定 する ことができます。

```
simple("${body.address.lines[last-1]}")
```

3 番目の最後の内容は、以下のとおりです。

```
simple("${body.address.lines[last-2]}")
```

また、以下のようにリストで size メソッドを呼び出すことができます。

```
simple("${body.address.lines.size}")
```

Camel 2.11.1 以降では、Java アレイの length フィールドも追加しました。以下に例を示します。

```
String[] lines = new String[]{"foo", "bar", "cat"};
exchange.getIn().setBody(lines);

simple("There are ${body.length} lines")
```

はい、これを以下のように Operator サポートと組み合わせることができます。

```
simple("${body.address.zip} > 1000")
```

282.5. OPERATOR サポート

パーサーは、単一の Operator のみをサポートするよう制限されます。

これを有効にするには、左側の値を $\{ } で囲む必要があります。構文は以下のようになります。

```
${leftValue} OP rightValue
```

rightValue は、$\{ } で囲まれた ' ', null, a constant value、または別の式で囲まれた String リテラルになります。

重要

演算子には、スペースを入れる 必要が あります。

Camel は rightValue の型を leftValue タイプに変換するため、空にできます。文字列を数値に変換すると > 数値の比較を使用できます。

以下の演算子がサポートされます。

Operator	説明
==	EQUALS
=~	Camel 2.16: 等号（文字列値を比較すると大文字と小文字が無視）
>	は次の値よりも大きい:
>=	より大きいか等しい
<	は次の値よりも小さい:
<=	より小さいか等しい
!=	NOT EQUALS
contains	文字列ベースの値に含まれるかどうかのテスト

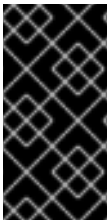
Operator	説明
not contains	文字列ベースの値に含まれていないかどうかのテスト
~~	文字列ベースの値の大文字と小文字を区別して含まれるかどうかをテストします。
regex	String 値として定義された特定の正規表現パターンに対するマッチング
not regex	String 値として定義された指定の正規表現パターンと一致しない
in	値のセットで一致する場合は、各要素をコンマで区切る必要があります。空の値を含める場合は、二重カンマ（例： ',bronze,silver,gold）を使用して定義する必要があります。これは、空の値を持つ 4 つの値のセットであり、3 つのマイタルになります。
not in	値のセットにない場合、各要素はコンマで区切る必要があります。空の値を含める場合は、二重カンマ（例： ',bronze,silver,gold）を使用して定義する必要があります。これは、空の値を持つ 4 つの値のセットであり、3 つのマイタルになります。
is	左側のタイプが instanceof the value である場合にマッチングします。
not is	左側のタイプが instanceof the value でない場合のマッチングの場合。
range	左側が number: **from...to で定義される値の範囲内にある場合はマッチングし** ます。Camel 2.9 以降では、範囲の値は一重引用符で囲む必要があります。
not range	左側が number: **from..to** で定義される値の範囲内にない場合に一致します。Camel 2.9 以降では、範囲の値は一重引用符で囲む必要があります。
は次の値で始まる	Camel 2.17.1、2.18: 左側の文字列が右側の文字列で始まるかどうかをテストするために使用します。
は次の値で終了する	Camel 2.17.1、2.18: 左側の文字列が右側の文字列で終わるかどうかをテストするために使用します。

さらに、以下の単項演算子を使用できます。

Operator	説明
++	**Camel 2.9:** 数字を 1 つずつインクリメントします。左側は関数である必要があります。それ以外の場合は、リテラルとして解析されます。
–	**Camel 2.9:** 数値を 1 つ減らす。左側は関数である必要があります。それ以外の場合は、リテラルとして解析されます。
\	**Camel 2.9.3 から 2.10.x へ** 値をエスケープするには、$ 記号を示す値（例： \$）をエスケープします。特殊： 改行の場合は \n、タブには \t、キャリッジリターンには \r を使用します。**注記：** File 言語 を使用したエスケープはサポートされて **いません**。注記 : Camel 2.11 以降ではエスケープ文字のサポートはなくなりましたが、以下の 3 つの特別なエスケープに置き換えられました。
\n	**Camel 2.11:** 改行文字を使用します。
\t	**Camel 2.11:** タブ文字を使用します。
\r	**Camel 2.11:** キャリッジリターン文字を使用します。
\}	**Camel 2.18:** } 文字をテキストとして使用するには、以下を行います。

また、次の論理演算子を使用して式をグループ化できます。

Operator	説明
および	代わりに && を使用してください。**論理 および 演算子 は、2 つの式をグループ化するために使用されます。**
または	**非推奨** のため、代わりに || を使用してください。論理 or 演算子を使用して、2 つの式をグループ化します。
&&	**Camel 2.9:** 論理 および 演算子 を使用して 2 つの式をグループ化します。
||	**Camel 2.9:** 論理 or 演算子を使用して、2 つの式をグループ化します。



重要

Camel 2.4 以前 における Operator また は Simple 言語式で 1 回 のみ使用 できます。Camel 2.5 以降では、これらの Operator を複数回使用できます。

AND の構文は以下のようになります。

```
${leftValue} OP rightValue and ${leftValue} OP rightValue
```

OR の構文は以下のようになります。

```
${leftValue} OP rightValue or ${leftValue} OP rightValue
```

例:

```
// exact equals match
simple("${in.header.foo} == 'foo'")

// ignore case when comparing, so if the header has value FOO this will match
simple("${in.header.foo} =~ 'foo'")

// here Camel will type convert '100' into the type of in.header.bar and if it is an Integer '100'
will also be converter to an Integer
simple("${in.header.bar} == '100'")

simple("${in.header.bar} == 100")

// 100 will be converter to the type of in.header.bar so we can do > comparison
simple("${in.header.bar} > 100")
```

282.5.1. 異なるタイプの比較

String や int などの異なるタイプと比較する場合は、少し注意して行ってください。Camel は左側からの型を 1 最優先として使用します。また、両方の値がそのタイプに基づいて比較できない場合には、右側のサイドタイプにフォールバックします。
これは、値を調整して特定のタイプを適用することができます。上記のバー値が String であるとします。次に、式を調節できます。

```
simple("100 < ${in.header.bar}")
```

これにより、int タイプが最優先で使用されるようになります。

Camel チームがバイナリー比較操作を改善し、String ベースよりも数値型を優先すると、将来的に変更される可能性があります。これはほとんどの場合、String タイプで、数字を比較する際に問題が発生します。

```
// testing for null
simple("${in.header.baz} == null")

// testing for not null
simple("${in.header.baz} != null")
```

正しい値が別の式であるもう少し高度な例

```
simple("${in.header.date} == ${date:now:yyyyMMdd}")

simple("${in.header.type} == ${bean:orderService?method=getOrderType}")
```

そしてサンプルには、タイトルに Camel という単語が含まれる場合のテスト

```
simple("${in.header.title} contains 'Camel'")
```

また、正規表現の例で、数字のヘッダーが 4 桁の値であるかどうかをテストします。

```
simple("${in.header.number} regex '\\d{4}'")
```

最後に、ヘッダーがリストの値のいずれかと等しい場合の例。各要素はコンマで区切り、スペースを入れないでください。
これは数字で機能します。Camel は各要素を左側の型に変換します。

```
simple("${in.header.type} in 'gold,silver'")
```

最後の 3 では、否定テストもサポートしていません。

```
simple("${in.header.type} not in 'gold,silver'")
```

また、タイプが特定のインスタンスであるかどうか（例： String）をテストすることができます。

```
simple("${in.header.type} is 'java.lang.String'")
```

以下のように記述できるように、すべての java.lang タイプに短縮名を追加しました。

```
simple("${in.header.type} is 'String'")
```

範囲もサポートされます。範囲の間隔は、数値と終了の両方が含まれます。たとえば、値が 100 から 199 の間であるかどうかをテストするには、以下を実行します。

```
simple("${in.header.number} range 100..199")
```

範囲にスペースなしの .. を使用することに注意してください。これは Groovy と同じ構文に基づいています。

Camel 2.9 以降では、範囲の値は一重引用符である必要があります。

```
simple("${in.header.number} range '100..199'")
```

282.5.2. Spring XML の使用

Spring XML にはさまざまなビルダーメソッドを持つ Java DSL と同じ機能がすべてないので、単純な演算子でテストするために他の言語を使用するべきではありません。これで、Simple 言語でこれを実行できます。以下の例では、ヘッダーがウィジェットの順序であるかどうかをテストします。

```
<from uri="seda:orders">
  <filter>
    <simple>${in.header.type} == 'widget'</simple>
    <to uri="bean:orderService?method=handleWidget"/>
  </filter>
</from>
```

282.6. の使用および / または

2 つの式がある場合は、and または or 演算子 と 組み合わせることができます。

ヒント

Camel 2.9 以降 の Camel 2.9 以降では、Camel 2.9 以降は && または || を使用します。

たとえば、以下のようになります。

```
simple("${in.header.title} contains 'Camel' and ${in.header.type'} == 'gold'")
```

当然ながら、また は もサポートされます。サンプルは以下のようになります。

```
simple("${in.header.title} contains 'Camel' or ${in.header.type'} == 'gold'")
```

注記： 現在 、 または Simple 言語式で 1 回 のみ使用できます。これは今後変更される可能性があります。
そのため、以下を行う ことはできません。

```
simple("${in.header.title} contains 'Camel' and ${in.header.type'} == 'gold' and
${in.header.number} range 100..200")
```

282.7. サンプル

以下の Spring XML の例では、ヘッダー値に基づいてフィルターを行います。

```
<from uri="seda:orders">
  <filter>
    <simple>${in.header.foo}</simple>
    <to uri="mock:fooOrders"/>
  </filter>
</from>
```

Simple 言語は、Message Filter パターンの上記の述語テストに使用できます。ここでは、in メッセージに foo ヘッダーがあるかどうかをテストします（キー foo を持つヘッダーが存在するかどうかをテストします）。式が true に評価されると、メッセージは mock:fooOrders エンドポイントにルーティングされます。そうでない場合、メッセージは破棄されます。

Java DSL と同じ例は次のとおりです。

```
from("seda:orders")
   .filter().simple("${in.header.foo}")
      .to("seda:fooOrders");
```

Simple 言語は、以下のような単純なテキスト連結にも使用できます。

```
from("direct:hello")
    .transform().simple("Hello ${in.header.user} how are you?")
    .to("mock:reply");
```

式で $\{ } プレースホルダーを使用して、Camel がこれを正しく解析できるようにする必要があります。

この例では、date コマンドを使用して現在の日付を出力します。

```
from("direct:hello")
    .transform().simple("The today is ${date:now:yyyyMMdd} and it is a great day.")
    .to("mock:reply");
```

また、以下の例では Bean 言語を呼び出して、返された文字列に含まれる Bean 上でメソッドを呼び出します。

```
from("direct:order")
    .transform().simple("OrderId: ${bean:orderIdGenerator}")
    .to("mock:reply");
```

orderIdGenerator は、レジストリーに登録されている Bean の ID に置き換えます。Spring を使用している場合は、Spring Bean id になります。

注文 ID ジェネレーター Bean で呼び出すメソッドを宣言したい場合は、以下のように .method 名の前に .method メソッドを呼び出します。ここで、generateId メソッドを呼び出します。

```
from("direct:order")
    .transform().simple("OrderId: ${bean:orderIdGenerator.generateId}")
    .to("mock:reply");
```

Bean コンポーネント自体に精通している ?method=methodname オプションを使用することもできます。

```
from("direct:order")
    .transform().simple("OrderId: ${bean:orderIdGenerator?method=generateId}")
    .to("mock:reply");
```

Camel 2.3 以降では、以下のようにボディーを特定の型に変換することもできます。

```
<transform>
  <simple>Hello ${bodyAs(String)} how are you?</simple>
</transform>
```

短縮表記が含まれるいくつかのタイプがあるため、java.lang. String の代わりに String を使用できます。これらは byte[]、String、Integer、Long です。その他のタイプはすべて、org.w3c.dom.Document など、FQN 名を使用する必要があります。

Camel 2.3 以降では、ヘッダー マップ から値を検索することもできます。

```
<transform>
  <simple>The gold value is ${header.type[gold]}</simple>
</transform>
```

上記のコードでは、名前 型 を持つヘッダーを検索し、java.util.Map として考慮し、キー gold でルックアップし、値を返します。ヘッダーが変換可能でない場合は、例外が発生します。名前 タイプのあるヘッダーが存在しない場合には、null を返します。

Camel 2.9 以降では、以下のような機能をネスト化できます。

```
<setHeader headerName="myHeader">
  <simple>${properties:${header.someKey}}</simple>
</setHeader>
```

282.8. 定数または列挙の参照

Camel 2.11 から利用可能

顧客向けの列挙があるとします。

また、Content Based Router では、Simple 言語を使用してこの列挙を参照することで、列挙するメッセージを確認することができます。

282.9. XML DSL での改行またはタブの使用

Camel 2.9.3 で利用可能

Camel 2.9.3 以降では、値をエスケープできるため、XML DSL で新しい行やタブを簡単に指定できます。

```
<transform>
  <simple>The following text\nis on a new line</simple>
</transform>
```

282.10. 先頭および末尾の空白処理

Camel 2.10.0 から利用可能

Camel 2.10.0 以降では、式のトリム属性を使用して、先頭および末尾の空白文字を削除するか、または保持するかを制御できます。デフォルト値は true で、空白文字を削除します。

```
<setBody>
  <simple trim="false">You get some trailing whitespace characters.    </simple>
</setBody>
```

282.11. 結果の型の設定

Camel 2.8 から利用可能

Simple 式に結果タイプを指定できるようになりました。これは、評価の結果が希望の型に変換されることを意味します。これは、ブール値や整数などのタイプを定義するのに最も使用できます。

たとえば、ヘッダーをブール型として設定するには、以下を行うことができます。

```
.setHeader("cool", simple("true", Boolean.class))
```

XML DSL の場合

```
<setHeader headerName="cool">
  <!-- use resultType to indicate that the type should be a java.lang.Boolean -->
  <simple resultType="java.lang.Boolean">true</simple>
</setHeader>
```

282.12. 関数開始トークンおよび終了トークンの変更

Camel 2.9.1 から利用可能

Java コードを使用して、関数の開始トークンおよび終了トークンを設定できます（setters changeFunctionStartToken と changeFunctionEndToken を SimpleLanguage を使用して使用します）。Spring XML より、以下のようにプロパティーに新しい変更されたトークンを使用して <bean> タグを定義できます。

```
<!-- configure Simple to use custom prefix/suffix tokens -->
<bean id="simple" class="org.apache.camel.language.simple.SimpleLanguage">
  <property name="functionStartToken" value="["/>
  <property name="functionEndToken" value="]"/>
</bean>
```

上記の例では、[ ] を変更したトークンとして使用します。

開始/終了トークンを変更することで、クラスパスの同じ camel-core を共有するすべての Camel アプリケーションでこれらを変更します。
たとえば、OSGi サーバーの場合、Web アプリケーションを WAR ファイルとしてのみ影響を与える多くのアプリケーションに影響する可能性があります。

282.13. 外部リソースからのスクリプトの読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、「classpath:」、"file:"、または "http: " などのリソースから Camel に読み込むことができます。
これは、「 resource:scheme:location」 構文を使用して行われます。たとえば、実行可能なクラスパスのファイルを参照します。

```
.setHeader("myHeader").simple("resource:classpath:mysimple.txt")
```

282.14. SPRING BEAN をエクスチェンジプロパティーに設定

Camel 2.6 で利用可能

以下のように Spring Bean をエクスチェンジプロパティーに設定できます。

```
<bean id="myBeanId" class="my.package.MyCustomClass" />
```

```
...
<route>
  ...
  <setProperty propertyName="monitoring.message">
    <simple>ref:myBeanId</simple>
  </setProperty>
  ...
</route>
```

282.15. 依存関係

Simple 言語は **camel-core** の一部です。

第283章 SIP コンポーネント

Camel バージョン 2.5 で利用可能

Camel の sip コンポーネントは Jain SIP 実装（JCP ライセンスで利用可能）をベースにした通信コンポーネントです。

SIP(Session Initiation Protocol)は IETF 定義のシグナルプロトコルで、インターネットプロトコル(IP)の音声やビデオ呼び出しなどのマルチメディア通信セッションを制御するために広く使用されます。SIP プロトコルは、基礎となるトランスポート層から独立するように設計されたアプリケーションレイヤープロトコルです。Transmission Control Protocol(TCP)、User Datagram Protocol(UDP)、または SCTP(Stream Control Transmission Protocol)で実行できます。

Jain SIP 実装は TCP および UDP のみをサポートします。

Camel SIP コンポーネントは、イベントの RFC3903 - Session Initiation Protocol(SIP)Extension で説明されているように、SIPPublish および Subscribe 機能 のみ をサポートします。

この Camel コンポーネントは、プロデューサーとコンシューマーエンドポイントの両方をサポートします。

Camel SIP Producers(Event Publishers)および SIP コンシューマー(Event Subscribers)は、SIP Presence Agent（ステートフルブローカーエンティティー）と呼ばれる中間エンティティーを使用してイベントと状態情報を相互に通信します。

SIP ベースの通信では、リスナーのある SIP Stack は SIP プロデューサーおよびコンシューマーの両方でインスタンス化される 必要 があります（localhost を使用している場合は別のポートを使用）。これは、通信中に SIP スタック間のハンドシェイクと確認応答をサポートするために必要です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sip</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

283.1. URI 形式

sip エンドポイントの URI スキームは以下のとおりです。

```
sip://johndoe@localhost:99999[?options]
sips://johndoe@localhost:99999/[?options]
```

このコンポーネントは、TCP と UDP の両方のプロデューサーおよびコンシューマーエンドポイントをサポートします。

URI にクエリーオプションを追加するには、?option=value&option=value&...

283.2. オプション

SIP コンポーネントは、SIP プロトコルを介して状態を伝播するために必要なカスタムのステートフルヘッダーを作成するための広範囲の設定オプションおよび機能を提供します。

SIP コンポーネントにはオプションがありません。

SIP エンドポイントは、URI 構文を使用して設定します。

```
sip:uri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

283.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
uri	接続先の SIP サーバーに **必要な** URI（ユーザー名とパスワードは john:secretmyserver:9999 などを含む）		URI

283.2.2. クエリーパラメーター（44 パラメーター）：

Name	説明	デフォルト	Type
cacheConnections (common)	接続作成のコストが削減されるように、SipStack が接続をキャッシュする。これは、接続が長時間実行される対話に使用される場合に便利です。	false	boolean
contentSubType (common)	contentSubType の設定は、有効な MimeSubType に設定できます。	plain	文字列
contentType (common)	contentType の設定は、有効な MimeType に設定できます。	text	文字列
eventHeaderName (common)	String ベースのイベントタイプの設定。		文字列
eventId (common)	String ベースのイベント ID の設定。レジストリーベースの FromHeader が指定されていない限り必須の設定		文字列
fromHost (common)	メッセージオリジンのホスト名。レジストリーベースの FromHeader が指定されていない限り必須の設定		文字列
fromPort (common)	メッセージオリジンのポート。レジストリーベースの FromHeader が指定されていない限り必須の設定		int
fromUser (common)	メッセージオリジンのユーザー名。レジストリーベースのカスタム FromHeader が指定されていない限り必須の設定です。		文字列
msgExpiration (common)	エンドポイントで受信したメッセージが有効とみなされる時間	3600	int
receiveTimeoutMillis (common)	レスポンスや承認を待つ時間を指定（別の SIP スタックから受信可能）	10000	Long
stackName (common)	SIP エンドポイントに関連付けられた SIP Stack インスタンスの名前。	NAME_NOT_SET	文字列
toHost (common)	メッセージレシーバーのホスト名。レジストリーベースの ToHeader が指定されていない限り必須の設定		文字列
toPort (common)	メッセージレシーバーのポート名。レジストリーベースの ToHeader が指定されていない限り必須の設定		int

Name	説明	デフォルト	Type
toUser (common)	メッセージレシーバーのユーザー名。レジストリーベースのカスタム ToHeader が指定されていない限り必須の設定です。		文字列
トランスポート（共通）	トランスポートプロトコルの選択設定。有効な選択肢は tcp または udp です。	tcp	文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
コンシューマー（コンシューマー）	この設定は、このエンドポイントに作成する必要があるヘッダーの種類（FromHeader,ToHeader など）を判断するために使用されます。	false	boolean
presenceAgent (consumer)	この設定は、優先順位エージェントとコンシューマーを区別するために使用されます。これは、SIP Camel コンポーネントが基本的なプレイエージェント（テスト目的でのみ）に同梱されるためです。コンシューマーはこのフラグを true に設定する必要があります。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
addressFactory (advanced)	カスタム AddressFactory を使用する		AddressFactory
callIdHeader (advanced)	呼び出しの詳細が含まれるカスタム Header オブジェクト。javax.sip.header.CallIdHeader タイプを実装する必要があります。		CallIdHeader

Name	説明	デフォルト	Type
contactHeader (advanced)	詳細な連絡先情報（メールアドレス、電話番号など）を含むオプションのカスタムヘッダーオブジェクト。javax.sip.header.ContactHeader タイプを実装する必要があります。		ContactHeader
contentTypeHeader (advanced)	メッセージコンテンツの詳細が含まれるカスタムヘッダーオブジェクト。javax.sip.header.ContentTypeHeader タイプを実装する必要があります。		ContentTypeHeader
eventHeader (advanced)	イベントの詳細が含まれるカスタムヘッダーオブジェクト。javax.sip.header.EventHeader タイプを実装する必要があります。		EventHeader
expiresHeader (advanced)	メッセージの有効期限の詳細が含まれるカスタムヘッダーオブジェクト。javax.sip.header.ExpiresHeader タイプを実装する必要があります。		ExpiresHeader
extensionHeader (advanced)	ユーザー/アプリケーション固有の詳細を含むカスタムヘッダーオブジェクト。javax.sip.header.ExtensionHeader タイプを実装する必要があります。		ExtensionHeader
fromHeader (advanced)	メッセージオリジン設定が含まれるカスタム Header オブジェクト。javax.sip.header.FromHeader タイプを実装する必要があります。		FromHeader
headerFactory (advanced)	カスタム HeaderFactory を使用する場合		HeaderFactory
listeningPoint (advanced)	カスタム ListeningPoint 実装を使用するには、以下を行います。		ListeningPoint
maxForwardsHeader (advanced)	最大プロキシー転送の詳細を含むカスタムヘッダーオブジェクト。このヘッダーは、可能な限り viaHeaders に制限を配置します。javax.sip.header.MaxForwardsHeader 型を実装する必要があります。		MaxForwardsHeader
maxMessageSize (advanced)	許可される最大メッセージサイズ（バイト単位）の設定。	1048576	int
messageFactory (advanced)	カスタム MessageFactory を使用する場合		MessageFactory

Name	説明	デフォルト	Type
sipFactory (advanced)	カスタム SipFactory を使用して使用する SipStack を作成するには、以下を実行します。		SipFactory
sipStack (advanced)	カスタム SipStack の使用		SipStack
sipUri (advanced)	カスタム SipURI を使用します。設定されていない場合、SipUri フォールバックで toUser toHost:toPort のオプションを使用します。		SipURI
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
toHeader (advanced)	メッセージレシーバー設定が含まれるカスタム Header オブジェクト。javax.sip.header.ToHeader タイプを実装する必要があります。		ToHeader
viaHeaders (advanced)	javax.sip.header.ViaHeader タイプのカスタムヘッダーオブジェクトのリスト。リクエスト転送のプロキシーアドレスが含まれる各 ViaHeader。（リクエストがリスナーに到達すると、このヘッダーは各プロキシーによって自動的に更新されることに注意してください。）		リスト
implementationDebugLogFile (logging)	ロギングに使用するクライアントデバッグログファイルの名前		文字列
implementationServerLogFile (logging)	ロギングに使用するサーバーログファイルの名前		文字列
implementationTraceLevel (logging)	トレーシングのログレベル	0	文字列
maxForwards (proxy)	プロキシー転送の最大数		int
useRouterForAllUris (proxy)	この設定は、要求がプロキシー経由で Presence Agent に送信される場合に使用されます。	false	boolean

283.3. SIP エンドポイントへの/からのメッセージの送信

283.3.1. Camel SIP パブリッシャーの作成

以下の例では、SIP Publisher が作成され、SIP イベントの公開を
ユーザー "agent@localhost:5152" に送信します。これは、SIP Publisher と Subscriber 間のブローカーとして動作する SIP Presence Agent のアドレスです。

- クライアントの SIP スタックの使用
- evtHdrName と呼ばれるレジストリーベースの eventHeader の使用
- evtId と呼ばれるレジストリーベースの eventId の使用
- リスナーが user2@localhost:3534 として設定された SIP スタックから。
- 公開されるイベントは EVENT_A です。
- REQUEST_METHOD と呼ばれる必須のヘッダーは Request.Publish に設定されています。これにより、エンドポイントをイベントパブリッシャーとして設定します。

```
producerTemplate.sendBodyAndHeader(
    "sip://agent@localhost:5152?
stackName=client&eventHeaderName=evtHdrName&eventId=evtid&fromUser=user2&fromHost=localhost&fromPort=3534",
    "EVENT_A",
    "REQUEST_METHOD",
    Request.PUBLISH);
```

283.3.2. Camel SIP サブスクライバーの作成

以下の例では、SIP Subscriber が作成され、
ユーザー "johndoe@localhost:5154" に送信される SIP イベントの公開を受け取ります。

- Subscriber という名前の SIP スタックの使用
- agent@localhost:5152 という名前の Presence Agent ユーザーを使用した登録
- evtHdrName と呼ばれるレジストリーベースの eventHeader の使用。evtHdrName に

は、「Event_A」に設定されたイベントが含まれます。

- evtId と呼ばれるレジストリーベースの eventId の使用

```
@Override
protected RouteBuilder createRouteBuilder() throws Exception {
   return new RouteBuilder() {
      @Override
      public void configure() throws Exception {
         // Create PresenceAgent
         from("sip://agent@localhost:5152?
stackName=PresenceAgent&presenceAgent=true&eventHeaderName=evtHdrName&eventId=e
vtid")
            .to("mock:neverland");

         // Create Sip Consumer(Event Subscriber)
         from("sip://johndoe@localhost:5154?
stackName=Subscriber&toUser=agent&toHost=localhost&toPort=5152&eventHeaderName=ev
tHdrName&eventId=evtid")
            .to("log:ReceivedEvent?level=DEBUG")
            .to("mock:notification");

      }
   };
}
```

Camel SIP コンポーネントには、テストおよびデモの目的にのみ使用するための Presence Agent も同梱されます。上記の Presence Agent をインスタンス化する例を紹介します。

Presence Agent はユーザー agent@localhost:5152 として設定されており、Publisher と Subscriber の両方と通信できることに注意してください。これには、Publisher と Subscriber とは別の SIP stackName があります。Camel コンシューマーとして設定されますが、実際にはルート内のメッセージをエンドポイント「mock:neverland」に送信しません。

第284章 単純な JMS バッチコンポーネント

Camel バージョン 2.16 から利用可能

SJMS Batch は、JMS キューからパフォーマンスの高いトランザクションバッチ消費のための特殊なコンポーネントです。コンシューマーのみのコンポーネントおよびアグリゲーターのハイブリッドとして考えることができます。

Camel の一般的なユースケースは、キューからメッセージを消費し、集約された状態を別のエンドポイントに送信する前に集約することです。処理を実行しているシステムが失敗した場合にデータが失われないようにするため、通常はキューからのトランザクション内で消費され、JDBC コンポーネントで見つかったものなどの永続 AggregationRepository に保存されている集約が集約されます。

Aggregator パターンの動作は、受信メッセージが集約される前に AggregationRepository からデータを取得し、後で結果を書き込む必要があります。実際、集約されたアーティファクトの数が増えると、読み取りおよび書き込みが徐々に遅くなります。この影響を示す任意の時間単位を使用した例は次のとおりです。

項目	読み取り時間	書き込み時間	合計時間
0	0	1	1
1	1	2	4
2	2	3	9
3	3	4	16
4	4	5	25
5	5	6	36
6	6	7	49
7	7	8	64
8	8	9	81
9	9	10	100

一方、SJMS Batch コンポーネントを使用した消費パフォーマンスはリニアです。各メッセージは、次のメッセージがフェッチされる前に AggregationStrategy を使用して消費され、集計されます。す

べての消費と集約が単一の JMS トランザクションで実行されるので、中間状態を永続化するために外部ストレージは必要ありません。上記の読み取りおよび書き込みコストを回避します。実際には、これにより、スループットが増加する複数の順序が高くなります。

最初のメッセージ以降のサイズまたは期間のいずれかで完了条件が満たされると、集約された Exchange はルートに渡されます。このエクスチェンジの処理中に、例外がスローされた場合やシステムがシャットダウンすると、元の消費されたメッセージがすべてキューに戻されます（または、ブローカー設定に応じてデッドレターキューに置かれます）。

通常のアグリゲーターとは異なり、集約条件のファシリティーはありません。つまり、メッセージの複数のグループに消費をバッチ処理することはできません。消費されるすべてのメッセージは単一のバッチに集約されます。

複数の JMS コンシューマーサポートが利用できます。これにより、1 つのルートを使用して並行して消費でき、同時に JMS メッセージグループなどの機能を使用して関連メッセージをグループ化できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sjms</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

284.1. URI 形式

```
sjms:[queue:]destinationName[?options]
```

destinationName は JMS キューです。デフォルトでは、destinationName はキュー名として解釈されます。

```
sjms:FOO.BAR
```

必要に応じて、オプションの queue: プレフィックスを含めることができます。

```
sjms:queue:FOO.BAR
```

トピック消費は、そのコンテキスト内でバッチ消費を使用する利点がないため、サポートされていま

せん。通常、トピックメッセージは永続的ではなく、損失は許容されます。失敗したトランザクション内で消費されると、トピックメッセージはブローカーによって再配信されない可能性があります。単純な SJMS コンシューマーエンドポイントは、このシナリオの通常の非永続性アグリゲーターと併用できます。

284.2. コンポーネントのオプションおよび設定

Simple JMS Batch コンポーネントは、以下に示す 5 つのオプションをサポートします。

Name	説明	デフォルト	Type
connectionFactory (advanced)	SjmsBatchComponent を有効にするには ConnectionFactory が必要です。		ConnectionFactory
asyncStartListener (advanced)	ルートの開始時にコンシューマーメッセージリスナーを非同期に起動するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの起動中に Camel がブロックされます。このオプションを true に設定すると、ルートの起動が可能になりますが、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用すると、接続が確立されないと、WARN レベルで例外がログに記録され、コンシューマーはメッセージを受信できなくなります。次にルートを再起動して再試行できます。	false	boolean
recoveryInterval (advanced)	リカバリーを試行する間隔を指定します（例： 接続の更新時（ミリ秒単位）。デフォルトは 5000 ミリ秒で、5 秒です。	5000	int
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Simple JMS Batch エンドポイントは、URI 構文を使用して設定します。

```
sjms-batch:destinationName
```

-

以下の path パラメーターおよびクエリーパラメーターを使用します。

284.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
destinationName	宛先名が ***必要です***。キューのみがサポートされます。名前の前に 'queue:' を付けることができます。		文字列

284.2.2. クエリーパラメーター（23 パラメーター）：

Name	説明	デフォルト	Type
aggregationStrategy (consumer)	バッチされたすべてのメッセージを 1 つのメッセージにマージする、使用する集約ストラテジーが ***必要です***。		AggregationStrategy
allowNullBody (consumer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false で、メッセージボディーが null の場合、JMSException が発生します。	true	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
completionInterval (consumer)	ミリ秒単位の完了間隔。これにより、スケジュールされたレートでバッチが間隔ごとに完了します。タイムアウトが発生し、バッチにメッセージがない場合にバッチは空になることがあります。完了のタイムアウトと完了間隔を同時に使用することができないことに注意してください。設定できるのは 1 つのみです。	1000	int

Name	説明	デフォルト	Type
completionPredicate (consumer)	完了述語。これにより、述語が true と評価されるとバッチが完了します。述語は、文字列構文を使用して Simple 言語を使用して設定することもできます。オプション eagerCheckCompletion を true に設定して、述語が受信メッセージと一致するようにします。そうでない場合は集約されたメッセージと一致します。		文字列
completionSize (consumer)	バッチの完了時に消費されるメッセージの数	200	int
completionTimeout (consumer)	バッチの完了時に最初のメッセージの最初のメッセージから受信までのタイムアウト（ミリ秒単位）。タイムアウトが発生し、バッチにメッセージがない場合にバッチは空になることがあります。完了のタイムアウトと完了間隔を同時に使用することができないことに注意してください。設定できるのは 1 つのみです。	500	int
consumerCount (consumer)	消費する JMS セッションの数	1	int
eagerCheckCompletion (consumer)	Eager completion チェックを使用します。これは、completionPredicate が受信エクスチェンジを使用することを意味します。completionPredicate を確認せずに、集約されたエクスチェンジを使用します。	false	boolean
includeAllJMSXProperties (consumer)	JMS から Camel メッセージへのマッピング時に、すべての JMSXxxx プロパティーを含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合は、このオプションは適用されません。	false	boolean
mapJmsMessage (consumer)	Camel が受信した JMS メッセージを適切なペイロードタイプ（javax.jms.TextMessage など）に自動マッピングするかどうかを指定します。詳細は、以下のマッピングがどのように機能するかについてのセクションを参照してください。	true	boolean
pollDuration （コンシューマー）	メッセージのポーリングごとにミリ秒単位の時間。completionTimeOut は短く、バッチが開始されている場合に使用されます。	1000	int

Name	説明	デフォルト	Type
sendEmptyMessageWhenIdle (consumer)	完了のタイムアウトまたは間隔を使用する場合、タイムアウトが発生し、バッチにメッセージがない場合にバッチを空になることがあります。このオプションが true でバッチが空の場合、空のメッセージがバッチに追加され、空のメッセージがルーティングされます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
asyncStartListener (advanced)	ルートの開始時にコンシューマーメッセージリスナーを非同期に起動するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの起動中に Camel がブロックされます。このオプションを true に設定すると、ルートの起動が可能になりますが、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用すると、接続が確立されないと、WARN レベルで例外がログに記録され、コンシューマーはメッセージを受信できなくなります。次にルートを再起動して再試行できます。	false	boolean
headerFilterStrategy (advanced)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
jmsKeyFormatStrategy (advanced)	JMS 鍵をエンコードおよびデコードするためのプラグ可能なストラテジー。これにより、JMS 仕様に準拠します。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーは、ドットとハイフン（. および -）を安全にマーシャリングします。passthrough ストラテジーは、鍵をそのまま残します。JMS ヘッダーキーに不正な文字が含まれるかどうかは気にしない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy

Name	説明	デフォルト	Type
keepAliveDelay (advanced)	有効なセッションを再確立するまでの遅延（ミリ秒単位）。これが正の値である場合、SjmsBatchConsumer はメッセージの消費中に IllegalStateException が表示されると、新しいセッションを作成しようとします。この遅延値により、ログのスパムを防ぐことができます。これが負の値である場合（デフォルトは -1）、SjmsBatchConsumer は、常に処理する前に動作します。つまり、IllegalStateException が表示される場合、ルートはシャットダウンします。	-1	int
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信するときに Camel が javax.jms.Message オブジェクトの新規インスタンスを作成する際に呼び出される指定の MessageCreatedStrategy を使用します。		MessageCreatedStrategy
recoveryInterval (advanced)	リカバリーを試行する間隔を指定します（例： 接続の更新時（ミリ秒単位）。デフォルトは 5000 ミリ秒で、5 秒です。	5000	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
timeoutCheckerExecutor Service (advanced)	completionInterval オプションを使用している場合は、バックグラウンドスレッドが作成され、完了間隔がトリガーされます。すべてのコンシューマーに新しいスレッドを作成するのではなく、カスタムスレッドプールを提供する場合に使用します。		ScheduledExecutor Service

completionSize エンドポイント属性は completionTimeout と併用されます。最初の条件が満たされると、集約された Exchange がルートで出力されます。

第285章 シンプルな JMS コンポーネント

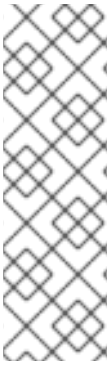
Camel バージョン 2.11 で利用可能

Simple JMS コンポーネント(SJMS)は、JMS クライアントの作成および設定に関してよく知られているベストプラクティスを使用する Camel で使用する JMS クライアントです。SJMS には、Camel によって明示的に作成された完全に新しい JMS クライアント API が含まれており、サードパーティーメッセージング実装が軽量で回復性を維持します。以下の機能が含まれています。

- 標準キューおよびトピックサポート(Durable & Non-Durable)
- InOnly & InOut MEP サポート
- 非同期プロデューサーおよびコンシューマー処理
- 内部 JMS トランザクションサポート

その他の主な機能は以下のとおりです。

- プラグ可能な接続リソース管理
- セッション、コンシューマー、プロデューサープーリング & キャッシュ管理
- バッチコンシューマーとプロデューサー
- トランザクションバッチコンシューマーとプロデューサー
- カスタマイズ可能なトランザクションコミットストラテジーのサポート（ローカル JMS トランザクションのみ）



注記

SJMS の S 理由

S は Simple および Standard および Springless を表します。また、camel-jms もすでに取得されています。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sjms</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

285.1. URI 形式

```
sjms:[queue:|topic:]destinationName[?options]
```

destinationName は JMS キューまたはトピック名です。デフォルトでは、destinationName はキュー名として解釈されます。たとえば、キューに接続するには、FOO.BAR は以下を使用します。

```
sjms:FOO.BAR
```

必要に応じて、オプションの queue: プレフィックスを含めることができます。

```
sjms:queue:FOO.BAR
```

トピックに接続するには、topic: プレフィックスを含める 必要があり ます。たとえば、トピック Stocks.Prices に接続するには、以下を使用します。

```
sjms:topic:Stocks.Prices
```

以下のフォーマットを使用して URI にクエリーオプションを追加します（? option=value&option=value&…

285.2. コンポーネントのオプションおよび設定

Simple JMS コンポーネントは、以下に示す 15 個のオプションをサポートします。

Name	説明	デフォルト	Type
connectionFactory (advanced)	SjmsComponent を有効にするには ConnectionFactory が必要です。ConnectionResource の一部として直接設定したり、設定したりできます。		ConnectionFactory
connectionResource (advanced)	ConnectionResource は、ConnectionFactory のカスタマイズおよびコンテナー制御を可能にするインターフェースです。詳細は、「プラグ可能な接続リソース管理」を参照してください。		ConnectionResource
connectionCount (common)	このコンポーネントで開始されるエンドポイントで利用可能な最大接続数	1	整数
jmsKeyFormatStrategy (advanced)	JMS 鍵をエンコードおよびデコードするためのプラグ可能なストラテジー。これにより、JMS 仕様に準拠します。Camel は、そのまま使える実装（デフォルト）を提供します。デフォルトのストラテジーは、ドットとハイフン（. および -）を安全にマーシャリングします。JMS ヘッダーキーに不正な文字が含まれるかどうかは気にしない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy
transactionCommit Strategy (transaction)	使用するコミットストラテジーの種類を設定します。Camel は、追加設定なしで、default と batch の 2 つの実装を提供します。		TransactionCommit Strategy
destinationCreation Strategy (advanced)	カスタムの DestinationCreationStrategy を使用します。		DestinationCreation Strategy
timedTaskManager (advanced)	カスタムの TimedTaskManager を使用する場合		TimedTaskManager
messageCreated Strategy (advanced)	Camel が JMS メッセージを送信するときに Camel が javax.jms.Message オブジェクトの新規インスタンスを作成する際に呼び出される指定の MessageCreatedStrategy を使用します。		MessageCreatedStrategy

Name	説明	デフォルト	Type
connectionTestOnBorrow (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用する場合は、プールから返される前に各 javax.jms.Connection をテストする（開始開始）する必要があります。	true	boolean
connectionUsername (security)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource の使用時に javax.jms.Connection を作成するときに使用するユーザー名。		文字列
connectionPassword (security)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource の使用時に javax.jms.Connection を作成する際に使用するパスワード。		文字列
connectionClientId (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource の使用時に javax.jms.Connection の作成時に使用するクライアント ID。		文字列
connectionMaxWait (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用する場合、プールが使い切られると、空き接続でブロックして待機する最大待機時間（ミリ秒単位）。	5000	Long
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Simple JMS エンドポイントは、URI 構文を使用して設定します。

```
sjms:destinationType:destinationName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

285.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
destinationType	使用する宛先の種類	queue	文字列
destinationName	**必要な** DestinationName は JMS キューまたはトピック名です。デフォルトでは、destinationName はキュー名として解釈されます。		文字列

285.2.2. クエリーパラメーター（34 パラメーター）：

Name	説明	デフォルト	Type
acknowledgementMode (common)	JMS 確認名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	SessionAcknowledgement Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumerCount (consumer)	このエンドポイントに使用されるコンシューマーリスナーの数を設定します。	1	int
durableSubscriptionId (consumer)	永続トピックに必要な永続サブスクリプション ID を設定します。		文字列
同期 （コンシューマー）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
messageSelector (consumer)	JMS メッセージセレクター構文を設定します。		文字列
namedReplyTo (producer)	InOut プロデューサーエンドポイントに使用される宛先名への返信を設定します。		文字列
永続 的（プロデューサー）	メッセージの永続性を有効または無効にするために使用されるフラグ。	true	boolean
producerCount (producer)	このエンドポイントに使用されるプロデューサーの数を設定します。	1	int
ttl (producer)	生成されたメッセージの Time To Live 値を調整するために使用されるフラグ。	-1	Long
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false で、メッセージボディーが null の場合、JMSException が発生します。	true	boolean
prefillPool (producer)	起動時にプロデューサー接続プールをプレフィルするか、または必要に応じて接続レイジーを作成するかどうか。	true	boolean
responseTimeOut (producer)	InOut 応答をタイムアウトするまでの待機時間を設定します。	5000	Long
asyncStartListener (advanced)	ルートの開始時にコンシューマーメッセージリスナーを非同期に起動するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの起動中に Camel がブロックされます。このオプションを true に設定すると、ルートの起動が可能になりますが、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用すると、接続が確立されないと、WARN レベルで例外がログに記録され、コンシューマーはメッセージを受信できなくなります。次にルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止する際に、コンシューマーメッセージリスナーを非同期的に停止するかどうか。	false	boolean
connectionCount (advanced)	このエンドポイントで利用可能な最大接続数		整数

Name	説明	デフォルト	Type
connectionFactory (advanced)	エンドポイントの connectionFactory を初期化します。これは、コンポーネントの connectionFactory よりも優先されます（存在する場合）。		ConnectionFactory
connectionResource (advanced)	エンドポイントの connectionResource を初期化します。これは、コンポーネントの connectionResource（ある場合）よりも優先されます。		ConnectionResource
destinationCreationStrategy (advanced)	カスタムの DestinationCreationStrategy を使用します。		DestinationCreation Strategy
exceptionListener (advanced)	基礎となる JMS 例外が通知される JMS 例外リスナーを指定します。		ExceptionListener
headerFilterStrategy (advanced)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
includeAllJMSXProperties (advanced)	JMS から Camel メッセージへのマッピング時に、すべての JMSXxxx プロパティーを含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合は、このオプションは適用されません。	false	boolean
jmsKeyFormatStrategy (advanced)	JMS 鍵をエンコードおよびデコードするためのプラグ可能なストラテジー。これにより、JMS 仕様に準拠します。Camel は、追加設定なしで、default と passthrough の 2 つの実装を提供します。デフォルトのストラテジーは、ドットとハイフン（. および -）を安全にマーシャリングします。passthrough ストラテジーは、鍵をそのまま残します。JMS ヘッダーキーに不正な文字が含まれるかどうかは気にしない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ（javax.jms.TextMessage など）に自動マッピングするかどうかを指定します。詳細は、以下のマッピングがどのように機能するかについてのセクションを参照してください。	true	boolean
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信するときに Camel が javax.jms.Message オブジェクトの新規インスタンスを作成する際に呼び出される指定の MessageCreatedStrategy を使用します。		MessageCreatedStrategy

Name	説明	デフォルト	Type
errorHandlerLoggingLevel (logging)	キャッチされない例外についてのデフォルトの errorHandler ロギングレベルの設定を可能にします。	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	デフォルトの errorHandler によってスタックトレースをログに記録するかどうかを制御できます。	true	boolean
トランザクション（トランザクション）	トランザクションモードを使用するかどうかを指定します。	false	boolean
transactionBatchCount (transaction)	トランザクションをコミットする前に処理するメッセージの数を設定する場合は、トランザクションがコミットされます。	-1	int
transactionBatchTimeout (transaction)	バッチトランザクションのタイムアウト（ミリ秒単位）を設定します。値は 1000 以上である必要があります。	5000	Long
transactionCommitStrategy (transaction)	コミットストラテジーを設定します。		TransactionCommit Strategy
sharedJMSSession (transaction)	JMS セッションを他の SJMS エンドポイントと共有するかどうかを指定します。ルートが複数の JMS プロバイダーにアクセスできる場合は、これをオフにします。複数の JMS プロバイダーに対してトランザクションが必要な場合は、jms コンポーネントを使用して XA トランザクションを活用します。	true	boolean

以下は、必要な ConnectionFactory プロバイダーで SjmsComponent を設定する方法の例になります。これは、デフォルトで単一の接続を作成し、コンポーネントの内部プーリング API を使用して格納し、スレッドを安全にセッション作成要求を処理できるようにします。

```
SjmsComponent component = new SjmsComponent();
component.setConnectionFactory(new ActiveMQConnectionFactory("tcp://localhost:61616"));
getContext().addComponent("sjms", component);
```

永続サブスクリプションのサポートに必要な SJMS コンポーネントについては、デフォルトの ConnectionFactoryResource インスタンスを上書きし、clientId プロパティーを設定します。

```
ConnectionFactoryResource connectionResource = new ConnectionFactoryResource();
connectionResource.setConnectionFactory(new
```

```
ActiveMQConnectionFactory("tcp://localhost:61616"));
connectionResource.setClientId("myclient-id");

SjmsComponent component = new SjmsComponent();
component.setConnectionResource(connectionResource);
component.setMaxConnections(1);
```

285.3. プロデューサーの使用

285.3.1. InOnly プロデューサー -（デフォルト）

InOnly プロデューサーは、SJMS プロデューサーエンドポイントのデフォルト動作です。

```
from("direct:start")
    .to("sjms:queue:bar");
```

285.3.2. InOut プロデューサー

InOut 動作を有効にするには、ExchangePattern 属性 を URI に追加します。デフォルトでは、コンシューマーごとに専用の TemporaryQueue を使用します。

```
from("direct:start")
    .to("sjms:queue:bar?exchangePattern=InOut");
```

名前付きのReplyTo を指定できますが、これによりモニターポイントを改善できます。

```
from("direct:start")
    .to("sjms:queue:bar?exchangePattern=InOut&namedReplyTo=my.reply.to.queue");
```

285.4. コンシューマーの使用

285.4.1. InOnly Consumer:（デフォルト）

InOnly xonsumer は、SJMS コンシューマーエンドポイントのデフォルト Exchange 動作です。

```
from("sjms:queue:bar")
    .to("mock:result");
```

285.4.2. InOut コンシューマー

InOut 動作を有効にするには、ExchangePattern 属性 を URI に追加します。

```
from("sjms:queue:in.out.test?exchangePattern=InOut")
    .transform(constant("Bye Camel"));
```

285.5. 高度な使用方法

285.5.1. プラグ可能な接続リソース管理

SJMS は、組み込み 接続 プールを使用して JMS 接続リソース管理を提供します。これにより、サードパーティーの API プーリングロジックに依存する必要がなくなります。ただし、J2EE や OSGi コンテナーによって提供される外部接続リソースマネージャーを使用する必要がある場合があります。この SJMS は、内部 SJMS 接続プール機能を上書きするために使用できるインターフェースを提供します。これは、ConnectionResource インターフェースを使用して実行できます。

Connection Resource は、必要に応じて接続をバンドルおよび返すメソッドが、接続プールを SJMS コンポーネントに提供するために使用されるコントラクトです。SJMS を外部接続プールマネージャーと統合する必要がある場合にユーザーを使用する必要があります。

標準の ConnectionFactory プロバイダーでは、SJMS をそのまま提供される ConnectionFactoryResource 実装を使用することが、このコンポーネントに対して最適化されるので、拡張することが推奨されます。

以下は、ActiveMQ PooledConnectionFactory でプラグイン可能な ConnectionResource を使用する例です。

```
public class AMQConnectionResource implements ConnectionResource {
    private PooledConnectionFactory pcf;

    public AMQConnectionResource(String connectString, int maxConnections) {
        super();
        pcf = new PooledConnectionFactory(connectString);
        pcf.setMaxConnections(maxConnections);
        pcf.start();
    }

    public void stop() {
        pcf.stop();
    }

    @Override
    public Connection borrowConnection() throws Exception {
        Connection answer = pcf.createConnection();
        answer.start();
```

```
        return answer;
    }

    @Override
    public Connection borrowConnection(long timeout) throws Exception {
        // SNIPPED...
    }

    @Override
    public void returnConnection(Connection connection) throws Exception {
        // Do nothing since there isn't a way to return a Connection
        // to the instance of PooledConnectionFactory
        log.info("Connection returned");
    }
}
```

次に、ConnectionResource を SjmsComponent に渡します。

```
CamelContext camelContext = new DefaultCamelContext();
AMQConnectionResource pool = new AMQConnectionResource("tcp://localhost:33333", 1);
SjmsComponent component = new SjmsComponent();
component.setConnectionResource(pool);
camelContext.addComponent("sjms", component);
```

使用方法の完全なサンプルを表示するには、ConnectionResourceIT を参照してください。

285.5.2. バッチメッセージのサポート

SjmsProducer は、List をカプセル化する Exchange を作成して、メッセージのコレクションの公開をサポートします。この SjmsProducer は、List の内容を繰り返し処理し、各メッセージを個別に公開します。

メッセージのバッチを生成する場合、各メッセージに固有のヘッダーを設定する必要がある場合は、SJMS BatchMessage クラスを使用できます。SjmsProducer が BatchMessage リストに遭遇すると、各 BatchMessage を繰り返し処理し、含まれるペイロードとヘッダーを公開します。

以下は BatchMessage クラスの使用例です。まず、BatchMessage のリストを作成します。

```
List<BatchMessage<String>> messages = new ArrayList<BatchMessage<String>>();
for (int i = 1; i <= messageCount; i++) {
    String body = "Hello World " + i;
    BatchMessage<String> message = new BatchMessage<String>(body, null);
    messages.add(message);
}
```

次に一覧を公開します。

```
template.sendBody("sjms:queue:batch.queue", messages);
```

285.5.3. カスタマイズ可能なトランザクションコミットストラテジー（ローカル JMS トランザクションのみ）

SJMS は、TransactionCommitStrategy インターフェースを使用してカスタムおよびプラグイン可能なトランザクションストラテジーを作成する手段を提供します。これにより、SessionTransactionSynchronization がセッションのコミットタイミングを決定するために使用する一意の状況のセットを定義できます。その使用例は BatchTransactionCommitStrategy で、次のセクションで詳しく説明されています。

285.5.4. トランザクションバッチコンシューマーとプロデューサー

SJMS コンポーネントは、Producer エンドポイントと Consumer エンドポイントの両方でローカル JMS トランザクションのバッチをサポートするように設計されています。ただし、それぞれでどのように処理されるかは非常に異なります。

SJMS コンシューマーエンドポイントは、関連付けられたセッションでコミットする前に X メッセージを処理する簡単な実装です。コンシューマーでバッチ処理トランザクションを有効にするには、最初に transacted パラメーターを true に設定し、transactionBatchCount を追加して 0 を超える値に設定します。たとえば、以下の設定では 10 個のメッセージごとにセッションをコミットします。

```
sjms:queue:transacted.batch.consumer?transacted=true&transactionBatchCount=10
```

コンシューマーエンドポイントでバッチの処理中に例外が発生すると、セッションロールバックが呼び出され、次の利用可能なコンシューマーにメッセージが再配信されます。また、カウンターは、関連付けられたセッションの BatchTransactionCommitStrategy に対して 0 にリセットされます。JMSRedelivered ヘッダーが true に設定されたメッセージを監視するように、バッチメッセージのプロセッサーにフックを配置することはユーザーの責任です。これは、ある時点でメッセージがロールバックされ、正常な処理が行われたことを示すインジケーターです。

トランザクションバッチコンシューマーは、セッション上でオープントランザクションをコミットする前に、メッセージ間でデフォルトの時間（5000 ミリ秒）を待機する内部タイマーのインスタンスも行います。デフォルト値の 5000ms（最低 1000ms）は、ほとんどのユースケースで十分ですが、さらにチューニングが必要な場合は、transactionBatchTimeout パラメーターを設定するだけで十分です。

```
sjms:queue:transacted.batch.consumer?
transacted=true&transactionBatchCount=10&transactionBatchTimeout=2000
```

コンテキストスイッチの量によりパフォーマンスに不必要な影響を及ぼすため、許可される最小値は 1000 ミリ秒です。

プロデューサーエンドポイントは処理されますが、異なる方法になります。各メッセージが宛先に配信された後にプロデューサーを使用すると、Exchange が閉じられ、そのメッセージへの参照がなくなりました。再配信ですべてのメッセージを利用可能にするには、BatchMessages を公開するプロデューサーエンドポイントでトランザクションを有効にするだけです。トランザクションは、バッチリスト内のすべてのメッセージが含まれるエクスチェンジの終了時にコミットされます。追加で設定する必要はありません。以下に例を示します。

```
List<BatchMessage<String>> messages = new ArrayList<BatchMessage<String>>();
for (int i = 1; i <= messageCount; i++) {
    String body = "Hello World " + i;
    BatchMessage<String> message = new BatchMessage<String>(body, null);
    messages.add(message);
}
```

トランザクションを有効にして List を公開します。

```
template.sendBody("sjms:queue:batch.queue?transacted=true", messages);
```

285.6. 追記

285.6.1. メッセージヘッダーの形式

SJMS コンポーネントは、Camel JMS コンポーネントで使用されるのと同じヘッダーフォーマットストラテジーを使用します。このプラグイン可能なストラテジーにより、ネットワーク上で送信されたメッセージが JMS Message 仕様に準拠するようになります。

exchange.in.header の場合、以下のルールがヘッダーキーに適用されます。

- JMS または JMSX で始まるキーが予約されています。
- exchange.in.headers キーはリテラルで、すべて有効な Java 識別子である必要があります（キー名のドットは使用しないでください）。
- Camel は、JMS メッセージの消費時にドットとハイフンと逆順を置き換えます。

- *Camel がメッセージを消費するときに* DOT *と逆の置換に置き換えられます。*

- *これは、Camel がメッセージを消費する際に* HYPHEN *と逆の置換に置き換えられます。*
  また、オプションの jmsKeyFormatStrategy も併せて参照してください。この場合、キーのフォーマットに独自のカスタムストラテジーを使用できます。

exchange.in.header では、ヘッダーの値に以下のルールが適用されます。

285.6.2. メッセージの内容

ネットワーク上でコンテンツを提供するには、配信されるメッセージのボディーが JMS メッセージ仕様に準拠することを確認する必要があります。そのため、生成されるすべてのものは、プリミティブまたはそれらのカウンターオブジェクト（ Integer、Long、Characterなど）のいずれかである必要があります。types、String、CharSequence、Date BigDecimal、および BigInteger はすべて toString（） 表現に変換されます。その他のタイプはすべてドロップされます。

285.6.3. クラスタリング

クラスター環境で InOut *と SJMS を使用する場合は、TemporaryQueue 宛先を使用するか、InOut プロデューサーエンドポイントごとに宛先への一意な名前付き応答を使用する必要があります。メッセージの相関は、ブローカーのメッセージセレクターではなく、エンドポイントによって処理されます。InOut Producer Endpoint は、Message JMSCorrelationID によってキャッシュされた Java Concurrency Exchangers を使用します。これにより、すべてのメッセージが宛先から消費されるため、関連するコンシューマーによって生成される順序で、ブローカーのオーバーヘッドが軽減されると同時にパフォーマンスが向上します。*

*現在、JMSCorrelationId を使用することが唯一の相関ストラテジーです。*InOut *コンシューマー はこのストラテジーを使用し、含まれる JMSReplyTo 宛先へのすべての応答メッセージで JMSCorrelationId もコピーされるようにします。*

285.7. トランザクションサポート

SJMS は現在内部 JMS トランザクションの使用のみをサポートします。Camel Transaction Processor または Java Transaction API(JTA)はサポートされません。

285.7.1. Springless Mean I can't use Spring?

全くない。以下は、Spring DSL を使用した SJMS コンポーネントの例です。

```
<route
   id="inout.named.reply.to.producer.route">
   <from
      uri="direct:invoke.named.reply.to.queue" />
   <to
      uri="sjms:queue:named.reply.to.queue?
namedReplyTo=my.response.queue&amp;amp;exchangePattern=InOut" />
</route>
```

Springless とは、Spring JMS API の依存関係から離れるという意味です。SJMS のパワーアップまで、新しい JMS クライアント API が開発されています。

第286章 SIMPLE JMS2 コンポーネント

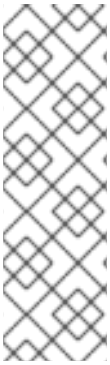
Camel バージョン 2.19 から利用可能

Simple JMS 2.0 Component(SJMS2)は、JMS クライアントの作成および設定に関してよく知られているベストプラクティスを使用する Camel で使用する JMS クライアントです。SJMS2 には、Camel 用に明示的に作成された完全に新しい JMS 2.0 クライアント API が含まれています。以下の機能が含まれています。

- *標準キューおよびトピックサポート(Durable & Non-Durable)*
- *InOnly & InOut MEP サポート*
- *非同期プロデューサーおよびコンシューマー処理*
- *内部 JMS トランザクションサポート*

その他の主な機能は以下のとおりです。

- *プラグ可能な接続リソース管理*
- *セッション、コンシューマー、プロデューサープーリング & キャッシュ管理*
- *バッチコンシューマーとプロデューサー*
- *トランザクションバッチコンシューマーとプロデューサー*
- *カスタマイズ可能なトランザクションコミットストラテジーのサポート（ローカル JMS トランザクションのみ）*



注記

SJMS の S 理由

S は Simple および Standard および Springless を表します。また、camel-jms もすでに取得されています。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sjms2</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

286.1. URI 形式

```
sjms2:[queue:|topic:]destinationName[?options]
```

destinationName は JMS キューまたはトピック名です。デフォルトでは、destinationName はキュー名として解釈されます。たとえば、キューに接続するには、FOO.BAR は以下を使用します。

```
sjms2:FOO.BAR
```

必要に応じて、オプションの queue: プレフィックスを含めることができます。

```
sjms2:queue:FOO.BAR
```

トピックに接続するには、topic: プレフィックスを含める 必要があり ます。たとえば、トピック Stocks.Prices に接続するには、以下を使用します。

```
sjms2:topic:Stocks.Prices
```

以下のフォーマットを使用して URI にクエリーオプションを追加します（? option=value&option=value&…

286.2. コンポーネントのオプションおよび設定

Simple JMS2 コンポーネントは、以下に示す 15 個のオプションをサポートします。

Name	説明	デフォルト	Type
connectionFactory (advanced)	SjmsComponent を有効にするには ConnectionFactory が必要です。ConnectionResource の一部として直接設定したり、設定したりできます。		ConnectionFactory
connectionResource (advanced)	ConnectionResource は、ConnectionFactory のカスタマイズおよびコンテナー制御を可能にするインターフェースです。詳細は、「プラグ可能な接続リソース管理」を参照してください。		ConnectionResource
connectionCount (common)	このコンポーネントで開始されるエンドポイントで利用可能な最大接続数	1	整数
jmsKeyFormatStrategy (advanced)	JMS 鍵をエンコードおよびデコードするためのプラグ可能なストラテジー。これにより、JMS 仕様に準拠します。Camel は、そのまま使える実装（デフォルト）を提供します。デフォルトのストラテジーは、ドットとハイフン（. および -）を安全にマーシャリングします。JMS ヘッダーキーに不正な文字が含まれるかどうかは気にしない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy
transactionCommit Strategy (transaction)	使用するコミットストラテジーの種類を設定します。Camel は、追加設定なしで、default と batch の 2 つの実装を提供します。		TransactionCommit Strategy
destinationCreation Strategy (advanced)	カスタムの DestinationCreationStrategy を使用します。		DestinationCreation Strategy
timedTaskManager (advanced)	カスタムの TimedTaskManager を使用する場合		TimedTaskManager
messageCreated Strategy (advanced)	Camel が JMS メッセージを送信するときに Camel が javax.jms.Message オブジェクトの新規インスタンスを作成する際に呼び出される指定の MessageCreatedStrategy を使用します。		MessageCreatedStrategy

Name	説明	デフォルト	Type
connectionTestOnBorrow (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用する場合は、プールから返される前に各 javax.jms.Connection をテストする（開始開始）する必要があります。	true	boolean
connectionUsername (security)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource の使用時に javax.jms.Connection を作成するときに使用するユーザー名。		文字列
connectionPassword (security)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource の使用時に javax.jms.Connection を作成する際に使用するパスワード。		文字列
connectionClientId (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource の使用時に javax.jms.Connection の作成時に使用するクライアント ID。		文字列
connectionMaxWait (advanced)	デフォルトの org.apache.camel.component.sjms.jms.ConnectionFactoryResource を使用する場合、プールが使い切られると、空き接続でブロックして待機する最大待機時間（ミリ秒単位）。	5000	Long
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Simple JMS2 エンドポイントは、URI 構文を使用して設定します。

```
sjms2:destinationType:destinationName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

286.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
destinationType	使用する宛先の種類	queue	文字列
destinationName	**必要な** DestinationName は JMS キューまたはトピック名です。デフォルトでは、destinationName はキュー名として解釈されます。		文字列

286.2.2. クエリーパラメーター（37 パラメーター）：

Name	説明	デフォルト	Type
acknowledgementMode (common)	JMS 確認名。SESSION_TRANSACTED、CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE のいずれかです。	AUTO_ACKNOWLEDGE	SessionAcknowledgement Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
consumerCount (consumer)	このエンドポイントに使用されるコンシューマーリスナーの数を設定します。	1	int
永続性 （コンシューマー）	トピックコンシューマーを永続に設定します。	false	boolean
durableSubscriptionId (consumer)	永続トピックに必要な永続サブスクリプション ID を設定します。		文字列
共有 （コンシューマー）	コンシューマーを共有に設定します。	false	boolean
subscriptionId (consumer)	永続または共有トピックに必要なサブスクリプション ID を設定します。		文字列
同期 （コンシューマー）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	true	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
messageSelector (consumer)	JMS メッセージセレクター構文を設定します。		文字列
namedReplyTo (producer)	InOut プロデューサーエンドポイントに使用される宛先名への返信を設定します。		文字列
永続 的（プロデューサー）	メッセージの永続性を有効または無効にするために使用されるフラグ。	true	boolean
producerCount (producer)	このエンドポイントに使用されるプロデューサーの数を設定します。	1	int
ttl (producer)	生成されたメッセージの Time To Live 値を調整するために使用されるフラグ。	-1	Long
allowNullBody (producer)	ボディーのないメッセージの送信を許可するかどうか。このオプションが false で、メッセージボディーが null の場合、JMSException が発生します。	true	boolean
prefillPool (producer)	起動時にプロデューサー接続プールをプレフィルするか、または必要に応じて接続レイジーを作成するかどうか。	true	boolean
responseTimeOut (producer)	InOut 応答をタイムアウトするまでの待機時間を設定します。	5000	Long

Name	説明	デフォルト	Type
asyncStartListener (advanced)	ルートの開始時にコンシューマーメッセージリスナーを非同期に起動するかどうか。たとえば、JmsConsumer がリモート JMS ブローカーへの接続を取得できない場合は、再試行中やフェイルオーバー中にブロックされる可能性があります。これにより、ルートの起動中に Camel がブロックされます。このオプションを true に設定すると、ルートの起動が可能になりますが、JmsConsumer は非同期モードで専用のスレッドを使用して JMS ブローカーに接続します。このオプションを使用すると、接続が確立されないと、WARN レベルで例外がログに記録され、コンシューマーはメッセージを受信できなくなります。次にルートを再起動して再試行できます。	false	boolean
asyncStopListener (advanced)	ルートを停止する際に、コンシューマーメッセージリスナーを非同期的に停止するかどうか。	false	boolean
connectionCount (advanced)	このエンドポイントで利用可能な最大接続数		整数
connectionFactory (advanced)	エンドポイントの connectionFactory を初期化します。これは、コンポーネントの connectionFactory よりも優先されます（存在する場合）。		ConnectionFactory
connectionResource (advanced)	エンドポイントの connectionResource を初期化します。これは、コンポーネントの connectionResource（ある場合）よりも優先されます。		ConnectionResource
destinationCreationStrategy (advanced)	カスタムの DestinationCreationStrategy を使用します。		DestinationCreation Strategy
exceptionListener (advanced)	基礎となる JMS 例外が通知される JMS 例外リスナーを指定します。		ExceptionListener
headerFilterStrategy (advanced)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
includeAllJMSXProperties (advanced)	JMS から Camel メッセージへのマッピング時に、すべての JMSXxxx プロパティーを含めるかどうか。これを true に設定すると、JMSXAppID や JMSXUserID などのプロパティーが含まれます。注記： カスタムの headerFilterStrategy を使用している場合は、このオプションは適用されません。	false	boolean

Name	説明	デフォルト	Type
jmsKeyFormatStrategy (advanced)	JMS 鍵をエンコードおよびデコードするプラグ可能なストラテジー。Camel は、追加設定なしの 2 つの実装（default および passthrough）を提供します。デフォルトのストラテジーは、ドットとハイフン（. および -）を安全にマーシャリングします。passthrough ストラテジーは、鍵をそのまま残します。JMS ヘッダーキーに不正な文字が含まれるかどうかは気にしない JMS ブローカーに使用できます。org.apache.camel.component.jms.JmsKeyFormatStrategy の独自の実装を提供し、表記を使用して参照できます。		JmsKeyFormatStrategy
mapJmsMessage (advanced)	Camel が受信した JMS メッセージを適切なペイロードタイプ（javax.jms.TextMessage など）に自動マッピングするかどうかを指定します。詳細は、以下のマッピングがどのように機能するかについてのセクションを参照してください。	true	boolean
messageCreatedStrategy (advanced)	Camel が JMS メッセージを送信するときに Camel が javax.jms.Message オブジェクトの新規インスタンスを作成する際に呼び出される指定の MessageCreatedStrategy を使用します。		MessageCreatedStrategy
errorHandlerLoggingLevel (logging)	キャッチされない例外についてのデフォルトの errorHandler ロギングレベルの設定を可能にします。	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	デフォルトの errorHandler によってスタックトレースをログに記録するかどうかを制御できます。	true	boolean
トランザクション（トランザクション）	トランザクションモードを使用するかどうかを指定します。	false	boolean
transactionBatchCount (transaction)	トランザクションをコミットする前に処理するメッセージの数を設定する場合は、トランザクションがコミットされます。	-1	int
transactionBatchTimeout (transaction)	バッチトランザクションのタイムアウト（ミリ秒単位）を設定します。値は 1000 以上である必要があります。	5000	Long
transactionCommitStrategy (transaction)	コミットストラテジーを設定します。		TransactionCommit Strategy

Name	説明	デフォルト	Type
sharedJMSSession (transaction)	JMS セッションを他の SJMS エンドポイントと共有するかどうかを指定します。ルートが複数の JMS プロバイダーにアクセスできる場合は、これをオフにします。複数の JMS プロバイダーに対してトランザクションが必要な場合は、jms コンポーネントを使用して XA トランザクションを活用します。	true	boolean

以下は、必要な ConnectionFactory プロバイダーで Sjms2Component を設定する方法を示しています。これは、デフォルトで単一の接続を作成し、コンポーネントの内部プーリング API を使用して格納し、スレッドでセッション作成リクエストに対応できるようにします。

```
Sjms2Component component = new Sjms2Component();
component.setConnectionFactory(new ActiveMQConnectionFactory("tcp://localhost:61616"));
getContext().addComponent("sjms2", component);
```

永続サブスクリプションのサポートに必要な SJMS2 コンポーネントの場合、デフォルトの ConnectionFactoryResource インスタンスを上書きし、clientId プロパティーを設定します。

```
ConnectionFactoryResource connectionResource = new ConnectionFactoryResource();
connectionResource.setConnectionFactory(new
ActiveMQConnectionFactory("tcp://localhost:61616"));
connectionResource.setClientId("myclient-id");

Sjms2Component component = new Sjms2Component();
component.setConnectionResource(connectionResource);
component.setMaxConnections(1);
```

286.3. プロデューサーの使用

286.3.1. InOnly プロデューサー -（デフォルト）

InOnly *プロデューサーは、SJMS2 プロデューサーエンドポイントのデフォルト動作です。*

```
from("direct:start")
    .to("sjms2:queue:bar");
```

286.3.2. InOut プロデューサー

InOut *動作を有効にするには、ExchangePattern 属性 を URI に追加します。デフォルトでは、コンシューマーごとに専用の TemporaryQueue を使用します。*

```
from("direct:start")
    .to("sjms2:queue:bar?exchangePattern=InOut");
```

名前付きのReplyTo を指定できますが、これによりモニターポイントを改善できます。

```
from("direct:start")
    .to("sjms2:queue:bar?exchangePattern=InOut&namedReplyTo=my.reply.to.queue");
```

286.4. コンシューマーの使用

286.4.1. 永続共有サブスクリプション

1 つ以上のコンシューマー間で共有できる永続サブスクリプションを作成するには、以下を実行します。JMS 2.0 準拠の接続ファクトリーを使用して、共通の subscriptionId を指定します。次に、サブスクリプションプロパティーの永続性を設定し、共有を true に設定します。

```
from("sjms2:topic:foo?consumerCount=3&subscriptionId=bar&durable=true&shared=true")
    .to("mock:result");

from("sjms2:topic:foo?consumerCount=2&subscriptionId=bar&durable=true&shared=true")
    .to("mock:result");
```

286.4.2. InOnly Consumer:（デフォルト）

InOnly xonsumer は、SJMS2 コンシューマーエンドポイントのデフォルト Exchange 動作です。

```
from("sjms2:queue:bar")
    .to("mock:result");
```

286.4.3. InOut コンシューマー

InOut 動作を有効にするには、ExchangePattern 属性 を URI に追加します。

```
from("sjms2:queue:in.out.test?exchangePattern=InOut")
    .transform(constant("Bye Camel"));
```

286.5. 高度な使用方法

286.5.1. プラグ可能な接続リソース管理

SJMS2 は、組み込み 接続 プールを使用して JMS 接続リソース管理を提供します。これにより、サードパーティーの API プーリングロジックに依存する必要がなくなります。ただし、J2EE や OSGi コンテナーによって提供される外部接続リソースマネージャーを使用する必要がある場合があります。この SJMS2 では、内部 SJMS2 接続プール機能を上書きするために使用できるインターフェースを提供します。これは、ConnectionResource インターフェースを使用して実行できます。

Connection Resource は、必要に応じて接続をバンドルおよび返すメソッドが、接続プールを SJMS2 コンポーネントに提供するために使用されるコントラクトです。SJMS2 を外部接続プールマネージャーと統合する必要がある場合にユーザーを使用する必要があります。

標準の ConnectionFactory プロバイダーでは、SJMS2 をそのまま提供される ConnectionFactoryResource 実装を使用することが、このコンポーネントに対して最適化されるので、拡張することが推奨されます。

以下は、ActiveMQ PooledConnectionFactory でプラグイン可能な ConnectionResource を使用する例です。

```
public class AMQConnectionResource implements ConnectionResource {
    private PooledConnectionFactory pcf;

    public AMQConnectionResource(String connectString, int maxConnections) {
        super();
        pcf = new PooledConnectionFactory(connectString);
        pcf.setMaxConnections(maxConnections);
        pcf.start();
    }

    public void stop() {
        pcf.stop();
    }

    @Override
    public Connection borrowConnection() throws Exception {
        Connection answer = pcf.createConnection();
        answer.start();
        return answer;
    }

    @Override
    public Connection borrowConnection(long timeout) throws Exception {
        // SNIPPED...
    }

    @Override
    public void returnConnection(Connection connection) throws Exception {
        // Do nothing since there isn't a way to return a Connection
        // to the instance of PooledConnectionFactory
```

```
        log.info("Connection returned");
    }
}
```

次に、ConnectionResource を Sjms2Component に渡します。

```
CamelContext camelContext = new DefaultCamelContext();
AMQConnectionResource pool = new AMQConnectionResource("tcp://localhost:33333", 1);
Sjms2Component component = new Sjms2Component();
component.setConnectionResource(pool);
camelContext.addComponent("sjms2", component);
```

使用方法の完全なサンプルを表示するには、ConnectionResourceIT を参照してください。

286.5.2. セッション、コンシューマー、プロデューサープーリング & キャッシュ管理

近日公開

286.5.3. バッチメッセージのサポート

Sjms2Producer は、List をカプセル化する Exchange を作成して、メッセージのコレクションの公開をサポートします。この Sjms2Producer は List の内容を繰り返し処理し、各メッセージを個別に公開します。

メッセージのバッチを生成する場合、各メッセージに固有のヘッダーを設定する必要がある場合は、SJMS2 BatchMessage クラスを使用できます。Sjms2Producer が BatchMessage リストに遭遇すると、各 BatchMessage を繰り返し処理し、含まれるペイロードとヘッダーを公開します。

以下は BatchMessage クラスの使用例です。まず、BatchMessage のリストを作成します。

```
List<BatchMessage<String>> messages = new ArrayList<BatchMessage<String>>();
for (int i = 1; i <= messageCount; i++) {
    String body = "Hello World " + i;
    BatchMessage<String> message = new BatchMessage<String>(body, null);
    messages.add(message);
}
```

次に一覧を公開します。

```
template.sendBody("sjms2:queue:batch.queue", messages);
```

286.5.4. カスタマイズ可能なトランザクションコミットストラテジー（ローカル JMS トランザクションのみ）

SJMS2 は、TransactionCommitStrategy インターフェースを使用してカスタムおよびプラグイン可能なトランザクションストラテジーを作成する手段を提供します。これにより、SessionTransactionSynchronization がセッションのコミットタイミングを決定するために使用する一意の状況のセットを定義できます。その使用例は BatchTransactionCommitStrategy で、次のセクションで詳しく説明されています。

286.5.5. トランザクションバッチコンシューマーとプロデューサー

SJMS2 コンポーネントは、Producer エンドポイントと Consumer エンドポイントの両方でローカル JMS トランザクションのバッチをサポートするように設計されています。ただし、それぞれでどのように処理されるかは非常に異なります。

SJMS2 コンシューマーエンドポイントは、関連付けられたセッションでコミットする前に X メッセージを処理する簡単な実装です。コンシューマーでバッチ処理トランザクションを有効にするには、最初に transacted パラメーターを true に設定し、transactionBatchCount を追加して 0 を超える値に設定します。たとえば、以下の設定では 10 個のメッセージごとにセッションをコミットします。

```
sjms2:queue:transacted.batch.consumer?transacted=true&transactionBatchCount=10
```

コンシューマーエンドポイントでバッチの処理中に例外が発生すると、セッションロールバックが呼び出され、次の利用可能なコンシューマーにメッセージが再配信されます。また、カウンターは、関連付けられたセッションの BatchTransactionCommitStrategy に対して 0 にリセットされます。JMSRedelivered ヘッダーが true に設定されたメッセージを監視するように、バッチメッセージのプロセッサーにフックを配置することはユーザーの責任です。これは、ある時点でメッセージがロールバックされ、正常な処理が行われたことを示すインジケーターです。

トランザクションバッチコンシューマーは、セッション上でオープントランザクションをコミットする前に、メッセージ間でデフォルトの時間（5000 ミリ秒）を待機する内部タイマーのインスタンスも行います。デフォルト値の 5000ms（最低 1000ms）は、ほとんどのユースケースで十分ですが、さらにチューニングが必要な場合は、transactionBatchTimeout パラメーターを設定するだけで十分です。

```
sjms2:queue:transacted.batch.consumer?
transacted=true&transactionBatchCount=10&transactionBatchTimeout=2000
```

コンテキストスイッチの量によりパフォーマンスに不必要な影響を及ぼすため、許可される最小値は 1000 ミリ秒です。

プロデューサーエンドポイントは処理されますが、異なる方法になります。各メッセージが宛先に

配信された後にプロデューサーを使用すると、Exchange が閉じられ、そのメッセージへの参照がなくなりました。再配信ですべてのメッセージを利用可能にするには、BatchMessages を公開するプロデューサーエンドポイントでトランザクションを有効にするだけです。トランザクションは、バッチリスト内のすべてのメッセージが含まれるエクスチェンジの終了時にコミットされます。追加で設定する必要はありません。以下に例を示します。

```
List<BatchMessage<String>> messages = new ArrayList<BatchMessage<String>>();
for (int i = 1; i <= messageCount; i++) {
    String body = "Hello World " + i;
    BatchMessage<String> message = new BatchMessage<String>(body, null);
    messages.add(message);
}
```

トランザクションを有効にして List を公開します。

```
template.sendBody("sjms2:queue:batch.queue?transacted=true", messages);
```

286.6. 追記

286.6.1. メッセージヘッダーの形式

SJMS2 コンポーネントは、Camel JMS コンポーネントで使用されるのと同じヘッダーフォーマットストラテジーを使用します。このプラグイン可能なストラテジーにより、ネットワーク上で送信されたメッセージが JMS Message 仕様に準拠するようになります。

exchange.in.header の場合、以下のルールがヘッダーキーに適用されます。

- JMS または JMSX で始まるキーが予約されています。
- exchange.in.headers キーはリテラルで、すべて有効な Java 識別子である必要があります（キー名のドットは使用しないでください）。
- Camel は、JMS メッセージの消費時にドットとハイフンと逆順を置き換えます。
  - Camel がメッセージを消費するときに DOT と逆の置換に置き換えられます。
  - これは、Camel がメッセージを消費する際に HYPHEN と逆の置換に置き換えられます。

また、オプションの jmsKeyFormatStrategy も併せて参照してください。この場合、キーのフォーマットに独自のカスタムストラテジーを使用できます。

exchange.in.header では、ヘッダーの値に以下のルールが適用されます。

286.6.2. メッセージの内容

ネットワーク上でコンテンツを提供するには、配信されるメッセージのボディーが JMS メッセージ仕様に準拠することを確認する必要があります。そのため、生成されるすべてのものは、プリミティブまたはそれらのカウンターオブジェクト（ Integer、Long、Characterなど）のいずれかである必要があります。types、String、CharSequence、Date BigDecimal、および BigInteger はすべて toString（） 表現に変換されます。その他のタイプはすべてドロップされます。

286.6.3. クラスタリング

クラスター環境で InOut と SJMS2 を使用する場合は、TemporaryQueue 宛先を使用するか、InOut プロデューサーエンドポイントごとに宛先への一意な名前付き応答を使用する必要があります。メッセージの相関は、ブローカーのメッセージセレクターではなく、エンドポイントによって処理されます。InOut Producer Endpoint は、Message JMSCorrelationID によってキャッシュされた Java Concurrency Exchangers を使用します。これにより、すべてのメッセージが宛先から消費されるため、関連するコンシューマーによって生成される順序で、ブローカーのオーバーヘッドが軽減されると同時にパフォーマンスが向上します。

現在、JMSCorrelationId を使用することが唯一の相関ストラテジーです。InOut コンシューマー はこのストラテジーを使用し、含まれる JMSReplyTo 宛先へのすべての応答メッセージで JMSCorrelationId もコピーされるようにします。

286.7. トランザクションサポート

SJMS2 は現在内部 JMS トランザクションの使用のみをサポートします。Camel Transaction Processor または Java Transaction API(JTA)はサポートされません。

286.7.1. Springless Mean I can't use Spring?

全くない。以下は、Spring DSL を使用した SJMS2 コンポーネントの例になります。

```
<route
   id="inout.named.reply.to.producer.route">
   <from
      uri="direct:invoke.named.reply.to.queue" />
   <to
```

```
        uri="sjms2:queue:named.reply.to.queue?
namedReplyTo=my.response.queue&amp;amp;exchangePattern=InOut" />
</route>
```

Springless とは、Spring JMS API の依存関係から離れるという意味です。SJMS2 のパワーまで、新しい JMS クライアント API が開発されています。

第287章 SLACK コンポーネント

Camel バージョン 2.16 から利用可能

slack コンポーネントを使用すると、 Slack のインスタンスに接続し、事前に確立された Slack 受信 Webhook 経由でメッセージボディーに含まれるメッセージを提供できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-slack</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

287.1. URI 形式

メッセージをチャネルに送信する。

```
slack:#channel[?options]
```

slackuser にダイレクトメッセージを送信します。

```
slack:@username[?options]
```

287.2. オプション

Slack コンポーネントは以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
webhookUrl (producer)	受信 Webhook URL		文字列

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Slack エンドポイントは URI 構文を使用して設定します。

```
slack:channel
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

287.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
channel	メッセージをユーザーに直接送信するためにチャネル名（構文名）または slackuser（構文 userName）が **必要です**。		文字列

287.2.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
iconEmoji (producer)	Slack emoji をアバターとして使用する		文字列
iconUrl (producer)	コンポーネントをチャネルまたはユーザーに送信するときに使用するアバター。		文字列
username (producer)	これは、ボットがメッセージをチャネルまたはユーザーに送信する際に必要なユーザー名です。		文字列
webhookUrl (producer)	受信 Webhook URL		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

287.3. SLACKCOMPONENT

XML を使用した SlackComponent は、インテグレーションの受信 Webhook URL をパラメーターとして含まれる Spring または Blueprint Bean として設定する必要があります。

```
<bean id="slack" class="org.apache.camel.component.slack.SlackComponent">
    <property name="webhookUrl"
value="https://hooks.slack.com/services/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"/>
</bean>
```

Java の場合は、Java コードを使用して設定できます。

287.4. 例

Blueprint を使用する CamelContext は以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" default-activation="lazy">

    <bean id="slack" class="org.apache.camel.component.slack.SlackComponent">
        <property name="webhookUrl"
value="https://hooks.slack.com/services/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"/>
    </bean>

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <route>
            <from uri="direct:test"/>
            <to uri="slack:#channel?iconEmoji=:camel:&amp;username=CamelTest"/>
        </route>
    </camelContext>

</blueprint>
```

287.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント

- *はじめに*

第288章 SMPP コンポーネント

Camel バージョン 2.2 で利用可能

このコンポーネントは、SMPP プロトコルを介して SMSC(Short Message Service Center)へのアクセスを提供し、SMS を送受信します。JSMPP ライブラリーは、プロトコル実装に使用されます。

Camel コンポーネントは現在、SMSC 自体ではなく、ESME （外部 Short Messaging Entity）として動作します。

Camel 2.9 以降、ReplaceSm、QuerySm、SubmitMulti、CancelSm、および DataSm を実行することもできます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-smpp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

288.1. SMS の制限

SMS は信頼性も安全でもありません。 信頼できる配信とセキュアな配信を必要とするユーザーは、XMPP または SIP コンポーネントを使用することを検討し、代わりに選択したプロトコルをサポートするスマートフォンアプリケーションと併用したい場合があります。

- 信頼性： SMPP 標準はエラーを示すフィードバックメカニズムを提供しますが、配信の非再配信および確認は、モバイルネットワークがこれらの応答を非表示またはシミュレートするために一般的ではありません。 たとえば、一部のネットワークは、宛先番号が無効であるか、または電源が入らない場合でも、すべてのメッセージの配信確認を自動的に送信します。 一部のネットワークは、スパムであると考えると警告なしでメッセージをドロップします。 ネットワーク内のスパム検出ルールは非常に欠かせません。1 つの送信元から 1 日あたり 100 件以上のメッセージがスパムとみなされる場合があります。

- Security: ラジオボタン Tower から Recipient handset への最後のホップの基本暗号化があります。 SMS メッセージは、ネットワークの他の部分で暗号化または認証されません。 オペレーターの中には、従業員が顧客からの SMS メッセージの履歴をブラウズしたり、コールしたりすることを可能にするものもあります。 メッセージの送信者のアイデンティティーを簡

単に作成することができます。 レゴリエーターや、モバイル電話業界でも、2 要素認証スキームでの SMS の使用と、セキュリティーが重要なその他の目的において SMS の使用に注意を払っています。

Camel コンポーネントを使用すると、メッセージを SMS ネットワークに送信するのと同じくらい簡単にできますが、これらの問題に対する解決策は提供できません。

288.2. データコーディング、アルファベットおよび国際化文字セット

データコーディングとアルファベットはメッセージごとに指定できます。 デフォルト値をエンドポイントに指定できます。 このオプションの関係と、複数の値が設定されている場合にコンポーネントがどのように動作するかを理解することが重要です。

データコーディングは、SMPP ワイヤ形式の 8 ビットフィールドです。

アルファベットは、データコーディングフィールドのビット 0-3 に対応します。 メッセージクラスが使用される一部のタイプのメッセージでは、（ビット 5 のデータコーディングフィールドを設定することで）、下層にある 2 ビットのデータコーディングフィールドがアルファベットとして解釈されず、ビット 2 と 3 のみがアルファベットに影響しません。

さらに、JSMPP ライブラリーの現行バージョンは、ビット 0 および 1 がメッセージクラスに使用されることを前提とすると、ビット 2 および 3 のみをサポートするように見えます。 このため、JSMPP の Alphabet クラスは ISO-8859-1 を示す 3（バイナリー 0011）の値をサポートしません。

JSMPP はメッセージクラスパラメーターを表しますが、Camel コンポーネントは現在、データコーディングフィールドに対応するビットを手動で設定する方法を提供していません。

送信メッセージにデータコーディングフィールドを設定する場合、Camel コンポーネントは以下の値を考慮し、最初に見つかったものを使用します。

- *ヘッダーに指定されたデータコーディング*
- *ヘッダーに指定されたアルファベット*
- *エンドポイント設定（URI パラメーター）に指定されたデータコーディング*

古いバージョンの Camel では、国際文字セットのサポートにバグがありました。 この機能は、すべてのメッセージにエンコーディングを 1 つ使用した場合にのみ機能し、ユーザーがメッセージごとに変更したい場合に便利です。 これを機能させる必要があるユーザーは、Camel のバージョンに、以下の修正が含まれるようにしてください。

JIRA Issues Macro: com.atlassian.sal.api.net.ResponseStatusException: Unexpected response received.ステータスコード： 404

.

Camel SMPP コンポーネントは、データコーディング値を SMSC に送信しようとするだけでなく、メッセージボディーも分析して Java String(Unicode)に変換して、バイトアレイでどのアルファベットを使用するかを決定する場合、Camel SMPP コンポーネントはデータコーディング値（ヘッダーまたは設定）を考慮せず、指定されたアルファベット（ヘッダーまたはエンドポイントパラメーターのいずれか）のみを考慮します。

String の文字が選択したアルファベットで表現できない場合は、クエスチョンマーク(?)の記号に置き換えられます。 API のユーザーは、メッセージのボディーをコンポーネントに渡す前に ISO-8859-1 に変換できるかどうかのチェックを検討してください。そうでない場合には、アルファベットヘッダーを UCS-2 エンコーディングを要求するように設定します。 アルファベットおよびデータコーディングのオプションが全く指定されていない場合、コンポーネントは必要なエンコーディングを検出し、データコーディングを設定することができます。

アルファベットコードの一覧は、SMPP 仕様 v3.4 セクション 5.2.19 に指定されます。 SMPP 仕様の重要な制限の 1 つは、GSM 3.38（7 ビット）文字セットの使用を明示的に要求するためのアルファベットコードがないことです。 アルファベットの値 0 を選択すると、SMSC デフォルト アルファベットが選択されます。これは通常 GSM 3.38 を意味しますが、保証されません。 SMPP ゲートウェイ Nexmo では、コントロールパネルオプションで、デフォルトを他の文字セットにマッピング できます。ユーザーは SMSC Operator をチェックして、デフォルトとして使用されている文字セットを正確に確認することが推奨されます。

288.3. メッセージ分割およびスロットリング

メッセージボディーを String からバイトアレイに変換した後、Camel コンポーネントは JSMPP に渡す前にメッセージを部分（140 バイト SMS サイズ制限内）に分割します。 これは自動的に完了します。

GSM 3.38 アルファベットが使用される場合、コンポーネントは最大 160 文字を 140 バイトのメッセージのボディーにパックします。 8 ビット文字セットを使用する場合には（例： ISO-8859-1 for

western Europe）、140 文字が 140 バイトのメッセージボディー内で許可されます。 16 ビット UCS-2 エンコーディングを使用する場合は、70 文字のみが各 140 バイトメッセージに適合します。

一部の SMSC プロバイダーがスロットリングルールを実装します。 分割されたメッセージの各部分は、プロバイダーのスロットリングメカニズムによって別々にカウントされる可能性があります。 Camel Throttler コンポーネントは、SMSC にルーティングする前に SMPP ルートでメッセージをスロットリングするのに役立ちます。

288.4. URI 形式

```
smpp://[username@]hostname[:port][?options]
smpps://[username@]hostname[:port][?options]
```

ユーザー 名が指定されていない場合、Camel はデフォルト値 smppclient を提供します。 ポート 番号が指定されていない場合、Camel はデフォルト値 2775 を提供します。 Camel 2.3: プロトコル名が「smpps」、「camel-smpp」の場合には、SSLSocket を使用してサーバーへの接続を開始します。

URI にクエリーオプションを追加するには、?option=value&option=value&…

288.5. URI オプション

SMPP コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有 SmppConfiguration を設定として使用します。		SmppConfiguration
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SMPP エンドポイントは URI 構文を使用します。

```
smpp:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

288.5.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
host	使用する SMSC サーバーのホスト名。	localhost	文字列
port	使用する SMSC サーバーのポート番号。	2775	整数

288.5.2. クエリーパラメーター（38 パラメーター）：

Name	説明	デフォルト	Type
initialReconnectDelay (common)	接続が失われた後に、コンシューマー/プロデューサーが SMSC への再接続を試みた後の最初の遅延をミリ秒単位で定義します。	5000	Long
maxReconnect (common)	SMSC への再接続を試行する最大回数を定義します。SMSC が負のバインド応答を返す場合です。	2147483647	int
reconnectDelay (common)	SMSC への接続が失われ、直前の接続に成功しなかった場合に、再接続試行の間隔（ミリ秒単位）を定義します。	5000	Long
splittingPolicy (common)	長いメッセージを処理するポリシーを指定できます： ALLOW - デフォルトの長いメッセージは、メッセージ TRUNCATE ごとに 140 バイトに分割されます。長いメッセージは分割され、最初のフラグメントのみが SMSC に送信されます。一部のプログラマーは後続のフラグメントを破棄するため、配信されないメッセージの部分を送信する SMPP コネクションの負荷が軽減されます。REJECT - メッセージを分割する必要がある場合、SMPP NegativeResponseException で拒否され、メッセージを表す理由コードが長すぎます。	ALLOW	SmppSplittingPolicy
systemType (common)	このパラメーターは、SMSC（max. 13 文字）にバインドされている ESME（外部 Short Message Entity）のタイプを分類するために使用されます。	cp	文字列

Name	説明	デフォルト	Type
addressRange (consumer)	SMPP 3.4 仕様の 5.2.7 セクションで定義されている SmppConsumer のアドレス範囲を指定できます。SmppConsumer は、この範囲内のアドレス（MSISDN または IP アドレス）をターゲットとする SMSC からのみメッセージを受信します。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
destAddr (producer)	宛先 SME アドレスを定義します。モバイル終了メッセージの場合、これは受信側 MS のディレクトリー番号です。SubmitSm、SubmitMulti、CancelSm、および DataSm のみ。	1717	文字列
destAddrNpi (producer)	SME 宛先アドレスパラメーターで使用される数字 (TON)のタイプを定義します。SubmitSm、SubmitMulti、CancelSm、および DataSm のみ。以下の NPI 値が定義されます： 0: Unknown 1: ISDN(E163/E164)2: Data(X.121)3: Telex(F.69)6: Land Mobile(E.212)8: National 9: Private 10: ERMES 13: Internet(IP)18: WAP Client Id(to be defined by WAP Forum)		byte
destAddrTon (producer)	SME 宛先アドレスパラメーターで使用される数字 (TON)のタイプを定義します。SubmitSm、SubmitMulti、CancelSm、および DataSm のみ。The following TON values are defined: 0: Unknown 1: International 2: National 3: Network Specific 4: Subscriber Number 5: Alphanumeric 6: Abbreviated		byte

Name	説明	デフォルト	Type
lazySessionCreation (producer)	Camel プロデューサーの起動時に SMSC が利用できない場合、セッションは遅延して例外を回避します。Camel は最初のエクスチェンジのメッセージヘッダー 'CamelSmppSystemId' と 'CamelSmppPassword' をチェックします。Camel はデータが存在する場合は、これらのデータを使用して SMSC に接続します。	false	boolean
numberingPlanIndicator (producer)	SME で使用される数値計画インジケーター(NPI)を定義します。以下の NPI 値が定義されます： 0: Unknown 1: ISDN(E163/E164)2: Data(X.121)3: Telex(F.69)6: Land Mobile(E.212)8: National 9: Private 10: ERMES 13: Internet(IP)18: WAP Client Id(to be defined by WAP Forum)		byte
priorityFlag (producer)	元の SME が優先度レベルを短いメッセージに割り当てることができます。SubmitSm および SubmitMulti のみ。4 つの優先度レベルがサポートされます。レベル 0（最低）優先度 1: レベル 1 の優先度 2: レベル 2 の優先度 3: レベル 3（最高）の優先度		byte
protocolId (producer)	プロトコル ID		byte
registeredDelivery (producer)	SMSC 配信受信や SME による確認応答を要求するために使用されます。0: No SMSC delivery receipt Request: 0: No SMSC delivery receipt が定義されます。1: SMSC 配信の受信は、最終的な配信結果が成功または失敗した場合に要求されました。2: 直前の配信結果が配信に失敗する場合に SMSC 配信がリクエストされます。		byte
replaceIfPresentFlag (producer)	SMSC を要求して、以前に送信されたメッセージ（配信を保留しているメッセージ）を置き換えるために使用されます。SMSC は、ソースアドレス、宛先アドレス、サービスタイプが新規メッセージの同じフィールドと一致すると提供された既存のメッセージを置き換えます。フラグ値が定義されている場合は、置換します： 0: Don't replace 1: Replace		byte
serviceType (producer)	サービスタイプパラメーターを使用して、メッセージに関連付けられた SMS アプリケーションサービスを指定できます。以下の一般的なサービスタイプが定義されます： CMT: Cellular Messaging CPT: Cellular Paging VMN: Voice Mail Notification VMA: Voice Mail Alerting WAP: Wireless Application Protocol USSD: Unstructured Supplementary Services Data	CMT	文字列

Name	説明	デフォルト	Type
sourceAddr (producer)	このメッセージを出た SME(Short Message Entity)のアドレスを定義します。	1616	文字列
sourceAddrNpi (producer)	SME originator アドレスパラメーターで使用される数値計画インジケーター(NPI)を定義します。以下の NPI 値が定義されます： 0: Unknown 1: ISDN(E163/E164)2: Data(X.121)3: Telex(F.69)6: Land Mobile(E.212)8: National 9: Private 10: ERMES 13: Internet(IP)18: WAP Client Id(to be defined by WAP Forum)		byte
sourceAddrTon (producer)	SME originator アドレスパラメーターで使用される数字(TON)のタイプを定義します。The following TON values are defined: 0: Unknown 1: International 2: National 3: Network Specific 4: Subscriber Number 5: Alphanumeric 6: Abbreviated		byte
typeOfNumber (producer)	SME で使用される数字(TON)のタイプを定義します。The following TON values are defined: 0: Unknown 1: International 2: National 3: Network Specific 4: Subscriber Number 5: Alphanumeric 6: Abbreviated		byte
enquireLinkTimer (advanced)	信用チェックの間隔（ミリ秒単位）を定義します。ESME と SMSC 間の通信パスをテストする際に、信頼できるチェックが使用されます。	5000	整数
sessionStateListener (advanced)	レジストリー の org.jsmpp.session.SessionStateListener を参照して、セッションの状態が変更されたときにコールバックを受け取ることができます。		SessionStateListener
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
transactionTimer (advanced)	トランザクション後に許可される非アクティブの最大期間を定義します。その後、SMPP エンティティーはセッションがアクティブでなくなることを想定できます。このタイマーは、SMPP エンティティー（SMSC または ESME など）のいずれかでアクティブにすることができます。	10000	整数
Alpha bet(codec)	SMPP 3.4 仕様に従ってデータのエンコーディングを定義します（セクション 5.2.19. 0: SMSC Default Alphabet 4: 8 bit Alphabet 8: UCS2 Alphabet）。		byte

Name	説明	デフォルト	Type
DataCoding (codec)	SMPP 3.4 仕様（「5.2.19」セクション）に従ってデータコーディングを定義します。データエンコーディングの例： 0: SMSC Default Alphabet 3: Latin 1(ISO-8859-1)4: Octet unspecified(8-bit binary)8: UCS2(ISO/IEC-10646)13: Extended Kanji JIS(X 0212-1990)		byte
encoding (codec)	短いメッセージユーザーデータのエンコーディングスキームを定義します。SubmitSm、ReplaceSm、および SubmitMulti のみ。	ISO-8859-1	文字列
httpProxyHost (proxy)	HTTP プロキシー経由で SMPP をトンネルする必要がある場合は、この属性を HTTP プロキシーのホスト名または IP アドレスに設定します。		文字列
httpProxyPassword (proxy)	HTTP プロキシーに Basic 認証が必要な場合は、この属性を HTTP プロキシーに必要なパスワードに設定します。		文字列
httpProxyPort (proxy)	HTTP プロキシー経由で SMPP をトンネルする必要がある場合は、この属性を HTTP プロキシーのポートに設定します。	3128	整数
httpProxyUsername (proxy)	HTTP プロキシーに Basic 認証が必要な場合は、この属性を HTTP プロキシーに必要なユーザー名に設定します。		文字列
proxyHeaders (proxy)	これらのヘッダーは、接続を確立するときにプロキシーサーバーに渡されます。		マップ
パスワード （セキュリティー）	SMSC サーバーに接続するためのパスワード。		文字列
systemId (security)	SMSC サーバーに接続するためのシステム ID（ユーザー名）。	smppclient	文字列
usingSSL (security)	smpps プロトコルで SSL を使用するかどうか	false	boolean

これらのオプションは、いくつでも指定できます。

> ***smpp://smppclient@localhost:2775?password=password&enquireLinkTimer=3000&transactionTimer=5000&systemType=consumer***

288.6. プロデューサーメッセージヘッダー

以下のメッセージヘッダーを使用して、SMPP プロデューサーの動作に影響を与えることができます。

ヘッダー	Type	説明
CamelSmppDestAddr	**List/String**	**SubmitSm、SubmitMulti、CancelSm、および DataSm に対してのみ**、宛先 SME アドレスを定義します。モバイル終了メッセージの場合、これは受信側 MS のディレクトリー番号です。SubmitMulti と **String** は **List<String** > である必要があります。
CamelSmppDestAddrTon	**Byte**	**SME 宛先アドレスパラメーターで使用される数字(TON)のタイプを定義するのは、SubmitSm、SubmitMulti、CancelSm、および DataSm のみ** です。上記で定義した **sourceAddrTon** URI オプション値を使用します。
CamelSmppDestAddrNpi	**Byte**	**SME 宛先アドレスパラメーターで使用される数値計画インジケーター(NPI)を定義するのは、SubmitSm、SubmitMulti、CancelSm、および DataSm のみ** です。上記で定義した URI オプション **sourceAddrNpi** 値を使用します。
CamelSmppSourceAddr	**文字列**	このメッセージを出た SME(Short Message Entity)のアドレスを定義します。
CamelSmppSourceAddrTon	**Byte**	SME originator アドレスパラメーターで使用される数字(TON)のタイプを定義します。上記で定義した **sourceAddrTon** URI オプション値を使用します。
CamelSmppSourceAddrNpi	**Byte**	SME originator アドレスパラメーターで使用される数値計画インジケーター(NPI)を定義します。上記で定義した URI オプション **sourceAddrNpi** 値を使用します。
CamelSmppServiceType	**文字列**	サービスタイプパラメーターを使用して、メッセージに関連付けられた SMS アプリケーションサービスを指定できます。上記の URI オプションの **serviceType** 設定を使用します。

ヘッダー	Type	説明
CamelSmppRegisteredDelivery	**Byte**	SubmitSm、ReplaceSm、SubmitMulti、および DataSm Is のみが、SMSC 配信受信や、SME による確認応答を要求するため に使用されます。上記の URI オプション **registeredDelivery** 設定を使用します。
CamelSmppPriorityFlag	**Byte**	SubmitSm および SubmitMulti に対してのみ、送信元の SME が優先度レベルを短いメッセージに割り当てることができます。上記の URI オプションの **priorityFlag** 設定を使用します。
CamelSmppScheduleDeliveryTime	**Date**	SubmitSm、SubmitMulti、および ReplaceSm に対してのみ、このパラメーターで、メッセージ配信が最初に試行されるスケジュール済み時間を指定します。これは、SMSC がこのメッセージの配信を試行する現在の SMSC 時間からの絶対日時または相対時間を定義します。絶対時間形式または相対時間形式で指定できます。時間形式のエンコーディングは、smpp 仕様 v3.4 の 第 7.1.1 章に指定されています。
CamelSmppValidityPeriod	**String /Date**	SubmitSm、SubmitMulti、および ReplaceSm The validity period パラメーターは SMSC の有効期限を示します。その後は、宛先に配信されなかった場合にメッセージを破棄します。日付 と指定された場合 **は**、絶対時間として解釈されます。Camel 2.9.1 **以降：** smpp 仕様 v3.4 の 第 7.1.1 の章で指定されているとおりに**String** として指定する場合は、絶対時間形式または相対時間形式で定義できます。
CamelSmppReplaceIfPresentFlag	**Byte**	SubmitSm および SubmitMulti に対してのみ、指定されたフラグパラメーターを使用して SMSC を要求し、以前に送信されたメッセージを置き換えますが、配信が保留中です。SMSC は、ソースアドレス、宛先アドレス、サービスタイプが新規メッセージの同じフィールドと一致すると提供された既存のメッセージを置き換えます。**0**、Don't replace および **1**、Replace の値が定義されます。
CamelSmppAlphabet / CamelSmppDataCoding	**Byte**	Camel 2.5 For SubmitSm, SubmitMulti and ReplaceSm （Camel 2.9 より前のバージョンでは、**CamelSmppAlphabet** の代わりに **CamelSmppDataCoding** を使用してください）。 SMPP 3.4 仕様に準拠したデータコーディングセクション 5.2.19。上記の URI オプション **アルファベット** 設定を使用します。
CamelSmppOptionalParameters	**Map< String , String >**	**非推奨となり、**Camel 2.13.0/3.0.0 Camel 2.10.5 **および** 2.11.1 **以降では削除され、**SubmitSm、SubmitMulti、および DataSm **専用**となります。任意のパラメーターは SMSC によって返されます。

ヘッダー	Type	説明
CamelSmppOptionalParameter	**map<Short,Object>**	Camel 2.10.7 **以降、および** SubmitSm、SubmitMulti、**および** DataSm **に対してのみ**、SMSC に送信される任意のパラメーターです。値は以下のように変換されます。**string →org.jsmpp.bean.OptionalParameter.COctetString**,**byte[] → org.jsmpp.bean.OptionalParameter.OctetString**,**Byte → org.jsmpp.bean.OptionalParameter.Byte**,**Integer → org.jsmpp.bean.OptionalParameter.Int**, short → **org.jsmpp.bean.OptionalParameter. Short** ,**null → org.jsmpp.bean.OptionalParameter.Null**
CamelSmppEncoding	文字列	**Camel 2.14.1 および Camel 2.15.0 以降**では *SubmitSm、SubmitMulti、および DataSm* のみ。 メッセージボディーのバイトのエンコーディング（文字セット名）を指定します。 メッセージボディーが文字列である場合、Java String は常に Unicode であるため、これは関係しません。 ボディーがバイトアレイである場合は、このヘッダーを使用して ISO-8859-1 またはその他の値を指定できます。 デフォルト値は、エンドポイント設定パラメーター **エンコーディング**で指定されます。
CamelSmppSplittingPolicy	文字列	**Camel 2.14.1 および Camel 2.15.0 以降**では *SubmitSm、SubmitMulti、および DataSm* のみ。 このエクスチェンジのメッセージ分割のポリシーを指定します。 設定可能な値は、エンドポイント設定パラメーター split **Policy で説明されてい**ます。

以下のメッセージヘッダーは、SMPP プロデューサーによって使用され、メッセージヘッダーに SMSC からの応答を設定します。

ヘッダー	Type	説明
CamelSmppId	**List<String>/String**	後で使用できるように提出した短いメッセージを識別するための ID。**Camel 2.9.0** から： ReplaceSm、QuerySm、CancelSm、および DataSm の場合、このヘッダー vaule は **String** です。SubmitSm または SubmitMultiSm の場合、ヘッダー vaule は **List<String> です**。
CamelSmppSentMessageCount	**整数**	Camel 2.9 **以降では、SubmitSm および SubmitMultiSm に対してのみ**、送信されたメッセージの総数。

ヘッダー	Type	説明
CamelSmppError	**Map<String, List<Map<String, Object>>>**	**Camel 2.9 以降では、SubmitMultiSm に対してのみ**、短いメッセージを送信して発生したエラーは **Map<String, List<Map<String, Object** >>（messageID :(destAddr : address, error : errorCode)）です。
CamelSmppOptionalParameters	**Map<String, String>**	**非推奨となり、Camel 2.13.0/3.0.0 から Camel 2.11.1 以降ではの み削除さ** れます。メッセージの送信により SMSC から返されるオプションのパラメーターです。
CamelSmppOptionalParameter	**map<Short, Object>**	**Camel 2.10.7 以降では、DataSm に対してのみ** 2.11.2 以降のみ、メッセージを送信して SMSC から返される任意のパラメーターです。キーは、オプションのパラメーターの **Short** コードです。値は以下のように変換されます。 **org.jsmpp.bean.OptionalParameter.COctetString → String**,**org.jsmpp.bean.OptionalParameter.OctetString → byte[]**, **org.jsmpp.bean.OptionalParameter.Byte → Byte**,**org.jsmpp.bean.OptionalParameter.Int → Integer**, **org.jsmpp.bean.OptionalParameter.Short → Short**,**org.jsmpp.bean.OptionalParameter.Null → null**

288.7. コンシューマーメッセージヘッダー

以下のメッセージヘッダーは、SMPP コンシューマーによって使用され、メッセージヘッダーに SMSC からの要求データを設定します。

ヘッダー	Type	説明
CamelSmppSequenceNumber	**整数**	**AlertNotification、deliverSm、および DataSm A のシーケンス番号のみ** により、応答 PDU を要求 PDU と関連付けることができます。関連付けられた SMPP 応答 PDU はこのフィールドを保持する必要があります。
CamelSmppCommandId	**整数**	**AlertNotification、DeliverSm、および DataSm The コマンド id フィールドのみ** が特定の SMPP PDU を特定します。定義された値の一覧は、smpp 仕様 v3.4 の 5.1.2.1 章を参照してください。

ヘッダー	Type	説明
CamelSmppSourceAddr	**文字列**	**AlertNotification、deliversSm および DataSm に対してのみ**、このメッセージが発信された SME(Short Message Entity)のアドレスを定義します。
CamelSmppSourceAddrNpi	**Byte**	**AlertNotification および DataSm に対してのみ**、SME originator アドレスパラメーターで使用される数値計画インジケーター(NPI)を定義します。上記で定義した URI オプション **sourceAddrNpi** 値を使用します。
CamelSmppSourceAddrTon	**Byte**	**AlertNotification および DataSm に対してのみ**、SME originator アドレスパラメーターで使用される数字(TON)のタイプを定義します。上記で定義した **sourceAddrTon** URI オプション値を使用します。
CamelSmppEsmeAddr	**文字列**	**AlertNotification にのみ**、宛先 ESME アドレスを定義します。モバイル終了メッセージの場合、これは受信側 MS のディレクトリー番号です。
CamelSmppEsmeAddrNpi	**Byte**	**AlertNotification では、ESME originator アドレスパラメーターで使用される数値計画インジケーター(NPI)のみ** を定義します。上記で定義した URI オプション **sourceAddrNpi** 値を使用します。
CamelSmppEsmeAddrTon	**Byte**	**AlertNotifications は、** ESME originator アドレスパラメーターで使用される数字(TON)のタイプを定義します。上記で定義した **sourceAddrTon** URI オプション値を使用します。
CamelSmppId	**文字列**	**smsc DeliveryReceipt および DataSm にのみ**、最初に送信されたときに SMSC によってメッセージに割り当てられたメッセージ ID。
CamelSmppDelivered	**整数**	配信された短いメッセージの **smsc DeliveryReceipt Number のみ**。これは、元のメッセージがディストリビューションリストに送信された場合にのみ該当します。この値は、必要に応じて先頭のゼロとともにパディングされます。
CamelSmppDoneDate	**Date**	**smsc DeliveryReceipt のみ** で、短いメッセージが最終状態に達した日時のみ。形式は YYMMDDhhmm です。

ヘッダー	Type	説明
CamelSmppStatus	**DeliveryReceiptState**	smsc DeliveryReceipt **のみ：** メッセージの最終ステータス。以下の値が定義されます。 **delIVRD**: メッセージは宛先(**EXPIRED**)に配信されます。 メッセージ有効期間の有効期限が切れました **DELETED**: メッセージが削除され、**UNDELIV**: メッセージは配信できません。**ACCEPTD**: メッセージは受け入れられる状態です（つまり、顧客サービスによるサブスクライバーの代わりに手動で読み取られています）、**UNKNOWN**: Message is in invalid state,**REJECTD**: メッセージが拒否状態である
CamelSmppCommandStatus	**整数**	DataSm The Command status of the **メッセージの場合のみ**。
CamelSmppError	**文字列**	smsc DeliveryReceipt where appropriate this may have a Network specific error code or an SMSC error code for the message.これらのエラーは Network または SMSC に固有のエラーであり、ここには含まれません。
CamelSmppSubmitDate	**Date**	smsc DeliveryReceipt **短縮メッセージが送信された日時のみ**。置き換えられたメッセージの場合、これは元のメッセージが置き換えられた日付になります。形式は YYMMDDhhmm です。
CamelSmppSubmitted	**整数**	最初 **に送信された短いメッセージの** smsc DeliveryReceipt **のみ** です。これは、元のメッセージがディストリビューションリストに送信された場合にのみ該当します。この値は、必要に応じて先頭のゼロとともにパディングされます。
CamelSmppDestAddr	**文字列**	DeliverSm **および** DataSm **のみ：** 宛先 SME アドレスを定義します。モバイル終了メッセージの場合、これは受信側 MS のディレクトリー番号です。
CamelSmppScheduleDeliveryTime	**文字列**	DeliverSm **のみ：** このパラメーターは、メッセージ配信が最初に試行されるスケジュール済み時間を指定します。これは、SMSC がこのメッセージの配信を試行する現在の SMSC 時間からの絶対日時または相対時間を定義します。絶対時間形式または相対時間形式で指定できます。時間形式のエンコーディングは、smpp 仕様 v3.4 のセクション 7.1.1 で指定します。
CamelSmppValidityPeriod	**文字列**	DeliverSm The validity period **パラメーターは**SMSC の有効期限を示します。その後は、宛先に配信されない場合はメッセージを破棄します。絶対時間形式または相対時間形式で定義できます。絶対および相対時間形式のエンコーディングは、smpp 仕様 v3.4 のセクション 7.1.1 で指定します。

ヘッダー	Type	説明
CamelSmppServiceType	**文字列**	**DeliverSm および DataSm にのみ**、サービスタイプパラメーターは、メッセージに関連付けられた SMS Application サービスを示します。
CamelSmppRegisteredDelivery	**Byte**	**DataSm Is に対してのみ**、配信受信や SME による確認応答の要求に使用されます。上記の Producer ヘッダーリストと同じ値。
CamelSmppDestAddrNpi	**Byte**	**DataSm に対してのみ**、宛先アドレスパラメーターの NPI（数値計画インジケーター）を定義します。上記で定義した URI オプション **sourceAddrNpi** 値を使用します。
CamelSmppDestAddrTon	**Byte**	**DataSm に対してのみ**、宛先アドレスパラメーターの数字(TON)のタイプを定義します。上記で定義した **sourceAddrTon** URI オプション値を使用します。
CamelSmppMessageType	**文字列**	**Camel 2.6 以降**: **AlertNotification**: an SMSC alert notification,**DataSm**: an SMSC data short message, **DeliveryReceipt**: an SMSC delivery receipt, DeliverSm : an SMSC delivery receipt, **DeliverSm**: an SMSC deliver short message
CamelSmppOptionalParameters	**Map<String, Object>**	**非推奨となり、Camel 2.13.0/3.0.0 Camel 2.10.5 以降で削除さ れる予定です。これは DeliverSm に対してのみ** 削除されます。オプションのパラメーターは、SMSC によって送信される任意のパラメーターです。
CamelSmppOptionalParameter	**map<Short, Object>**	**Camel 2.10.7、2.11.2 以降、および DeliverSm に対してのみ**、オプションのパラメーターが SMSC によって送信される。キーは、オプションのパラメーターの **Short** コードです。値は以下のように変換されます。 **org.jsmpp.bean.OptionalParameter.COctetString → String**,**org.jsmpp.bean.OptionalParameter.OctetString → byte[]**, **org.jsmpp.bean.OptionalParameter.Byte → Byte**,**org.jsmpp.bean.OptionalParameter.Int → Integer**, **org.jsmpp.bean.OptionalParameter.Short → Short**,**org.jsmpp.bean.OptionalParameter.Null → null**

ヒント

JSMPP ライブラリー の詳細は、JSMPP ライブラリー のドキュメントを参照してください。

288.8. 例外処理

このコンポーネントは、一般的な Camel 例外処理機能をサポートします。

SubmitSm（デフォルトアクション）でメッセージを送信したエラーが発生した場合、org.apache.camel.component.smpp.SmppException がネストされた例外 (org.jsmpp.extra.NegativeResponseException)でスローされます。NegativeResponseException.getCommandStatus（）を呼び出して、正確な SMPP ネガティブ応答コードを取得します。値は SMPP 仕様 3.4、セクション 5.1.3 で説明されています。
Camel 2.8 以降: SMPP コンシューマーが DeliverSm または DataSm の短いメッセージを受信し、これらのメッセージの処理に失敗した場合、その失敗を処理する代わりに ProcessRequestException をスローすることもできます。この場合、この例外は基礎となる JSMPP ライブラリー に転送され、含まれるエラーコードが SMSC に戻ります。この機能は、SMSC に後で短いメッセージを再送するよう指示するなどに役立ちます。これは、以下のコード行で実行できます。

```
from("smpp://smppclient@localhost:2775?
password=password&enquireLinkTimer=3000&transactionTimer=5000&systemType=consum
er")
  .doTry()
    .to("bean:dao?method=updateSmsState")
  .doCatch(Exception.class)
    .throwException(new ProcessRequestException("update of sms state failed", 100))
  .end();
```

エラーコードの完全なリストとそれらの意味は、SMPP 仕様 を参照してください。

288.9. サンプル

Java DSL を使用して SMS を送信するルート：

```
from("direct:start")
  .to("smpp://smppclient@localhost:2775?

password=password&enquireLinkTimer=3000&transactionTimer=5000&systemType=producer
");
```

Spring XML DSL を使用して SMS を送信するルート：

```
<route>
  <from uri="direct:start"/>
  <to uri="smpp://smppclient@localhost:2775?

password=password&amp;enquireLinkTimer=3000&amp;transactionTimer=5000&amp;systemType=prc
ducer"/>
</route>
```

Java DSL を使用して SMS を受信するルート：

```
from("smpp://smppclient@localhost:2775?
password=password&enquireLinkTimer=3000&transactionTimer=5000&systemType=consum
er")
  .to("bean:foo");
```

Spring XML DSL を使用して SMS を受信するルート：

```
  <route>
    <from uri="smpp://smppclient@localhost:2775?

password=password&amp;enquireLinkTimer=3000&amp;transactionTimer=5000&amp;systemType=co
nsumer"/>
    <to uri="bean:foo"/>
  </route>
```

ヒント

SMSC シミュレーター は、テストに SMSC シミュレーターが必要な場合は、Logica が提供するシミュレーターを使用できます。

288.10. デバッグロギング

このコンポーネントにはログレベル DEBUG があります。これは、問題のデバッグに役立ちます。log4j を使用する場合は、以下の行を設定に追加できます。

```
log4j.logger.org.apache.camel.component.smpp=DEBUG
```

288.11. 関連項目

- *Configuring Camel (Camel の設定)*

- *コンポーネント*
- *エンドポイント*
- *はじめに*

第289章 SNMP コンポーネント

Camel バージョン 2.1 で利用可能

snmp: コンポーネントを使用すると、SNMP 対応のデバイスをポーリングしたり、トラップを受信することができます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-snmp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

289.1. URI 形式

```
snmp://hostname[:port][?Options]
```

コンポーネントは、SNMP 対応のデバイスからの OID 値をポーリングし、トラップの受信に対応します。

URI にクエリーオプションを追加するには、?option=value&option=value&...

289.2. SNMP プロデューサー

2.18 リリースから利用可能

また、GET メソッドを使用して情報を要求するために使用することもできます。

応答ボディータイプは org.apache.camel.component.snmp.SnmpMessage です。

289.3. オプション

SNMP コンポーネントにはオプションがありません。

SNMP エンドポイントは、URI 構文を使用して設定します。

```
snmp:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

289.3.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
host	SNMP 対応デバイスに **必要な** ホスト名		文字列
port	SNMP 対応デバイスに **必要な** ポート番号		整数

289.3.2. クエリーパラメーター（34 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
遅延 （コンシューマー）	更新レートを秒単位で設定します。	60000	Long
OID （コンシューマー）	対象の値を定義します。よく理解するには、Wikipedia をご覧ください。単一の OID または OID の詳細な一覧を提供できます。例：oids=1.3.6.1.2.1.1.3.0,1.3.6.1.2.1.25.3.2.1.5.1,1.3.6.1.2.1.25.3.5.1.1.1,1.3.6.1.2.1.43.5.1.1.11.1		文字列
プロトコル （コンシューマー）	使用するプロトコルを選択できます。udp または tcp のいずれかを使用できます。	udp	文字列

Name	説明	デフォルト	Type
再試行 （コンシューマー）	要求を取り消す前に再試行が実行される頻度を定義します。	2	int
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
snmpCommunity （コンシューマー）	snmp 要求のコミュニティーオクテット文字列を設定します。	public	文字列
snmpContextEngineId (consumer)	スコープ指定された PDU のコンテキストエンジン ID フィールドを設定します。		文字列
snmpContextName (consumer)	このスコープ指定の PDU のコンテキスト名フィールドを設定します。		文字列
snmpVersion （コンシューマー）	要求の snmp バージョンを設定します。値が 0 の場合は SNMPv1、1 は SNMPv2c を意味し、値 3 は SNMPv3 を意味します。	0	int
タイムアウト （コンシューマー）	要求のタイムアウト値をミリ秒単位で設定します。	1500	int
Type (consumer)	ポーリング、トラップなど、実行する操作。		SnmpActionType
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

Name	説明	デフォルト	Type
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit

Name	説明	デフォルト	Type
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
authenticationPassphrase (security)	認証パスフレーズ。null ではない場合、authenticationProtocol も null ではない必要があります。RFC3414 11.2 では、パスフレーズの最小長が 8 バイトである必要があります。authenticationPassphrase の長さが 8 バイト未満の場合は、IllegalArgumentException がスローされます。		文字列
authenticationProtocol (security)	セキュリティーレベルが認証を有効にするように設定されている場合に使用する認証プロトコル。使用できる値は MD5、SHA1 です。		文字列
privacyPassphrase (security)	プライバシーパスフレーズ。null ではない場合、PrivacyProtocol も null ではない必要があります。RFC3414 11.2 では、パスフレーズの最小長が 8 バイトである必要があります。authenticationPassphrase の長さが 8 バイト未満の場合は、IllegalArgumentException がスローされます。		文字列
privacyProtocol (security)	このユーザーに関連付けられるプライバシープロトコル ID。null に設定すると、このユーザーは暗号化されていないメッセージのみをサポートします。		文字列
securityLevel (security)	このターゲットのセキュリティーレベルを設定します。指定したセキュリティーレベルは、このターゲットに設定されたセキュリティー名に関連付けられたセキュリティーモデルに依存する必要があります。値 1 は認証がなく、暗号化なしを意味します。このセキュリティーレベルでメッセージを作成して読み取りできますが、値 2 は認証と暗号化なしです。適切な認証キーを持つメッセージのみがこのセキュリティーレベルでメッセージを作成できますが、メッセージの内容を読み取ることができます。値が 3 の場合は、「Authentication and encryption」を意味します。このセキュリティーレベルでは、正しい認証キーを持つメッセージしか作成できず、正しい暗号化/暗号キーを持つメッセージのみがメッセージの内容を読み取ることができます。	3	int
securityName (security)	このターゲットで使用するセキュリティー名を設定します。		文字列

289.4. ポーリングの結果

状況を考慮すると、以下の OID をポーリングします。

OID

```
1.3.6.1.2.1.1.3.0
1.3.6.1.2.1.25.3.2.1.5.1
1.3.6.1.2.1.25.3.5.1.1.1
1.3.6.1.2.1.43.5.1.1.11.1
```

結果は以下のようになります。

toString 変換の結果

```
<?xml version="1.0" encoding="UTF-8"?>
<snmp>
  <entry>
    <oid>1.3.6.1.2.1.1.3.0</oid>
    <value>6 days, 21:14:28.00</value>
  </entry>
  <entry>
    <oid>1.3.6.1.2.1.25.3.2.1.5.1</oid>
    <value>2</value>
  </entry>
  <entry>
    <oid>1.3.6.1.2.1.25.3.5.1.1.1</oid>
    <value>3</value>
  </entry>
  <entry>
    <oid>1.3.6.1.2.1.43.5.1.1.11.1</oid>
    <value>6</value>
  </entry>
  <entry>
    <oid>1.3.6.1.2.1.1.1.0</oid>
    <value>My Very Special Printer Of Brand Unknown</value>
  </entry>
</snmp>
```

認識されている可能性があるため、requested....1.3.6.1.2.1.1.1.0 よりも多くの結果が存在します。この特別なケースでは、デバイスにより自動的に入力されます。したがって、何回かかっても、準備した数よりも多くのリクエストを受け取ってしまう可能性があります。

289.5. 例

リモートデバイスをポーリングします。

```
snmp:192.168.178.23:161?protocol=udp&type=POLL&oids=1.3.6.1.2.1.1.5.0
```

トラップレシーバーを設定します（ここでは OID 情報は必要ありません）。

```
snmp:127.0.0.1:162?protocol=udp&type=TRAP
```

Camel 2.10.0 から は、メッセージヘッダー 'securityName', ピアアドレスで SNMP TRAP のコミュニティーをメッセージヘッダー 'peerAddress' で取得することができます。

Java でのルーティングの例 :（SNMP PDU を XML 文字列に変換）

```
from("snmp:192.168.178.23:161?protocol=udp&type=POLL&oids=1.3.6.1.2.1.1.5.0").
convertBodyTo(String.class).
to("activemq:snmp.states");
```

289.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第290章 SOAP DATAFORMAT

Camel バージョン 2.3 の時点で利用可能

SOAP は、JAXB2 および JAX-WS アノテーションを使用して SOAP ペイロードをマーシャリングおよびアンマーシャリングする Data Format です。これは、CXF スタックを使用せずに Apache CXF の基本的な機能を提供します。

サポート対象の SOAP バージョン

SOAP 1.1 がデフォルトでサポートされます。SOAP 1.2 は Camel 2.11 以降でサポートされます。

名前空間接頭辞のマッピング

SOAP データフォーマットを使用してマーシャリングする際に名前空間接頭辞のマッピングを制御する方法については、JAXB を参照してください。

290.1. SOAP オプション

SOAP データフォーマットは、以下に示す 7 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
contextPath		文字列	JAXB クラスが置かれているパッケージ名。
encoding		文字列	特定のエンコーディングを過剰に実行し、特定のエンコーディングを使用します。

Name	デフォルト	Java タイプ	説明
elementNameStrategyRef		文字列	レジストリーから検索する要素ストラテジーを参照します。要素名ストラテジーは、2 つの目的で使用されます。1 つ目は、オブジェクトを SOAP メッセージにマーシャリングする際に、指定のオブジェクトと soap アクションの xml 要素名を検索することです。2 つ目は、特定の soap フォールト名に対して Exception クラスを見つけることです。以下の 3 つの要素ストラテジークラス名は追加設定なしで提供されます。QNameStrategy: インスタンス化時に設定された固定 qName を使用します。例外ルックアップはサポートされていない TypeNameStrategy: 指定のタイプの XMLType アノテーションから名前および namespace を使用します。名前空間が設定されていない場合は、package-info が使用されます。例外検索はサポートされていない ServiceInterfaceStrategy - webservice インターフェースからの情報を使用して、タイプ名を判別し、SOAP 障害の例外クラスを検索します。3 つのクラスはすべてパッケージ名 org.apache.camel.dataformat.soap.name にあります。cxf-codegen または同様のツールで Web サービススタブコードを生成した場合は、ServiceInterfaceStrategy の使用が必要になる可能性があります。アノテーションが付けられたサービスインターフェースがない場合は、QNameStrategy または TypeNameStrategy を使用する必要があります。
version	1.1	文字列	SOAP version は 1.1 または 1.2 のいずれかにする必要があります。デフォルトは 1.1 です。
namespacePrefixRef		文字列	JAXB または SOAP を使用してマーシャリングする場合、JAXB 実装は ns2、ns3、ns4 などの名前空間接頭辞を自動的に割り当てます。このマッピングを制御するために、Camel では必要なマッピングが含まれるマップを参照できます。
schema		文字列	既存のスキーマに対して検証する。接頭辞 classpath:、file:、または http: を使用して、リソースの解決方法を指定できます。','文字を使用して、複数のスキーマファイルを分離できます。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

290.2. ELEMENTNAMESTRATEGY

要素名ストラテジーは、2 つの目的で使用されます。1 つ目は、オブジェクトを SOAP メッセージにマーシャリングする際に、指定のオブジェクトと soap アクションの xml 要素名を検索することです。2 つ目は、特定の soap フォールト名に対して Exception クラスを見つけることです。

ストラテジー	用途
QNameStrategy	インスタンス化時に設定された固定 qName を使用します。例外検索はサポートされていません。
TypeNameStrategy	指定のタイプの @XMLType アノテーションにある名前および namespace を使用します。名前空間が設定されていない場合は、package-info が使用されます。例外検索はサポートされていません。
ServiceInterfaceStrategy	webservice インターフェースからの情報を使用してタイプ名を判別し、SOAP 障害の例外クラスを見つけます。

cxf-codegen または同様のツールで Web サービススタブコードを生成した場合は、ServiceInterfaceStrategy を使用することが推奨されます。アノテーションが付けられたサービスインターフェースがない場合は、QNameStrategy または TypeNameStrategy を使用する必要があります。

290.3. JAVA DSL の使用

以下の例では、com.example.customerservice パッケージで設定された soap の名前付き DataFormat を使用して JAXBContext を初期化します。2 番目のパラメーターは ElementNameStrategy です。ルートは、通常のオブジェクトと例外をマーシャリングできます。（以下は SOAP Envelope をキューに送信するだけです。Web サービスプロバイダーは実際には SOAP 呼び出しが実際にキューをリッスンする必要があります。この場合、SOAP リクエストは 1 通りです。要求応答が必要な場合は、次の例を確認してください。

```
SoapJaxbDataFormat soap = new SoapJaxbDataFormat("com.example.customerservice", new
ServiceInterfaceStrategy(CustomerService.class));
from("direct:start")
  .marshal(soap)
  .to("jms:myQueue");
```

ヒント

「SOAP dataformat inherits from the JAXB dataformat inherits from the JAXB dataformat applys as the most settings applies here」も参照 してください。

290.3.1. SOAP 1.2 の使用

Camel 2.11 から利用可能

```
SoapJaxbDataFormat soap = new SoapJaxbDataFormat("com.example.customerservice", new
ServiceInterfaceStrategy(CustomerService.class));
soap.setVersion("1.2");
from("direct:start")
  .marshal(soap)
  .to("jms:myQueue");
```

XML DSL を使用する場合、<soapjaxb> 要素に version 属性を設定できます。

```
    <!-- Defining a ServiceInterfaceStrategy for retrieving the element name when marshalling --
>
    <bean id="myNameStrategy"
class="org.apache.camel.dataformat.soap.name.ServiceInterfaceStrategy">
        <constructor-arg value="com.example.customerservice.CustomerService"/>
    <constructor-arg value="true"/>
    </bean>
```

Camel ルート

```
<route>
  <from uri="direct:start"/>
  <marshal>
    <soapjaxb contentPath="com.example.customerservice" version="1.2"
elementNameStrategyRef="myNameStrategy"/>
  </marshal>
  <to uri="jms:myQueue"/>
</route>
```

290.4. マルチパートメッセージ

Camel 2.8.1 から利用可能

マルチパート SOAP メッセージは、ServiceInterfaceStrategy によってサポートされます。ServiceInterfaceStrategy は、JAX-WS 2.2 に従ってアノテーションが付けられ、Document Bare スタイルの要件を満たすサービスインターフェース定義で初期化する必要があります。target メソッドは、JAX-WS 仕様に従って以下の基準を満たす必要があります。1 に おい て、ヘッダー以外の戻り値のタイプが void 以外のタイプの場合は、最も 1 つまたは in-out または out non-header パラメーターが必要です。3 は、戻り値のタイプが void である場合には、ほとんどの in/out パラメーターまたは out non-header パラメーターが必要です。

ServiceInterfaceStrategy は、マッピングストラテジーがリクエストパラメーターまたは応答パラ

メーターに適用されるかどうかを示すブール値パラメーターを使用して初期化する必要があります。

```
ServiceInterfaceStrategy strat =  new
ServiceInterfaceStrategy(com.example.customerservice.multipart.MultiPartCustomerService.c
lass, true);
SoapJaxbDataFormat soapDataFormat = new
SoapJaxbDataFormat("com.example.customerservice.multipart", strat);
```

290.4.1. マルチパート要求

マルチパートリクエストのペイロードパラメーターは、ターゲット操作の署名を反映する BeanInvocation オブジェクトを使用して開始されます。camel-soap DataFormat は、marshal（）プロセッサーが呼び出されると、BeanInvocation の内容を SOAP ヘッダーおよびボディーのフィールドにマッピングします。

```
BeanInvocation beanInvocation = new BeanInvocation();

// Identify the target method
beanInvocation.setMethod(MultiPartCustomerService.class.getMethod("getCustomersByNam
e",
    GetCustomersByName.class, com.example.customerservice.multipart.Product.class));

// Populate the method arguments
GetCustomersByName getCustomersByName = new GetCustomersByName();
getCustomersByName.setName("Dr. Multipart");

Product product = new Product();
product.setName("Multiuse Product");
product.setDescription("Useful for lots of things.");

Object[] args = new Object[] {getCustomersByName, product};

// Add the arguments to the bean invocation
beanInvocation.setArgs(args);

// Set the bean invocation object as the message body
exchange.getIn().setBody(beanInvocation);
```

290.4.2. マルチパートレスポンス

マルチパート soap 応答には、soap ボディーに要素が含まれる可能性があり、soap ヘッダーに 1 つ以上の要素があります。camel-soap DataFormat は soap ボディーの要素をアンマーシャリングし（存在する場合）、エクスチェンジの out メッセージのボディーに配置します。ヘッダー要素は JAXB マップされたオブジェクトタイプにマーシャリングされ ません。代わりに、これらの要素は camel out メッセージヘッダー org.apache.camel.dataformat.soap.UNMARSHALLED_HEADER_LIST に配置されます。要素は、ignoreJAXBElement プロパティーの設定に応じて、要素インスタンス値または JAXBElement 値として表示されます。このプロパティーは camel-jaxb から継承されます。

camel-soap DataFormate は、ignoreUnmarshalledHeaders の値を true に設定すると、ヘッダーのコンテンツを すべて に無視することもできます。

290.4.3. ホルダーオブジェクトマッピング

JAX-WS は、In/Out パラメーターおよび Out パラメーターに型パラメーター化された javax.xml.ws.Holder オブジェクトの使用を指定します。BeanInvocation を構築するときに Holder オブジェクトを使用するか、パラメーター化されたタイプのインスタンスを直接使用できます。camel-soap DataFormat は、Holder の値のクラスの JAXB マッピングに従って Holder の値をマーシャルします。'Holder オブジェクトのマッピングは、アンマーシャリング応答には提供されません。

290.5. 例

290.5.1. WebService クライアント

以下のルートは、リクエストのマーシャリングと、応答または障害のアンマーシャリングをサポートします。

```
String WS_URI = "cxf://http://myserver/customerservice?
serviceClass=com.example.customerservice&dataFormat=MESSAGE";
SoapJaxbDataFormat soapDF = new SoapJaxbDataFormat("com.example.customerservice",
new ServiceInterfaceStrategy(CustomerService.class));
from("direct:customerServiceClient")
  .onException(Exception.class)
    .handled(true)
    .unmarshal(soapDF)
  .end()
  .marshal(soapDF)
  .to(WS_URI)
  .unmarshal(soapDF);
```

以下のスニペットはサービスインターフェースのプロキシーを作成し、上記のルートに SOAP 呼び出しを行います。

```
import org.apache.camel.Endpoint;
import org.apache.camel.component.bean.ProxyHelper;
...

Endpoint startEndpoint = context.getEndpoint("direct:customerServiceClient");
ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
// CustomerService below is the service endpoint interface, *not* the javax.xml.ws.Service
subclass
CustomerService proxy = ProxyHelper.createProxy(startEndpoint, classLoader,
CustomerService.class);
GetCustomersByNameResponse response = proxy.getCustomersByName(new
GetCustomersByName());
```

290.5.2. WebService Server

以下のルートを使用して、jms キューの customerServiceQueue をリッスンし、CustomerServiceImpl クラスを使用してリクエストを処理する webservice サーバーをセットアップします。customerServiceImpl of course は CustomerService インターフェースを実装する必要があります。サーバークラスを直接インスタンス化する代わりに、Spring コンテキストで通常の Bean として定義できます。

```
SoapJaxbDataFormat soapDF = new SoapJaxbDataFormat("com.example.customerservice",
new ServiceInterfaceStrategy(CustomerService.class));
CustomerService serverBean = new CustomerServiceImpl();
from("jms://queue:customerServiceQueue")
  .onException(Exception.class)
    .handled(true)
    .marshal(soapDF)
  .end()
  .unmarshal(soapDF)
  .bean(serverBean)
  .marshal(soapDF);
```

290.6. 依存関係

camel ルートで SOAP データ形式を使用するには、以下の依存関係を pom に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-soap</artifactId>
  <version>2.3.0</version>
</dependency>
```

第291章 SOLR コンポーネント

Camel バージョン 2.9 で利用可能

Solr コンポーネントを使用すると、Apache Lucene Solr サーバー（SolrJ 3.5.0 ベース）とのインターフェースが可能です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-solr</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

291.1. URI 形式

注記： solrs および solrCloud は Camel 2.14 以降新しいバージョンに追加されます。

```
solr://host[:port]/solr?[options]
solrs://host[:port]/solr?[options]
solrCloud://host[:port]/solr?[options]
```

291.2. SOLR オプション

Solr コンポーネントにはオプションがありません。

Solr エンドポイントは、URI 構文を使用して設定します。

```
solr:url
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

291.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
url	solr サーバーに **必要な** ホスト名とポート		文字列

291.2.2. クエリーパラメーター（13 パラメーター）：

Name	説明	デフォルト	Type
allowCompression (producer)	サーバー側では、gzip または deflate のサポートして、		ブール値
connectionTimeout (producer)	基礎となる HttpConnectionManager の connectionTimeout		整数
defaultMaxConnectionsPer Host (producer)	基礎となる HttpConnectionManager の maxConnectionsPerHost		整数
followRedirects (producer)	Solr サーバーの取得にリダイレクトが使用されるかどうかを示します。		ブール値
maxRetries (producer)	一時的なエラー発生時に試行を試行する最大再試行数		整数
maxTotalConnections (producer)	基礎となる HttpConnectionManager の maxTotalConnection		整数
requestHandler (producer)	使用されるリクエストハンドラーの設定		文字列
soTimeout (producer)	基礎となる HttpConnectionManager の読み取りタイムアウト。これはクエリーには適していますが、インデックスには適さない場合があります。		整数
streamingQueueSize (producer)	StreamingUpdateSolrServer のキューサイズの設定	10	int
streamingThreadCount (producer)	StreamingUpdateSolrServer のスレッド数の設定	2	int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
コレクション (solrCloud)	solrCloud サーバーが使用できるコレクション名を設定します。		文字列

Name	説明	デフォルト	Type
zkHost (solrCloud)	solrCloud が使用できる ZooKeeper ホスト情報（zkhost=localhost:8123 など）を設定します。		文字列

291.3. メッセージ操作

現在、以下の Solr 操作がサポートされています。キーが「SolrOperation」のエクスチェンジヘッダーと、以下の値が以下のいずれかに設定されるだけです。一部の操作では、メッセージボディーも設定する必要があります。

- ***INSERT 操作では CommonsHttpSolrServerを使用します。***
- ***INSERT_STREAMING 操作は StreamingUpdateSolrServer (Camel 2.9.2)を使用します。***

操作	メッセージボディー	説明
INSERT/INSERT_STREAMING	該当なし	メッセージヘッダーを使用してインデックスを追加します（「SolrField.」のプレフィックスにする必要があります）。
INSERT/INSERT_STREAMING	ファイル	指定したファイルを使用してインデックスを追加します（ContentStreamUpdateRequest を使用）。
INSERT/INSERT_STREAMING	SolrInputDocument	**Camel 2.9.2** は指定の SolrInputDocument に基づいてインデックスを更新します。
INSERT/INSERT_STREAMING	String XML	**Camel 2.9.2** は指定の XML に基づいてインデックスを更新します（SolrInputDocument 形式に従う必要があります）。

操作	メッセージボディー	説明
ADD_BEAN	Beanインスタンス	アノテーション付き Beanの値に基づいてインデックスを追加します。
ADD_BEANS	collection<bean>	Camel 2.15 がアノテーション付き Beanのコレクションに基づいてインデックスを追加
DELETE_BY_ID	削除するインデックス ID	ID 別にレコードを削除します。
DELETE_BY_QUERY	クエリー文字列	クエリーでレコードを削除します。
COMMIT	該当なし	保留中のインデックスの変更に対してコミットを実行します。
ROLLBACK	該当なし	保留中のインデックスの変更でロールバックを実行します。
最適化	該当なし	保留中のインデックス変更に対してコミットを実行し、最適化コマンドを実行します。

291.4. 例

以下は、単純な INSERT、DELETE、および COMMIT の例になります。

```
from("direct:insert")
    .setHeader(SolrConstants.OPERATION, constant(SolrConstants.OPERATION_INSERT))
    .setHeader(SolrConstants.FIELD + "id", body())
    .to("solr://localhost:8983/solr");

from("direct:delete")
    .setHeader(SolrConstants.OPERATION,
constant(SolrConstants.OPERATION_DELETE_BY_ID))
    .to("solr://localhost:8983/solr");

from("direct:commit")
    .setHeader(SolrConstants.OPERATION, constant(SolrConstants.OPERATION_COMMIT))
    .to("solr://localhost:8983/solr");
```

```
<route>
    <from uri="direct:insert"/>
    <setHeader headerName="SolrOperation">
        <constant>INSERT</constant>
    </setHeader>
    <setHeader headerName="SolrField.id">
        <simple>${body}</simple>
    </setHeader>
    <to uri="solr://localhost:8983/solr"/>
</route>
<route>
    <from uri="direct:delete"/>
    <setHeader headerName="SolrOperation">
        <constant>DELETE_BY_ID</constant>
    </setHeader>
    <to uri="solr://localhost:8983/solr"/>
</route>
<route>
    <from uri="direct:commit"/>
    <setHeader headerName="SolrOperation">
        <constant>COMMIT</constant>
    </setHeader>
    <to uri="solr://localhost:8983/solr"/>
</route>
```

クライアントでは、ボディーメッセージをルートを挿入または削除してコミットルートを呼び出すだけです。

```
template.sendBody("direct:insert", "1234");
template.sendBody("direct:commit", null);
template.sendBody("direct:delete", "1234");
template.sendBody("direct:commit", null);
```

291.5. SOLR のクエリー

現在、このコンポーネントはデータのネイティブクエリーをサポートしていません（後で追加することもできます）。現時点では、以下のように HTTP を使用して Solr にクエリーできます。

```
//define the route to perform a basic query
from("direct:query")
    .recipientList(simple("http://localhost:8983/solr/select/?q=${body}"))
    .convertBodyTo(String.class);
...
//query for an id of '1234' (url encoded)
String responseXml = (String) template.requestBody("direct:query", "id%3A1234");
```

詳細情報は、これらのリソース...

Solr Query Tutorial

Solr クエリー構文

291.6. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

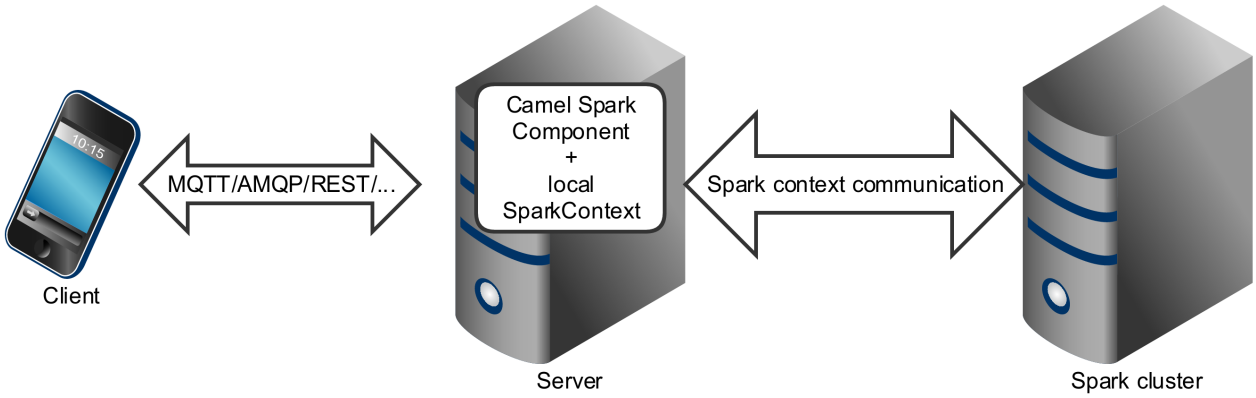
第292章 APACHE SPARK コンポーネント

Camel バージョン 2.17 から利用可能

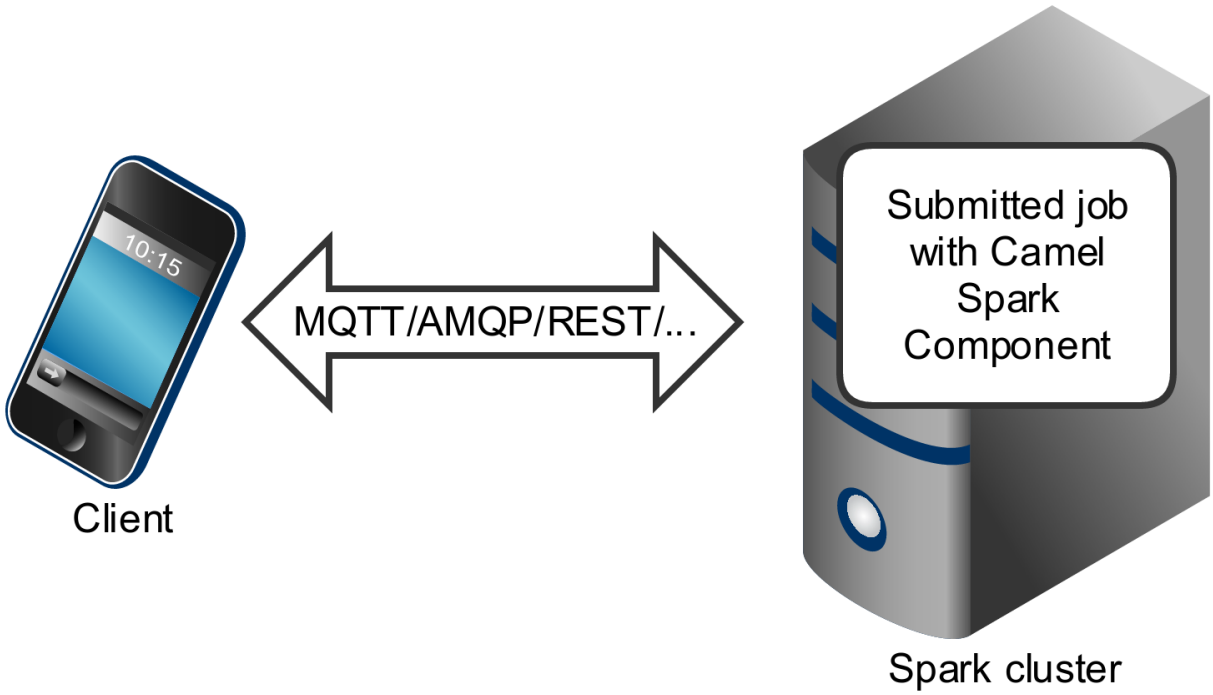
本ドキュメントページでは、Apache Camel の Apache Spark コンポーネントについて説明します。Camel との Spark インテグレーションの主な目的は、Camel コネクターと Spark タスク間のブリッジを提供することです。特定の Camel コネクターは、さまざまなトランスポートからメッセージをルーティングし、実行するタスクを動的に選択し、そのタスクの入力データとして受信メッセージを使用し、最後に実行結果を Camel パイプラインに戻す方法を提供します。

292.1. サポート対象のアーキテクチャースタイル

Spark コンポーネントは、アプリケーションサーバーにデプロイされているドライバーアプリケーションとして使用できます（または fat jar として実行）。



Spark コンポーネントは、Spark クラスターに直接ジョブとして送信することもできます。



Spark コンポーネントは主に、Spark クラスターと他のエンドポイント間のブリッジとして機能する長時間実行されるジョブ として機能するように設計されていますが、1 回実行(run -once)の短いジョブとして使用できます。

292.2. OSGI サーバーでの SPARK の実行

現在、Spark コンポーネントは OSGi コンテナーでの実行をサポートしていません。Spark は fat jar として実行されるように設計されており、通常はジョブとしてクラスターに送信されます。OSGi サーバーで Spark を実行している理由は、少なくとも困難であり、Camel ではサポートされていません。

292.3. URI 形式

現在、Spark コンポーネントはプロデューサーのみをサポートします。Spark ジョブを呼び出して結果を返すことを目的としています。RDD、データフレーム、または Hive SQL ジョブを呼び出します。

Spark URI 形式

```
spark:{rdd|dataframe|hive}
```

292.3.1. Spark オプション

Apache Spark コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
rdd (producer)	計算対象である RDD。		JavaRDDLike
rddCallback (producer)	RDD に対するアクションの実行関数。		RddCallback
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Apache Spark エンドポイントは、URI 構文を使用して設定します。

```
spark:endpointType
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

292.3.2. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
endpointType	エンドポイントに **必要な** タイプ（rdd、dataframe、hive）。		EndpointType

292.3.3. クエリーパラメーター（6 パラメーター）：

Name	説明	デフォルト	Type
collect（プロデューサー）	結果を収集またはカウントするかどうかを示します。	true	boolean
dataFrame (producer)	計算先となる DataFrame。		DataFrame
dataFrameCallback (producer)	DataFrame に対するアクション実行関数。		DataFrameCallback

Name	説明	デフォルト	Type
rdd (producer)	計算対象である RDD。		JavaRDDLike
rddCallback (producer)	RDD に対するアクションの実行関数。		RddCallback
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

 # RDD jobs

RDD ジョブを呼び出すには、以下の URI を使用します。

Spark RDD プロデューサー

```
spark:rdd?rdd=#testFileRdd&rddCallback=#transformation
```

rdd オプションは、Camel レジストリーから RDD インスタンスの名前（ org.apache.spark.api.java.JavaRDDLikeのサブクラス）を参照しますが、rddCallback は org.apache.camel.component.spark.RddCallback インターフェース（レジストリーとも呼ばれます）の実装を参照します。RDD コールバックは、指定の RDD に対して受信メッセージを適用する単一の方法を提供します。コールバック計算の結果は、エクスチェンジのボディーとして保存されます。

Spark RDD コールバック

```
public interface RddCallback<T> {
    T onRdd(JavaRDDLike rdd, Object... payloads);
}
```

以下のスニペットは、メッセージをジョブへの入力として送信し、結果を返す方法を示しています。

スパークジョブの呼び出し

```
String pattern = "job input";
long linesCount = producerTemplate.requestBody("spark:rdd?
rdd=#myRdd&rddCallback=#countLinesContaining", pattern, long.class);
```

Spring Bean として登録された上記のスニペットの RDD コールバックは、以下のようになります。

Spark RDD コールバック

```
@Bean
RddCallback<Long> countLinesContaining() {
   return new RddCallback<Long>() {
      Long onRdd(JavaRDDLike rdd, Object... payloads) {
         String pattern = (String) payloads[0];
         return rdd.filter({line -> line.contains(pattern)}).count();
      }
   }
}
```

Spring の RDD 定義は以下のようになります。

Spark RDD の定義

```
@Bean
JavaRDDLike myRdd(JavaSparkContext sparkContext) {
  return sparkContext.textFile("testrdd.txt");
}
```

292.3.4. void RDD コールバック

RDD コールバックが Camel パイプラインに値を返しない場合は、null 値を返すか、VoidRddCallback ベースクラスを使用できます。

Spark RDD の定義

```
@Bean
RddCallback<Void> rddCallback() {
  return new VoidRddCallback() {
      @Override
      public void doOnRdd(JavaRDDLike rdd, Object... payloads) {
         rdd.saveAsTextFile(output.getAbsolutePath());
```

```
        }
    };
}
```

292.3.5. RDD コールバックの変換

RDD コールバックに送信される入力データのタイプが分かる場合は、 ConvertingRddCallback を使用して、Camel が受信メッセージを自動的に変換してからコールバックに挿入することができます。

Spark RDD の定義

```
@Bean
RddCallback<Long> rddCallback(CamelContext context) {
  return new ConvertingRddCallback<Long>(context, int.class, int.class) {
        @Override
        public Long doOnRdd(JavaRDDLike rdd, Object... payloads) {
            return rdd.count() * (int) payloads[0] * (int) payloads[1];
        }
    };
}
```

292.3.6. アノテーション付き RDD コールバック

おそらく RDD コールバックを使用する最も簡単な方法は、 @RddCallback アノテーションが付けられたメソッドを持つクラスを提供することです。

アノテーション付き RDD コールバック定義

```
import static
org.apache.camel.component.spark.annotations.AnnotatedRddCallback.annotatedRddCallbac
k;

@Bean
RddCallback<Long> rddCallback() {
    return annotatedRddCallback(new MyTransformation());
}

...

import org.apache.camel.component.spark.annotation.RddCallback;

public class MyTransformation {

    @RddCallback
    long countLines(JavaRDD<String> textFile, int first, int second) {
```

```
        return textFile.count() * first * second;
    }

}
```

アノテーションが付けられた RDD コールバックファクトリーメソッドに CamelContext を渡すと、作成されたコールバックは、アノテーションが付けられたメソッドのパラメーターに一致するように受信ペイロードを変換できます。

アノテーションが付けられた RDD コールバックのボディー変換

```
import static
org.apache.camel.component.spark.annotations.AnnotatedRddCallback.annotatedRddCallbac
k;

@Bean
RddCallback<Long> rddCallback(CamelContext camelContext) {
    return annotatedRddCallback(new MyTransformation(), camelContext);
}

...


import org.apache.camel.component.spark.annotation.RddCallback;

public class MyTransformation {

    @RddCallback
    long countLines(JavaRDD<String> textFile, int first, int second) {
        return textFile.count() * first * second;
    }

}

...

// Convert String "10" to integer
long result = producerTemplate.requestBody("spark:rdd?
rdd=#rdd&rddCallback=#rddCallback" Arrays.asList(10, "10"), long.class);
```

292.4. DATAFRAME ジョブ

RDDs Spark コンポーネントを使用する代わりに、DataFrame と連携することもできます。

DataFrame ジョブを呼び出すには、以下の URI を使用します。

Spark RDD プロデューサー

```
spark:dataframe?dataFrame=#testDataFrame&dataFrameCallback=#transformation
```

dataFrame オプションは、Camel レジストリーからの DataFrame インスタンスの名前（org.apache.spark.sql.DataFrame のインスタンス）を参照しますが、dataFrameCallback は org.apache.camel.component.spark.DataFrameCallback インターフェースの実装を指します（レジストリーとも呼ばれます）。DataFrame コールバックは、指定の DataFrame に受信メッセージを適用する単一の方法を提供します。コールバック計算の結果は、エクスチェンジのボディーとして保存されます。

Spark RDD コールバック

```
public interface DataFrameCallback<T> {
    T onDataFrame(DataFrame dataFrame, Object... payloads);
}
```

以下のスニペットは、メッセージをジョブへの入力として送信し、結果を返す方法を示しています。

スパークジョブの呼び出し

```
String model = "Micra";
long linesCount = producerTemplate.requestBody("spark:dataFrame?
dataFrame=#cars&dataFrameCallback=#findCarWithModel", model, long.class);
```

Spring Bean として登録された上記のスニペットの DataFrame コールバックは、以下のようになります。

Spark RDD コールバック

```
@Bean
RddCallback<Long> findCarWithModel() {
    return new DataFrameCallback<Long>() {
        @Override
        public Long onDataFrame(DataFrame dataFrame, Object... payloads) {
            String model = (String) payloads[0];
            return dataFrame.where(dataFrame.col("model").eqNullSafe(model)).count();
```

```
        }
    };
}
```

Spring の DataFrame 定義は以下のようになります。

Spark RDD の定義

```
@Bean
DataFrame cars(HiveContext hiveContext) {
    DataFrame jsonCars = hiveContext.read().json("/var/data/cars.json");
    jsonCars.registerTempTable("cars");
    return jsonCars;
}
```

292.5. HIVE ジョブ

RDD または DataFrame Spark コンポーネントを使用する代わりに、Hive SQL クエリーをペイロードとして受信することもできます。 Hive クエリーを Spark コンポーネントに送信するには、以下の URI を使用します。

Spark RDD プロデューサー

```
spark:hive
```

以下のスニペットは、メッセージをジョブへの入力として送信し、結果を返す方法を示しています。

スパークジョブの呼び出し

```
long carsCount = template.requestBody("spark:hive?collect=false", "SELECT * FROM cars",
Long.class);
List<Row> cars = template.requestBody("spark:hive", "SELECT * FROM cars", List.class);
```

クエリーを実行するテーブルは、クエリーを実行する前に HiveContext に登録する必要があります。たとえば、Spring では以下のように登録できます。

Spark RDD の定義

```
@Bean
DataFrame cars(HiveContext hiveContext) {
    DataFrame jsonCars = hiveContext.read().json("/var/data/cars.json");
    jsonCars.registerTempTable("cars");
    return jsonCars;
}
```

292.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第293章 SPARK REST コンポーネント

Camel バージョン 2.14 から利用可能

Spark-rest コンポーネントでは、Rest DSL を使用して Spark REST Java ライブラリーを使用して REST エンドポイントを定義できます。

INFO: Spark Java requires Java 8 runtime.

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-spark-rest</artifactId>
    <version>${camel-version}</version>
  </dependency>
```

293.1. URI 形式

```
spark-rest://verb:path?[options]
```

293.2. URI オプション

Spark Rest コンポーネントは、以下に示す 12 個のオプションをサポートします。

Name	説明	デフォルト	Type
ポート（コンシューマー）	ポート番号。デフォルトでは 4567 を使用します。	4567	int
ipAddress (consumer)	Spark がリッスンする IP アドレスを設定します。呼び出されない場合、デフォルトのアドレスは '0.0.0.0' になります。	0.0.0.0	文字列
minThreads (advanced)	Spark スレッドプールの最小スレッド数（グローバルに共有）		int
maxThreads (advanced)	Spark スレッドプールの最大スレッド数（グローバルに共有）		int

Name	説明	デフォルト	Type
timeOutMillis (advanced)	スレッドアイドルタイムアウト（ミリ秒単位）では、長期間アイドル状態であったスレッドがスレッドプールから終了される。		int
keystoreFile (security)	キーストアファイルを使用するようにセキュアな接続を設定します。		文字列
keystorePassword (security)	キーストアパスワードを使用するように接続をセキュアにする		文字列
truststoreFile (security)	トラストストアファイルを使用するようにセキュアな接続を設定します。		文字列
truststorePassword (security)	トラストストアのパスワードを使用するようにセキュアな接続を設定します。		文字列
sparkConfiguration (advanced)	共有 SparkConfiguration の使用		SparkConfiguration
sparkBinding (advanced)	カスタム SparkBinding を使用して Camel メッセージへ/からマッピングします。		SparkBinding
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Spark Rest エンドポイントは URI 構文を使用して設定します。

```
spark-rest:verb:path
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

293.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
verb	**必要な** get、post、PUT、patch、delete、head、trace、connect、または options が必要です。		文字列

Name	説明	デフォルト	Type
path	Spark 構文をサポートするコンテンツパスが **必要です**。		文字列

293.2.2. クエリーパラメーター（11 パラメーター）：

Name	説明	デフォルト	Type
Accept (consumer)	'text/xml' や 'application/json' などの型を受け入れます。デフォルトでは、あらゆる種類のタイプを受け入れます。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
disableStreamCache (consumer)	Spark HttpRequestgetContent（）からの raw 入力ストリームがキャッシュされているかどうかを決定します（Camel はストリームを軽量メモリーベースのストリームキャッシュに読み込みます）。デフォルトでは、Camel は Netty 入力ストリームをキャッシュして、複数回ロードし、Camel がストリームからすべてのデータを取得できるようにします。ただし、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。たとえば、ファイルまたは他の永続ストアに直接ストリーミングする場合などに、raw ストリームにアクセスする必要がある場合などにこのオプションを true に設定します。このオプションを有効にすると、Netty ストリームが追加設定なしで複数回読み取ることができないため、Spark raw ストリームでリーダーインデックスを手動でリセットする必要があります。	false	boolean

Name	説明	デフォルト	Type
mapHeaders (consumer)	このオプションが有効な場合には、Spark から Camel Message へのバインディング時にヘッダーもマッピングされます（たとえば、ヘッダーとして Camel Message に追加されます）。このオプションを無効にして無効にすることができます。ヘッダーには、Spark HTTP リクエストインスタンスを返す getRequest（）メソッドを使用して org.apache.camel.component.sparkrest.SparkMessage メッセージからアクセスできます。	true	boolean
transferException (consumer)	有効で Exchange がコンシューマー側で処理に失敗し、発生した例外が application/x-java-serialized-object のコンテンツタイプとして応答でシリアライズされたかどうか。これは、デフォルトではオフになっています。これを有効にすると、Java は受信データをリクエストから Java へデシリアライズし、潜在的なセキュリティーリスクとなる可能性があることに注意してください。	false	boolean
urlDecodeHeaders (consumer)	このオプションが有効な場合には、Spark から Camel Message へのバインディング時にヘッダー値がデコードされます（例： %20 はスペース文字です）。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
matchOnUriPrefix (advanced)	完全一致がない場合、コンシューマーが URI プレフィックスに一致することでターゲットコンシューマーの検索を試行するかどうか。	false	boolean
sparkBinding (advanced)	カスタム SparkBinding を使用して Camel メッセージへ/からマッピングします。		SparkBinding
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

293.3. SPARK 構文を使用したパス

path オプションは Spark REST 構文で定義します。ここで、パラメーターとスプラー(slat)のサポートを使用して REST コンテキストパスを定義します。詳細は、Spark Java Route ドキュメント を参照してください。

以下は、固定パスを使用した Camel ルートです。

```
from("spark-rest:get:hello")
  .transform().constant("Bye World");
```

以下のルートは、キー「me」を持つ Camel ヘッダーにマッピングされるパラメーターを使用します。

```
from("spark-rest:get:hello/:me")
  .transform().simple("Bye ${header.me}");
```

293.4. CAMEL メッセージへのマッピング

Spark Request オブジェクトは、getRequest メソッドを使用して未加工の Spark 要求にアクセスできる org.apache.camel.component.sparkrest.SparkMessage として Camel Message にマッピングされます。デフォルトでは、Spark ボディーは Camel メッセージボディーにマッピングされ、HTTP ヘッダー / Spark パラメーターは Camel Message ヘッダーにマッピングされます。Spark splat 構文には特別なサポートがあり、これはキースプレープで Camel メッセージヘッダーにマッピングされます。

たとえば、以下のルートはコンテキストパスで Spark splat（アスタリスク記号）を使用します。コンテキストパスでは、Simple 言語のヘッダーとしてアクセスして応答メッセージを構築します。

```
from("spark-rest:get:/hello/*/to/*")
  .transform().simple("Bye big ${header.splat[1]} from ${header.splat[0]}");
```

293.5. REST DSL

Apache Camel は、REST サービスを優れた REST スタイルで定義できる新しい Rest DSL を提供します。たとえば、以下のように REST hello サービスを定義できます。

```
return new RouteBuilder() {
   @Override
   public void configure() throws Exception {
        rest("/hello/{me}").get()
```

```
            .route().transform().simple("Bye ${header.me}");
    }
};
```

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <rest uri="/hello/{me}">
    <get>
      <route>
        <transform>
          <simple>Bye ${header.me}</simple>
        </transform>
      </route>
    </get>
  </rest>
</camelContext>
```

詳細は「Rest DSL」を参照してください。

293.6. その他の例

Apache Camel ディストリビューションには camel-example-spark-rest-tomcat のサンプルがあります。これは、Apache Tomcat または同様の Web コンテナーに camel-spark-rest を使用する方法を実証します。

第294章 SPEL LANGUAGE

Camel バージョン 2.7 で利用可能

Camel では、SpEL(Spring Expression Language) を DSL または XML 設定で Expression または Predicate として使用できます。

注記

Spring ランタイムで SpEL を使用することを推奨します。しかし、Camel 2.21 以降では、他のランタイムに SpEL を使用できます（Spring ランタイムで実行されていない場合、SpEL は機能できない可能性があります）。

294.1. 変数

以下の変数は、SpEL で記述された式および述語で利用できます。

変数	型	説明
this	エクスチェンジ	Exchange はルートオブジェクトです。
exchange	エクスチェンジ	Exchange オブジェクト
exception	Throwable	エクスチェンジの例外 (ある場合)
exchangeId	文字列	エクスチェンジ ID
fault	メッセージ	Fault メッセージ（ある場合）
ボディー	オブジェクト	IN メッセージのボディー。
request	メッセージ	exchange.in メッセージ

変数	型	説明
response	メッセージ	exchange.out メッセージ（ある場合）
properties	マップ	エクスチェンジプロパティー
property(name)	オブジェクト	指定の名前のプロパティー
property(name, type)	Type	指定の名前のプロパティー（指定のタイプ）

294.2. オプション

SpEL 言語は、以下に示す 1 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
trim	**true**	**ブール値**	値をトリミングして先頭および末尾の空白と改行を削除するかどうか。

294.3. サンプル

294.3.1. 式のテンプレート

式のテンプレートが有効にされているため、SpEL 式を #{ } 区切り文字で囲む必要があります。これにより、SpEL 式と正規表現を組み合わせて、非常に軽量のテンプレート言語として使用できます。

たとえば、以下のルートを作成する場合は、以下のようになります。

```
from("direct:example")
    .setBody(spel("Hello #{request.body}! What a beautiful #{request.headers['dayOrNight']}"))
    .to("mock:result");
```

上記のルートでは、spel は、org.apache.camel.language.spelExpression.spelExpression.spel からインポートする必要のある static メソッドです。これは、spel を、setBody メソッドにパラメー

ターとして渡される式として使用するためです。Fluent API を使用する場合は、代わりにこれを行うことができます。

```
from("direct:example")
    .setBody().spel("Hello #{request.body}! What a beautiful #{request.headers['dayOrNight']}")
    .to("mock:result");
```

setBody（） メソッドの spel メソッドを使用することに注意してください。また、org.apache.camel.language.spelExpression.spelExpression.spelExpression.spel から spel メソッドを静的にインポートする必要はありません。

そして、本文に文字列 "gitops" と、値が "day" のヘッダー "dayOrNight" を持つメッセージを送信します。

```
template.sendBodyAndHeader("direct:example", "World", "dayOrNight", "day");
```

mock:result の出力は "Hello World! になります。異常な日"

294.3.2. Bean インテグレーション

SpEL 式の Registry（主に ApplicationContext）で定義された Bean を参照できます。たとえば、ApplicationContext に「foo」という名前の Bean がある場合は、以下のようにこの Bean で「bar」メソッドを呼び出すことができます。

```
#{@foo.bar == 'xyz'}
```

294.3.3. エンタープライズ統合パターンにおける SpEL

SpEL を Recipient List の式または Message Filter 内の述語として使用することができます。

```
<route>
  <from uri="direct:foo"/>
  <filter>
    <spel>#{request.headers['foo'] == 'bar'}</spel>
    <to uri="direct:bar"/>
  </filter>
</route>
```

そして、Java DSL で同等です。

```
from("direct:foo")
    .filter().spel("#{request.headers['foo'] == 'bar'}")
    .to("direct:bar");
```

294.4. 外部リソースからのスクリプトの読み込み

Camel 2.11 から利用可能

スクリプトを外部化して、「classpath:」、"file:"、または "http: " などのリソースから Camel に読み込むことができます。
これは、「 resource:scheme:location」 構文を使用して行われます。たとえば、実行可能なクラスパスのファイルを参照します。

```
.setHeader("myHeader").spel("resource:classpath:myspel.txt")
```

第295章 SPLUNK コンポーネント

Camel バージョン 2.13 から利用可能

Splunk コンポーネントは、Splunk が提供する クライアント api を使用して Splunk へのアクセスを提供し、Splunk でイベントを公開および検索できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-splunk</artifactId>
    <version>${camel-version}</version>
  </dependency>
```

295.1. URI 形式

```
splunk://[endpoint]?[options]
```

295.2. プロデューサーエンドポイント：

エンドポイント	説明
stream	名前付きのインデックスまたは指定されていない場合は、データを名前付きインデックスにストリーミングします。ストリームモードを使用する場合は、イベントがインデックスに到達する前に、Splunk が内部バッファー（1MB など）を持っていることに注意してください。リアルタイムが必要な場合は、送信モードまたは tcp モードを使用することが推奨されます。
submit	送信モード。Splunk rest api を使用してイベントを名前付きインデックスに公開し、指定されていない場合はデフォルトにします。
tcp	TCP モード。データを tcp ポートにストリーミングし、Splunk でオープンレシーバーポートが必要です。

イベントをパブリッシュする場合、メッセージボディーには SplunkEvent が含まれている必要があります。 メッセージボディーのコメントを参照してください。

例

```
from("direct:start").convertBodyTo(SplunkEvent.class)
    .to("splunk://submit?
username=user&password=123&index=myindex&sourceType=someSourceType&source=my
Source")...
```

この例では、SplunkEvent クラスに変換するコンバーターが必要です。

295.3. コンシューマーエンドポイント：

エンドポイント	説明
normal	通常の検索を実行し、search オプションに検索クエリーが必要になります。
savedsearch	splunk に保存されている検索クエリーに基づいて検索を実行し、保存されたSearch オプションにクエリーの名前が必要です。

例

```
from("splunk://normal?delay=5s&username=user&password=123&initEarliestTime=-
10s&search=search index=myindex sourcetype=someSourcetype")
    .to("direct:search-result");
```

camel-splunk は、本文に SplunkEvent を使用して、検索結果ごとにルートエクスチェンジを作成します。

295.4. URI オプション

Splunk コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
splunkConfiguration Factory (advanced)	SplunkConfigurationFactory を使用するには、以下を実行します。		SplunkConfiguration Factory
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Name	説明	デフォルト	Type

Splunk エンドポイントは、URI 構文を使用して設定します。

```
splunk:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

295.4.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	***必須*** 名には目的がありません		文字列

295.4.2. クエリーパラメーター（42 パラメーター）：

Name	説明	デフォルト	Type
app (common)	Splunk アプリケーション		文字列
connectionTimeout (common)	Splunk サーバーへの接続時の MS のタイムアウト	5000	int
ホスト （共通）	Splunk ホスト。	localhost	文字列
owner (common)	Splunk 所有者		文字列
ポート （共通）	Splunk ポート	8089	int
スキーム (common)	Splunk スキーム	https	文字列

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
count （コンシューマー）	返すエンティティーの最大数を示す数字。		int
earliestTime (consumer)	検索時間枠の最も早い時間。		文字列
initEarliestTime (consumer)	最初の検索の初期開始オフセット		文字列
latestTime (consumer)	検索時間枠の最新時間。		文字列
savedSearch (consumer)	実行する Splunk に保存されているクエリーの名前		文字列
検索 （コンシューマー）	実行する Splunk クエリー		文字列
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
streaming (consumer)	ストリーミングモードを設定します。streaming モードは、バッチではなく、受信時にエクスチェンジを送信します。		ブール値
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
eventHost (producer)	デフォルトの Splunk イベントホストフィールドを上書きします。		文字列
index (producer)	書き込む Splunk インデックス		文字列
raw (producer)	ペイロードが raw を挿入するかどうか。	false	boolean
ソース （プロデューサー）	Splunk ソース引数		文字列
sourceType (producer)	Splunk sourcetype 引数		文字列
tcpReceiverPort (producer)	Splunk tcp receiver ポート		int
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long

Name	説明	デフォルト	Type
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
パスワード （セキュリティー）	Splunk のパスワード		文字列
sslProtocol (security)	使用する ssl プロトコルを設定します。	TLSv1.2	SSLSecurityProtocol
ユーザー名 （セキュリティー）	Splunk のユーザー名		文字列

Name	説明	デフォルト	Type
useSunHttpsHandler (security)	sun.net www.protocol.https.Handler Https ハンドラーを使用して Splunk 接続を確立します。これは、アプリケーションサーバーで実行し、アプリケーションの https 処理を回避する場合に役立ちます。	false	boolean

295.5. メッセージボディー

Splunk は、キーと値のペアのデータで動作します。SplunkEvent クラスは、このようなデータのプレースホルダーであり、プロデューサーのメッセージボディーに置く必要があります。同様に、コンシューマーの検索結果ごとにボディーで返されます。

Camel 2.16.0 以降では、プロデューサーエンドポイントに raw オプションを設定して、raw データを Splunk に送信できます。これは、Splunk がサポートをビルドしている eg. json/xml などのペイロードに役立ちます。

295.6. ユースケース

Twitter で music でのツイートを検索し、イベントを Splunk に公開

```
    from("twitter://search?
type=polling&keywords=music&delay=10&consumerKey=abc&consumerSecret=def&accessT
oken=hij&accessTokenSecret=xxx")
        .convertBodyTo(SplunkEvent.class)
        .to("splunk://submit?username=foo&password=bar&index=camel-
tweets&sourceType=twitter&source=music-tweets");
```

Tweet を SplunkEvent に変換するには、以下のようなコンバーターを使用できます。

```
@Converter
public class Tweet2SplunkEvent {
   @Converter
   public static SplunkEvent convertTweet(Status status) {
      SplunkEvent data = new SplunkEvent("twitter-message", null);
      //data.addPair("source", status.getSource());
      data.addPair("from_user", status.getUser().getScreenName());
      data.addPair("in_reply_to", status.getInReplyToScreenName());
      data.addPair(SplunkEvent.COMMON_START_TIME, status.getCreatedAt());
      data.addPair(SplunkEvent.COMMON_EVENT_ID, status.getId());
      data.addPair("text", status.getText());
      data.addPair("retweet_count", status.getRetweetCount());
      if (status.getPlace() != null) {
```

```
            data.addPair("place_country", status.getPlace().getCountry());
            data.addPair("place_name", status.getPlace().getName());
            data.addPair("place_street", status.getPlace().getStreetAddress());
        }
        if (status.getGeoLocation() != null) {
            data.addPair("geo_latitude", status.getGeoLocation().getLatitude());
            data.addPair("geo_longitude", status.getGeoLocation().getLongitude());
        }
        return data;
    }
}
```

Splunk でツイートの検索

```
    from("splunk://normal?username=foo&password=bar&initEarliestTime=-
2m&search=search index=camel-tweets sourcetype=twitter")
        .log("${body}");
```

295.7. その他のコメント

Splunk には、これを分析して表示するために事前にビルドされたアプリケーションと共に機械生成データを利用するさまざまなオプションがあります。
たとえば、jmx アプリケーションなどを使用して、jmx 属性（例： ルートおよび jvm メトリクスを Splunk に公開）し、これをダッシュボードに表示できます。

295.8. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第296章 SPRING サポート

Apache Camel は、複数の方法で Spring Framework と適切に動作するように設計されています。

- Camel は JMS や JPAなどのコンポーネントで Spring Transactions をデフォルトのトランザクション処理として使用します。
- Camel は Spring 2 XML 処理と Xml の設定と連携する
- Camel Spring XML Schema's are defined in Xml Reference（Camel Spring XML スキーマの Xml Referenceで定義されている）
- Camel は強力なバージョンの Spring Remoting をサポートします。これは、クライアントとサーバー間の強力なルーティングと、トランスポートで利用可能なコンポーネントすべての使用をサポートします。
- Camel は、Spring ApplicationContext で定義された Bean と強力な Bean 統合を提供します。
- Camel は、Spring リソース用の Type Converter サポートの提供など、さまざまな Spring ヘルパークラスと統合します。
- Spring はコンポーネントインスタンスまたは CamelContext インスタンス自体を依存関係化し、Spring Bean をコンポーネントおよびエンドポイントとして自動公開します。
- Spring テストフレームワークを再利用し、エンタープライズ統合パターンと Camel の強力な モック エンドポイントおよびテストエンドポイントを使用して、ユニットテストと統合 テスト を簡素化できます。
- Camel 2.15 以降では、Camel は camel-spring-boot コンポーネントを使用して Spring Boot をサポートします。

296.1. SPRING を使用した CAMELCONTEXT の設定

CamelContextFactoryBean を使用すると、Spring.xml 内で CamelContext を設定できます。これにより、参照されるコンポーネントおよびエンドポイントインスタンスと共に CamelContext が参照されるルートと共に自動的に起動します。

- *Camel スキーマの追加*
- *ルートを 2 つの方法で設定します。*
  - *Java コードの使用*
  - *Spring XML の使用*

296.2. CAMEL スキーマの追加

Camel 1.x の場合は、以下の namespace を使用する必要があります。

> *http://activemq.apache.org/camel/schema/spring*

以下のスキーマの場所になります。

> *http://activemq.apache.org/camel/schema/spring/camel-spring.xsd*

Camel を schemaLocation 宣言に追加する必要があります。

> *http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd*

そのため、XML ファイルは以下のようになります。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
          http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
          http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">
```

296.2.1. camel: namespace の使用

または、XML 宣言の camel XSD を参照できます。

```
xmlns:camel="http://camel.apache.org/schema/spring"
```

i. そのため、宣言は以下のようになります。

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:camel="http://camel.apache.org/schema/spring"
    xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">
```

i. 次に、camel: namespace プレフィックスを使用します。インラインの namespace 宣言を省略できます。

```
<camel:camelContext id="camel5">
  <camel:package>org.apache.camel.spring.example</camel:package>
</camel:camelContext>
```

296.2.2. Spring を使用した高度な設定

詳細は「Spring を使用した CamelContext の高度な設定」を参照してください。

$# using Java Code

Java コードを使用して RouteBuilder 実装を定義できます。これらは spring で Bean として定義してから、Camel コンテキストで参照できます。

296.2.3. <package> の使用

Camel は、指定のパッケージ内のルートの自動検出と初期化を可能にする強力な機能も提供します。これは、Spring コンテキスト定義の camel コンテキストにタグを追加して、RouteBuilder 実装について再帰的に検索するパッケージを指定して設定されます。1.X でこの機能を使用するには、<package></package> タグが必要です。たとえば、検索すべきパッケージのコンマ区切りの一覧を指定する必要があります。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <package>org.apache.camel.spring.config.scan.route</package>
</camelContext>
```

警告： パッケージ名を org.apache.camel またはこのサブパッケージとして指定する場合は注意してください。これにより、Camel がルートについて独自のパッケージで検索し、問題が発生する可能性があります。

INFO:*Will ignore already instantiated classes*.<package> および <packageScan> は Spring によってすでに作成されたクラスをスキップします。そのため、ルートビルダーを spring bean タグとして定義すると、そのクラスはスキップされます。< routeBuilder ref="theBeanId"/> または < contextScan> 機能を使用し て、これらの Bean を含めることが できます。

296.2.4. <packageScan> の使用

Camel 2.0 では、これは、パス一致のような Ant を使用した検出されたルートクラスの選択的包含および除外を可能にするために拡張されました。Spring では、これは <packageScan/> タグを追加して指定されます。タグには 1 つ以上の 'package' 要素が含まれる必要があり（1.x と同様に）、任意で 1 つ以上の「includes」または「excludes」要素で、検出されたクラスの完全修飾名に適用するパターンを指定します。以下に例を示します。

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <packageScan>
    <package>org.example.routes</package>
    <excludes>**.*Excluded*</excludes>
    <includes>**.*</includes>
  </packageScan>
</camelContext>
```

除外パターンは、包含パターンの前に適用されます。include パターンまたは exclude パターンが定義されていない場合は、パッケージで検出されるすべての Route クラスが返されます。

上記の例では、Camel はすべての 'org.example.routes' パッケージと RouteBuilder クラスのすべてのサブパッケージをスキャンします。スキャンで、'MyRoute'' と呼ばれる org.example.routes と、サブパッケージ 'excluded' の別の 'MyExcludedRoute' の 2 つの RouteBuilder が見つかるとします。各クラスの完全修飾名（org.example.routes.MyRoute、org.example.routes.excluded.MyExcludedRoute）が適用され、include パターンおよび exclude パターンが適用されます。

除外パターン *.*Excluded は fqcn 'org.example.routes.excluded.MyExcludedRoute' と初期化から camel に一致させます。

ここでは、これは以下のように Spring の AntPatternMatcher 実装を使用しています。

```
? matches one character
* matches zero or more characters
** matches zero or more segments of a fully qualified name
```

以下に例を示します。

. excluded は org.simple.Excluded、org.apache.camel.SomeExcludedRoute または org.example.RouteWhichIsExcluded

*.??clude ded は org.simple.IncludedRoute, org.simple.Excluded match org.simple.PrecludedRoute と一致します。

296.2.5. contextScan の使用

Camel 2.4 で利用可能

ルートビルダーインスタンスの Spring ApplicationContext など、Camel がコンテナーコンテキストをスキャンできるようにします。これにより、Spring の < component-scan > 機能を使用し、Camel がスキャンプロセスで Spring によって作成された RouteBuilder インスタンスを取得できるようになります。

これにより、Spring @Component を使用してルートにアノテーションを付け、それらのルートを Camel に含めることができます。

```
@Component
public class MyRoute extends SpringRouteBuilder {

    @Override
    public void configure() throws Exception {
        from("direct:start").to("mock:result");
    }
}
```

< packageScan > ドキュメントで説明されているように、ANT スタイルを使用して包含と除外を行うこともできます。

296.3. 他の XML ファイルからルートをインポートする方法

Camel 2.3 の時点で利用可能

Xml Configuration を使用して Camel でルートを定義する場合、他の XML ファイルで一部のルートを定義する必要がある場合があります。たとえば、ルートが多数あり、一部のルートが別の XML ファイルにある場合、アプリケーションの管理に役立ちます。共通のルートおよび再利用可能なルートを別の XML ファイルに保存することもできますが、これは必要に応じてインポートできます。

Camel 2.3 では、新しい <route Context/> タグで実施する < camelContext/ > 外のルートを定義できるようになり ました。

注記 : <routeContext> を使用すると、分離され、<camelContext> で定義された既存の <onException>、<intercept>、<dataFormats>、および同様の相互通信機能を再利用できません。つまり、<routeContext> は現在分離されています。これは Camel 3.x で変更される可能性があります。

たとえば、以下のように 2 つのルートを含む myCoolRoutes.xml という名前のファイルを使用できます。

myCoolRoutes.xml

その後、CamelContext が含まれる XML ファイルで Spring を使用して myCoolRoute.xml ファイルをインポートできます。
そして、< camelContext/> 内では、以下のよう に ID で <routeContext/ > を参照できます。

また、CamelContext 内にルートがあり、さらに RouteContext で外部化されることも、組み合わせて一致させることができることに注意してください。

< routeContextRef/> はいくつで も指定できます。

再利用可能なルート

< routeContext/> で定義されたルートは、複数の < camelContext/> で再利用でき ます。ただし、再利用される定義のみになります。ランタイム時に、各 CamelContext は定義に基づいてルートの独自のインスタンスを作成します。

296.3.1. タイム除外をテストします。

テスト時には、テストシナリオに該当しない、または便利なものから、一致するルートを選択的に除外できます。たとえば、'org.example.routes' パッケージの spring コンテキストファイル routes-context.xml と 3 つの Route Builders RouteA、RouteB、および RouteC が含まれる場合があります。packageScan 定義は、これらの 3 つのルートすべてを検出し、初期化します。

RouteC がテストシナリオに該当せず、テスト中に多くのノイズを生成するとします。この特定のテストからこのルートを除外すると良いでしょう。これを可能にするように SpringTestSupport クラスが変更されました。1 つのクラスやクラスの配列を除外するために上書きできる 2 つのメソッド（excludedRoute および excludedRoutes）を提供します。

```
public class RouteAandRouteBOnlyTest extends SpringTestSupport {
    @Override
    protected Class excludeRoute() {
        return RouteC.class;
    }
}
```

MyExcludedRouteBuilder.class を除外するために spring で camelContext の初期化にフックするには、Spring コンテキストの作成をインターセプトする必要があります。createApplicationContext を上書きして Spring コンテキストを作成するとき、getRouteExcludingApplicationContext（）メソッドを呼び出して除外を処理する特別な親 Spring コンテキストを提供します。

```
@Override
protected AbstractXmlApplicationContext createApplicationContext() {
    return new ClassPathXmlApplicationContext(new String[] {"routes-context.xml"},
getRouteExcludingApplicationContext());
}
```

RouteC が初期化から除外されるようになりました。同様に、RouteC のみをテストする別のテストでは、上書きにより RouteB および RouteA を除外できます。

```
@Override
protected Class[] excludeRoutes() {
    return new Class[]{RouteA.class, RouteB.class};
}
```

296.4. SPRING XML の使用

Spring 2.0 XML 設定を使用して、以下の 例 のようにルートの Xml 設定を指定できます。

296.5. コンポーネントおよびエンドポイントの設定

以下の 例 のように、Spring XML で Component または Endpoint インスタンスを設定できます。

これにより、一部の名前（上記の例の activemq）を使用してコンポーネントを設定し、activemq:[queue:|topic:]destinationName を使用してコンポーネントを参照できます。これは、SpringCamelContext がエンドポイント URI に使用するスキーム名の Spring コンテキストからコンポーネントを取得することで機能します。

詳細は、「エンドポイントとコンポーネント の設定」を参照して ください。

296.6. CAMELCONTEXTAWARE

POJO の CamelContext とともにインジェクトする場合、CamelContextAware インターフェースの実装は CamelContextAware インターフェース を実装している場合、Spring が POJO を作成すると、CamelContext は POJO に注入されます。さらにインジェクションについては、「 Bean インテグレーション 」も参照してください。

296.7. 統合テスト

Spring Transactions を使用したテスト時にルートがハングしないように、Transactional Client の Spring Integration Testing に関する情報を参照してください。

296.8. 関連項目

- Spring JMS チュートリアル
- 新しい Spring ベースの Camel ルートの作成
- Spring の例
- XML リファレンス
- Spring を使用した CamelContext の詳細設定

- *他の XML ファイルからルートをインポートする方法*

第297章 SPRING BATCH コンポーネント

Camel バージョン 2.10 で利用可能

spring-batch: コンポーネントおよびサポートクラスは Camel と Spring Batch インフラストラクチャー間の統合テストを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-batch</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

297.1. URI 形式

```
spring-batch:jobName[?options]
```

jobName は、Camel レジストリーにある Spring Batch ジョブの名前を表します。または、JobRegistry が指定されている場合は、代わりにジョブを見つけるために使用されます。

警告： このコンポーネントは、プロデューサーエンドポイントを定義するためにのみ使用できます。つまり、from（）ステートメントで Spring Batch コンポーネントを使用できないことを意味します。

297.2. オプション

Spring Batch コンポーネントは以下の 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
jobLauncher (producer)	使用する JobLauncher を明示的に指定します。		JobLauncher
jobRegistry (producer)	使用する JobRegistry を明示的に指定します。		JobRegistry

Name	説明	デフォルト	Type
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Spring Batch エンドポイントは、URI 構文を使用して設定します。

```
spring-batch:jobName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

297.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
jobName	レジストリーにある Spring Batch ジョブの名前が **必要です**。		文字列

297.2.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
jobFromHeader (producer)	jobName が URI ではなくヘッダーから取得するかどうかを明示的に定義します。	false	boolean
jobLauncher (producer)	使用する JobLauncher を明示的に指定します。		JobLauncher
jobRegistry (producer)	使用する JobRegistry を明示的に指定します。		JobRegistry
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

297.3. 用途

Spring Batch コンポーネントがメッセージを受信すると、ジョブ実行がトリガーされます。ジョブは、以下のアルゴリズムに従って解決された org.springframework.batch.core.launch.JobLaucher インスタンスを使用して実行されます。

- JobLauncher がコンポーネントに手動で設定されている場合は、これを使用します。
- jobLauncherRef オプションがコンポーネントに設定されている場合は、Camel Registry で、指定の名前で JobLauncher を検索します。非推奨となり、Camel 3.0 で削除されます。
- JobLauncher 名の Camel Registry に JobLauncher が登録されている場合は、これを使用します。
- 上記の手順のいずれも JobLauncher を解決でき、Camel レジストリー内に JobLauncher インスタンスが 1 つしかない場合は、これを使用します。

メッセージで見つかったすべてのヘッダーはジョブパラメーターとして JobLauncher に渡されます。文字 列、Long、Double、および java.util.Date の値は org.springframework.batch.core.JobParametersBuilder にコピーされます。他のデータタイプは Strings に変換されます。

297.4. 例

Spring バッチジョブ実行のトリガー

```
from("direct:startBatch").to("spring-batch:myJob");
```

JobLauncher を明示的に使用して Spring Batch ジョブ実行をトリガーします。

```
from("direct:startBatch").to("spring-batch:myJob?jobLauncherRef=myJobLauncher");
```

JobLauncher によって返される Camel 2.11.1 JobExecution インスタンスからは、SpringBatchProducer が出力メッセージとして転送されます。Spring Batch API を直接使用して、JobExecution インスタンスを使用して一部の操作を実行できます。

```
from("direct:startBatch").to("spring-batch:myJob").to("mock:JobExecutions");
...
MockEndpoint mockEndpoint = ...;
```

```
JobExecution jobExecution =
mockEndpoint.getExchanges().get(0).getIn().getBody(JobExecution.class);
BatchStatus currentJobStatus = jobExecution.getStatus();
```

297.5. サポートクラス

コンポーネントとは別に、Camel Spring Batch は Spring Batch インフラストラクチャーにフックするために使用できるクラスもサポートします。

297.5.1. CamelItemReader

CamelItemReader を使用すると、Camel インフラストラクチャーから直接バッチデータを読み取ることができます。

たとえば、以下のスニペットは、JMS キューからデータを読み取るように Spring Batch を設定します。

```
<bean id="camelReader"
class="org.apache.camel.component.spring.batch.support.CamelItemReader">
  <constructor-arg ref="consumerTemplate"/>
  <constructor-arg value="jms:dataQueue"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="camelReader" writer="someWriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

297.5.2. CamelItemWriter

CamelItemWriter は CamelItemReader と同様の目的を持ちますが、処理されたデータのチャンクを書き込む専用です。

たとえば、以下のスニペットは、JMS キューからデータを読み取るように Spring Batch を設定します。

```
<bean id="camelwriter" class="org.apache.camel.component.spring.batch.support.CamelItemWriter">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="jms:dataQueue"/>
</bean>
```

```
<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="camelwriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

297.5.3. CamelItemProcessor

CamelItemProcessor は、Spring Batch org.springframework.batch.item.ItemProcessor インターフェースの実装です。リクエスト応答パターン で後者実装がリレーされ、バッチ項目の処理を Camel インフラストラクチャーに委譲します。処理するアイテムは、メッセージのボディーとして Camel エンドポイントに送信されます。

たとえば、以下のスニペットは、Direct エンドポイント と Simple 式言語 を使用して、バッチ項目を簡単に処理します。

```
<camel:camelContext>
  <camel:route>
    <camel:from uri="direct:processor"/>
    <camel:setExchangePattern pattern="InOut"/>
    <camel:setBody>
      <camel:simple>Processed ${body}</camel:simple>
    </camel:setBody>
  </camel:route>
</camel:camelContext>

<bean id="camelProcessor"
class="org.apache.camel.component.spring.batch.support.CamelItemProcessor">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="direct:processor"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="someWriter" processor="camelProcessor" commit-
interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

297.5.4. CamelJobExecutionListener

CamelJobExecutionListener は、ジョブ実行イベントを Camel エンドポイントに送信する org.springframework.batch.core.JobExecutionListener インターフェースの実装です。

Spring Batch によって生成される org.springframework.batch.core.JobExecution インスタンスは、メッセージのボディーとして送信されます。before- と after-callbacks SPRING_BATCH_JOB_EVENT_TYPE ヘッダーを区別するために、BEFORE または AFTER の値に設定されます。

以下のスニペット例では、Spring Batch ジョブ実行イベントを JMS キューに送信します。

```
<bean id="camelJobExecutionListener"
class="org.apache.camel.component.spring.batch.support.CamelJobExecutionListener">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="jms:batchEventsBus"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="someWriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
  <batch:listeners>
    <batch:listener ref="camelJobExecutionListener"/>
  </batch:listeners>
</batch:job>
```

297.6. SPRING CLOUD

Camel 2.19 から利用可能

Spring Cloud コンポーネント

このコンポーネントを使用するには、Maven ユーザーは以下の依存関係を pom.xml に追加する必要があります。

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-spring-cloud</artifactId>
    <version>${camel.version}</version> <!-- use the same version as your Camel core version --
>
</dependency>
```

camel-spring-cloud jar には spring.factories ファイルが含まれるため、クラスパスに依存関係を追加するとすぐに、Spring Boot は自動的に Camel を設定します。

297.6.1. Camel Spring Cloud Starter

Camel 2.19 から利用可能

スターターを使用するには、以下を Spring ブートの pom.xml ファイルに追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-cloud-starter</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version --
>
</dependency>
```

297.7. SPRING CLOUD NETFLIX

Camel 2.19 から利用可能

Spring Cloud Netflix コンポーネントブリッジは Camel Cloud と Spring Cloud Netflix のブリッジにより、Spring Cloud Netflix サービス検出および負荷分散機能を活用したり、Camel Service Discovery 実装を Spring Cloud Netflix の Ribbon 負荷の ServerList ソースとして使用することができます。

このコンポーネントを使用するには、Maven ユーザーは以下の依存関係を pom.xml に追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-cloud-netflix</artifactId>
  <version>${camel.version}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

camel-spring-cloud-netflix jar には spring.factories ファイルが含まれています。そのため、クラスパスに依存関係を追加すると、Spring Boot は自動的に Camel を設定します。

Camel Spring Cloud Netflix は、以下のプロパティーを使用して無効にできます。

```
# Enable/Disable the whole integration, default true
camel.cloud.netflix = true
```

```
# Enable/Disable the integration with Ribbon, default true
camel.cloud.netflix.ribbon = true
```

297.8. SPRING CLOUD NETFLIX STARTER

Camel 2.19 から利用可能

スターターを使用するには、以下を Spring ブートの pom.xml ファイルに追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-cloud-netflix-starter</artifactId>
  <version>${camel.version}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第298章 SPRING イベントコンポーネント

Camel バージョン 1.4 で利用可能

spring-event: コンポーネントは Spring ApplicationEvent オブジェクトへのアクセスを提供します。これにより、ApplicationEvent オブジェクトを Spring ApplicationContext に公開したり、それらを消費したりできます。その後、エンタープライズ統合パターン を使用して、メッセージフィルターなどの処理を行うこと ができ ます。

298.1. URI 形式

```
spring-event://default[?options]
```

この時点では、このコンポーネントにはオプションがありません。これは今後のリリースで簡単に変更できるため、再度確認してください。

298.2. SPRING イベントオプション

Spring Event コンポーネントにはオプションがありません。

Spring Event エンドポイントは、URI 構文を使用して設定します。

```
spring-event:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

298.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	エンドポイントの名前		文字列

298.2.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

298.3. 関連項目

- ***Configuring Camel (Camel の設定)***
- ***コンポーネント***
- ***エンドポイント***
- ***はじめに***

第299章 SPRING 統合コンポーネント

Camel バージョン 1.4 で利用可能

spring-integration: コンポーネントは Camel コンポーネントが spring インテグレーションエンドポイント と通信するためのブリッジを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-integration</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

299.1. URI 形式

```
spring-integration:defaultChannelName[?options]
```

defaultChannelName は、Spring Integration Spring コンテキストによって使用されるデフォルトのチャネル名を表します。Spring Integration コンシューマーの inputChannel 名および Spring Integration プロバイダーの outputChannel 名と同じになります。

URI にクエリーオプションを追加するには、?option=value&option=value&…

299.2. オプション

Spring Integration コンポーネントにはオプションがありません。

Spring Integration エンドポイントは、URI 構文を使用して設定します。

```
spring-integration:defaultChannel
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

299.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
defaultChannel	Spring Integration Spring コンテキストによって使用されるデフォルトのチャネル名が **必要** です。Spring Integration コンシューマーの inputChannel 名および Spring Integration プロバイダーの outputChannel 名と同じになります。		文字列

299.2.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
inOut (common)	Spring インテグレーションエンドポイントが使用するエクスチェンジパターン。inOut=true の場合、リプライチャネルは Spring Integration Message ヘッダーからか、またはエンドポイントに設定されたもののいずれかを想定されます。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
inputChannel (consumer)	このエンドポイントが Spring インテグレーションから消費する Spring インテグレーション入力チャネル名。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
outputChannel (producer)	Spring インテグレーションにメッセージを送信するために使用される Spring インテグレーションの出力チャネル名。		文字列

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

299.3. 用途

Spring インテグレーションコンポーネントは、Spring インテグレーションの入力チャネルおよび出力チャネルを介して Camel エンドポイントを Spring インテグレーションエンドポイントに接続するブリッジです。このコンポーネントを使用して、Camel メッセージを Spring Integration エンドポイントに送信したり、Camel ルーティングコンテキストで Spring インテグレーションエンドポイントからメッセージを受信することができます。

299.4. 例

299.4.1. Spring インテグレーションエンドポイントの使用

以下のように URI を使用して Spring インテグレーションエンドポイントを設定できます。

または、Spring インテグレーションチャネル名を直接使用します。

299.4.2. ソースおよびターゲットアダプター

Spring インテグレーションは、Spring インテグレーションチャネルから Camel エンドポイントまたは Camel エンドポイントから Spring インテグレーションチャネルにメッセージをルーティングできる Spring インテグレーションのソースおよびターゲットアダプターも提供します。

この例では、以下の namespace を使用します。

以下のように、ソースまたはターゲットを Camel エンドポイントにバインドできます。

299.5. 関連項目

- *Configuring Camel (Camel の設定)*

- コンポーネント
- エンドポイント
- はじめに

299.6. SPRING JAVA CONFIG

XML Config を使用して Bean を一緒に接続して Spring が開始しました。しかし、場合によっては XML を使用せず、Spring JavaConfig プロジェクトとともに Guice の作成を行う Java コードを使用しています。

Camel では XML または Java 設定アプローチのいずれかを使用できます。実際にお好みの方が望ましいです。

299.6.1. Spring Java Config の使用

Camel プロジェクトで Spring Java Config を使用するには、以下を pom.xml に追加するのが最も簡単な方法です。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-javaconfig</artifactId>
  <version>${camel-version}</version>
</dependency>
```

これにより、Spring 内で Camel を設定するためのヘルパークラスとともに、Spring JavaConfig ライブラリーの依存関係が追加されます。

このライブラリーは完全に任意であることに注意してください。Camel を Java Config と組み合わせることができるだけです。

299.6.2. 設定

JavaConfig を Camel で使用する最も一般的なケースとして、ルーターが使用する定義されたルートのリストを持つ設定が作成されます。

```
@Configuration
public class MyRouteConfiguration extends CamelConfiguration {

    @Autowire
    private MyRouteBuilder myRouteBuilder;

    @Autowire
    private MyAnotherRouteBuilder myAnotherRouteBuilder;

    @Override
    public List<RouteBuilder> routes() {
        return Arrays.asList(myRouteBuilder, myAnotherRouteBuilder);
    }

}
```

Camel 2.13.0 以降では、route （） 定義を省略し、Spring コンテキストにある RouteBuilder インスタンスにフォールバックすることができます。

```
@Configuration
@ComponentScan("com.example.routes")
public class MyRouteConfiguration extends CamelConfiguration {
}
```

299.6.3. テスト

Camel 2.11.0 以降、CamelSpringDelegatingTestContextLoader で CamelSpringJUnit4ClassRunner を使用できます。これは、Java Config および Camel インテグレーションをテストするのに推奨される方法です。

RouteBuilder インスタンスのコレクションを作成する場合は、CamelConfiguration ヘルパークラスから派生し、routes（） メソッドを実装します。route（）メソッドを上書きしていない場合（Camel 2.13.0 から開始）、CamelConfiguration は Spring コンテキストで利用可能な RouteBuilder インスタンスをすべて使用することに注意してください。

以下の 例は、Java Config を使用して Camel 2.10 以前との Java Config インテグレーションをテストする方法を示しています。JavaConfigContextLoader は非推奨となり、CamelSpringDelegatingTestContextLoader の代わりに Camel の今後のバージョンで削除される可能性があります。

@ContextConfiguration アノテーションは、使用する設定として ContextConfig クラスをロードするように Spring Testing フレームワークに指示します。このクラスは、ヘルパー Spring Java Config クラスである SingleRouteCamelConfiguration から派生します。このクラスは、CamelContext を設定し、作成する RouteBuilder を登録します。

第300章 SPRING LDAP コンポーネント

Camel バージョン 2.11 で利用可能

spring-ldap: コンポーネントは Spring LDAP の Camel ラッパーを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-ldap</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

300.1. URI 形式

```
spring-ldap:springLdapTemplate[?options]
```

springLdapTemplate は Spring LDAP テンプレート Bean の名前です。この Bean では、LDAP アクセスの URL およびクレデンシャルを設定します。

300.2. オプション

Spring LDAP コンポーネントにはオプションがありません。

Spring LDAP エンドポイントは、URI 構文を使用して設定します。

```
spring-ldap:templateName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

300.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
templateName	Spring LDAP テンプレート Bean の **必須** 名		文字列

300.2.2. クエリーパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
操作 （プロデューサー）	LDAP 操作を実行する **必要** があります。		LdapOperation
scope (producer)	検索操作の範囲。	subtree	文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

300.3. 用途

コンポーネントはプロデューサーエンドポイントのみをサポートします。コンシューマーエンドポイントの作成を試みると、UnsupportedOperationException が発生します。
メッセージのボディーはマップ（java.util.Mapのインスタンス）である必要があります。ContextSource の設定でベース DN が指定されていない限り、このマップには LDAP 操作を実行するルートノードを指定する dn キー（function_driven 操作には必要ありません）を持つエントリーが含まれている必要があります。マップの他のエントリーは操作固有です（以下を参照）。

バインドおよびバインド 解除 の操作では、メッセージのボディーは変更されません。search および function_driven 操作については、ボディーは検索の結果に設定されます。http://static.springsource.org/spring-ldap/site/apidocs/org/springframework/ldap/core/LdapTemplate.html#search%28java.lang.String,%20java.lang.String,%20int,%20org.springframework.ldap.core.AttributesMapper%29 を参照してください。

300.3.1. Search

メッセージボディーには、キー フィルター のエントリーが必要です。この値は、有効な LDAP フィルターを表す String である必要があります。http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol#Search_and_Compare を参照してください。

300.3.2. バインド

メッセージボディーには、キー 属性 のエントリーが必要です。値は javax.naming.directory.Attributes のインスタンスである必要があります。このエントリーは、作成する LDAP ノードを指定します。

300.3.3. unbind（バインド解除）

これ以外のエントリーは必要ありません。指定した dn のノードが削除されます。

300.3.4. 認証

メッセージボディーには、キー フィルター と パスワード が含まれるエントリーが必要です。値は、有効な LDAP フィルターとユーザーパスワードを表す String のインスタンスである必要があります。

300.3.5. 属性の変更

メッセージボディーには、キー modificationItems が含まれるエントリーが必要です。値は javax.naming.directory.ModificationItemタイプの任意の配列のインスタンスである必要があります。

300.3.6. 関数駆動型

メッセージボディーには、keys function と request のエントリーが必要です。function の値は java.util.function.BiFunction<L、Q、S> タイプである必要があります。L type パラメーターは org.springframework.ldap.core.LdapOperations のタイプでなければなりません。request の値は、関数の Q タイプのパラメーターと同じタイプで、関数 内で呼び出される LdapTemplate メソッドによって想定されるパラメーターをカプセル化する必要があります。 S type パラメーターは、呼び出される LdapTemplate メソッドによって返される応答タイプを表します。この操作により、上記の操作で対応していない LdapTemplate メソッドの動的呼び出しが可能です。

キー定義

スペルエラーを回避するために、以下の定数が org.apache.camel.springldap.SpringLdapProducer に定義されています。

- public static final String DN = "dn"

- *public static final String FILTER = "filter"*
- *public static final String ATTRIBUTES = "attributes"*
- *public static final String PASSWORD = "password";*
- *public static final String MODIFICATION_ITEMS = "modificationItems";*
- *public static final String FUNCTION = "function";*
- *public static final String REQUEST = "request";*

第301章 SPRING REDIS コンポーネント

Camel バージョン 2.11 で利用可能

このコンポーネントでは、Redis からのメッセージの送受信が可能です。Redis は高度なキーと値のストアです。キーには文字列、ハッシュ、リスト、セット、およびソートセットを含めることができます。さらに、アプリケーション間の通信に pub/sub 機能を提供します。
Camel は、コマンドを実行するためのプロデューサーを提供します。コンシューマーは、重複したメッセージをフィルタリングするためにべき等リポジトリーをサブスクライブするコンシューマーを提供します。

*INFO:*前提条件* このコンポーネントを使用するには、Redis サーバーが実行している必要があります。*

301.1. URI 形式

```
spring-redis://host:port[?options]
```

URI には、?options=value&option2=value&... という形式でクエリーオプションを追加できます。

301.2. URI オプション

Spring Redis コンポーネントにはオプションがありません。

Spring Redis エンドポイントは、URI 構文を使用して設定します。

```
spring-redis:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

301.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
host	Redis サーバーが実行されているホストが **必要** です。		文字列

Name	説明	デフォルト	Type
port	**必要な** Redis サーバーのポート番号		整数

301.2.2. クエリーパラメーター（10 パラメーター）：

Name	説明	デフォルト	Type
チャンネル（共通）	サブスクライブするトピック名または名前パターンの一覧。複数の名前はコンマで区切ることができます。		文字列
コマンド (common)	デフォルトコマンド： メッセージヘッダーで上書きできます。コンシューマーは PSUBSCRIBE および SUBSCRIBE コマンドのみをサポートすることに注意してください。	SET	コマンド
connectionFactory (common)	使用する事前設定された RedisConnectionFactory インスタンスへの参照。		RedisConnectionFactory
redisTemplate (common)	使用する事前設定された RedisTemplate インスタンスへの参照。		RedisTemplate
シリアライザー (common)	使用する事前設定された RedisSerializer インスタンスへの参照。		RedisSerializer
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
listenerContainer (consumer)	使用する事前設定された RedisMessageListenerContainer インスタンスへの参照。		RedisMessageListener Container
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

301.3. 用途

https://github.com/apache/camel/tree/master/components/camel-spring-redis/src/test/java/org/apache/camel/component/redis で利用可能なユニットテストも参照してください。

301.3.1. Redis プロデューサーによって評価されるメッセージヘッダー

プロデューサーはサーバーにコマンドを実行し、各コマンドには特定タイプで異なるパラメーターセットがあります。コマンド実行の結果はメッセージのボディーに返されます。

ハッシュコマンド	説明	パラメーター	結果
HSET	ハッシュフィールドの文字列値を設定します。	CamelRedis.Key (String), CamelRedis.Field (String), CamelRedis.Value (Object)	void
HGET	ハッシュフィールドの値を取得します。	CamelRedis.Key (String), CamelRedis.Field (String)	文字列
HSETNX	フィールドが存在しない場合のみ、ハッシュフィールドの値を設定します。	CamelRedis.Key (String), CamelRedis.Field (String), CamelRedis.Value (Object)	void

ハッシュコマンド	説明	パラメーター	結果
HMSET	複数のハッシュフィールドに複数の値を設定する	CamelRedis.Key (String), CamelRedis.Values(Map<String, Object>)	void
HMGET	指定のハッシュフィールドの値をすべて取得します。	CamelRedis.Key (String), CamelRedis.Fields (Collection<String>)	Collection<Object>
HINCRBY	ハッシュフィールドの整数値を指定の数でインクリメントします。	CamelRedis.Key (String), CamelRedis.Field (String), CamelRedis.Value (Long)	Long
HEXISTS	ハッシュフィールドが存在するかどうかを判別	CamelRedis.Key (String), CamelRedis.Field (String)	ブール値
HDEL	1 つ以上のハッシュフィールドを削除します。	CamelRedis.Key (String), CamelRedis.Field (String)	void
HLEN	ハッシュ内のフィールド数を取得します。	CamelRedis.Key (String)	Long
HKEYS	ハッシュ内のフィールドをすべて取得します。	CamelRedis.Key (String)	set<String>
HVALS	ハッシュ内のすべての値を取得します。	CamelRedis.Key (String)	Collection<Object>

ハッシュコマンド	説明	パラメーター	結果
HGETALL	ハッシュのすべてのフィールドおよび値を取得します。	CamelRedis.Key (String)	map<String, Object>

コマンドの一覧表示	説明	パラメーター	結果
RPUSH	リストに値を 1 つまたは複数追加	CamelRedis.Key (String), CamelRedis.Value (Object)	Long
RPUSHX	リストが存在する場合のみ、一覧に値を追加します。	CamelRedis.Key (String), CamelRedis.Value (Object)	Long
LPUSH	1 つまたは複数の値をリストに追加します。	CamelRedis.Key (String), CamelRedis.Value (Object)	Long
LLEN	リストの長さの取得	CamelRedis.Key (String)	Long
LRANGE	リストからさまざまな要素を取得します。	CamelRedis.Key (String), CamelRedis.Start (Long), CamelRedis.End (Long)	List<Object>
LTRIM	指定された範囲にリストをトリミングします。	CamelRedis.Key (String), CamelRedis.Start (Long), CamelRedis.End (Long)	void
LINDEX	インデックスによるリストから要素を取得する	CamelRedis.Key (String), CamelRedis.Index (Long)	文字列

コマンドの一覧表示	説明	パラメーター	結果
LINSERT	リスト内の別の要素の前または後への要素の挿入	CamelRedis.Key(String)、CamelRedis.Value(Object)、CamelRedis.Pivot（文字列）、CamelRedis.Position（文字列）	Long
LSET	インデックスでリスト内の要素の値を設定します。	CamelRedis.Key (String), CamelRedis.Value (Object), CamelRedis.Index (Long)	void
LREM	リストからの要素の削除	CamelRedis.Key (String), CamelRedis.Value (Object), CamelRedis.Count (Long)	Long
LPOP	リストの最初の要素を削除し、取得します。	CamelRedis.Key (String)	オブジェクト
RPOP	リストの最後の要素を削除し、取得します。	CamelRedis.Key (String)	文字列
RPOPLPUSH	リストの最後の要素を削除し、別のリストに追加し、返します。	CamelRedis.Key (String), CamelRedis.Destination (String)	オブジェクト

コマンドの一覧表示	説明	パラメーター	結果
BRPOPLPUSH	リストから値をポップアップし、別のリストにプッシュして返します。あるいは、利用できるまでブロックします。	CamelRedis.Key (String), CamelRedis.Destination (String), CamelRedis.Timeout (Long)	オブジェクト
BLPOP	一覧の最初の要素を削除して取得するか、またはブロックが利用可能になるまでブロックします。	CamelRedis.Key (String), CamelRedis.Timeout (Long)	オブジェクト
BRPOP	リストの最後の要素を削除して取得するか、または1つが利用可能になるまでブロックします。	CamelRedis.Key (String), CamelRedis.Timeout (Long)	文字列

コマンドの実行	説明	パラメーター	結果
SADD	セットへの1つ以上のメンバーの追加	CamelRedis.Key (String), CamelRedis.Value (Object)	ブール値
SMEMBERS	セットのすべてのメンバーを取得します。	CamelRedis.Key (String)	set<Object>
SREM	セットから1つ以上のメンバーを削除します。	CamelRedis.Key (String), CamelRedis.Value (Object)	ブール値

コマンドの実行	説明	パラメーター	結果
SPOP	セットからランダムなメンバーを削除して返します。	CamelRedis.Key (String)	文字列
SMOVE	あるセットから別のセットへのメンバーの移動	CamelRedis.Key (String), CamelRedis.Value (Object), CamelRedis.Destination (String)	ブール値
SCARD	セットのメンバー数を取得します。	CamelRedis.Key (String)	Long
SISMEMBER	指定した値がセットのメンバーであるかどうかを判別	CamelRedis.Key (String), CamelRedis.Value (Object)	ブール値
SINTER	Intersect 複数セット	CamelRedis.Key (String), CamelRedis.Keys (String)	set<Object>
SINTERSTORE	intersect の複数セットと、キーセットを保存する	CamelRedis.Key (String), CamelRedis.Keys (String), CamelRedis.Destination (String)	void
SUNION	複数のセットの追加	CamelRedis.Key (String), CamelRedis.Keys (String)	set<Object>
SUNIONSTORE	複数のセットを追加し、結果として得られるセットをキーに保存します。	CamelRedis.Key (String), CamelRedis.Keys (String), CamelRedis.Destination (String)	void

コマンドの実行	説明	パラメーター	結果
SDIFF	複数のセットを減算	CamelRedis.Key (String), CamelRedis.Keys (String)	set<Object>
SDIFFSTORE	複数のセットを減算し、結果セットをキーに保存します。	CamelRedis.Key (String), CamelRedis.Keys (String), CamelRedis.Destination (String)	void
SRANDMEMBER	セットから1つまたは複数のランダムなメンバーを取得する	CamelRedis.Key (String)	文字列

順序付けされた set コマンド	説明	パラメーター	結果
ZADD	ソートされたセットに1つ以上のメンバーを追加するか、すでに存在している場合はそのスコアを更新します。	CamelRedis.Key (String), CamelRedis.Value (Object), CamelRedis.Score (Double)	ブール値

順序付けされた set コマンド	説明	パラメーター	結果
ZRANGE	インデックスでソートされたセット内のメンバーの範囲を返します。	CamelRedis.Key(String)、CamelRedis.Start(Long)、CamelRedis.End(Long)、CamelRedis.WithScore(Boolean)	オブジェクト
ZREM	ソートされたセットから1つ以上のメンバーを削除します。	CamelRedis.Key (String), CamelRedis.Value (Object)	ブール値
ZINCRBY	ソートセット内のメンバーのスコアを増やす	CamelRedis.Key (String), CamelRedis.Value (Object), CamelRedis.Increment (Double)	double
ZRANK	ソートセット内のメンバーのインデックスを決定する	CamelRedis.Key (String), CamelRedis.Value (Object)	Long
ZREVRANK	ソートセット内のメンバーのインデックスを決定し、スコアは高から低いものから順に並べます。	CamelRedis.Key (String), CamelRedis.Value (Object)	Long
ZREVRANGE	ソートされたセット内のメンバーの範囲をインデックスで返します。スコアは高から低い値から順に並べられます。	CamelRedis.Key(String)、CamelRedis.Start(Long)、CamelRedis.End(Long)、CamelRedis.WithScore(Boolean)	オブジェクト

順序付けされた set コマンド	説明	パラメーター	結果
ZCARD	ソートセット内のメンバー数を取得します。	CamelRedis.Key (String)	Long
ZCOUNT	指定値内のスコアでソートされたセット内のメンバーをカウントします。	CamelRedis.Key (String), CamelRedis.Min (Double), CamelRedis.Max (Double)	Long
ZRANGEBYSCORE	スコアでソートされたセット内のメンバーの範囲を返します。	CamelRedis.Key (String), CamelRedis.Min (Double), CamelRedis.Max (Double)	set<Object>
ZREVRANGEBYSCORE	スコア順でソートされたセット内のメンバーの範囲を返します（スコアは高から低い）。	CamelRedis.Key (String), CamelRedis.Min (Double), CamelRedis.Max (Double)	set<Object>
ZREMRANGEBYRANK	指定されたインデックス内のソートされたセット内の全メンバーを削除します。	CamelRedis.Key (String), CamelRedis.Start (Long), CamelRedis.End (Long)	void
ZREMRANGEBYSCORE	指定したスコア内のソートセット内の全メンバーを削除します。	CamelRedis.Key (String), CamelRedis.Start (Long), CamelRedis.End (Long)	void
ZUNIONSTORE	複数のソートセットを追加し、結果のソートセットを新しいキーに保存します。	CamelRedis.Key (String), CamelRedis.Keys (String), CamelRedis.Destination (String)	void

順序付けされた set コマンド	説明	パラメーター	結果
ZINTERSTORE	intersect に複数のソートセットが含まれ、その結果のソートセットが新規キーに保存されます。	CamelRedis.Key (String), CamelRedis.Keys (String), CamelRedis.Destination (String)	void

文字列コマンド	説明	パラメーター	結果
SET	キーの文字列値を設定します。	CamelRedis.Key (String), CamelRedis.Value (Object)	void
GET	キーの値を取得します。	CamelRedis.Key (String)	オブジェクト
STRLEN	キーに保存されている値の長さを取得します。	CamelRedis.Key (String)	Long
APPEND	値をキーに追加します。	CamelRedis.Key (String), CamelRedis.Value (String)	整数
SETBIT	キーに保存されている文字列値のオフセットでビットを設定またはクリアします。	CamelRedis.Key (String), CamelRedis.Offset (Long), CamelRedis.Value (Boolean)	void
GETBIT	キーに保存されている文字列値のオフセットでビット値を返します。	CamelRedis.Key (String), CamelRedis.Offset (Long)	ブール値

文字列コマンド	説明	パラメーター	結果
SETRANGE	指定のオフセットで始まるキーで文字列の一部を上書きします。	CamelRedis.Key (String), CamelRedis.Value (Object), CamelRedis.Offset (Long)	void
GETRANGE	キーに保存されている文字列のサブ文字列を取得します。	CamelRedis.Key (String), CamelRedis.Start (Long), CamelRedis.End (Long)	文字列
SETNX	キーが存在しない場合のみ、キーの値を設定する	CamelRedis.Key (String), CamelRedis.Value (Object)	ブール値
SETEX	キーの値と有効期限を設定します。	CamelRedis.Key (String), CamelRedis.Value (Object), CamelRedis.Timeout (Long), SECONDS	void
DECRBY	指定された数値でキーの整数値を減らします。	CamelRedis.Key (String), CamelRedis.Value (Long)	Long
DECR	キーの整数値のデクリメント	CamelRedis.Key (String),	Long
INCRBY	指定された量でキーの整数値を増やします。	CamelRedis.Key (String), CamelRedis.Value (Long)	Long
INCR	キーの整数値を 1 つずつ増やします。	CamelRedis.Key (String)	Long

文字列コマンド	説明	パラメーター	結果
MGET	指定されたすべてのキーの値を取得します。	CamelRedis.Fields (Collection<String>)	List<Object>
MSET	複数のキーの値を複数に設定する	CamelRedis.Values(Map<String, Object>)	void
MSETNX	キーが存在しない場合のみ、複数の鍵に複数の値を設定します。	CamelRedis.Key (String), CamelRedis.Value (Object)	void
GETSET	キーの文字列値を設定し、古い値を返します。	CamelRedis.Key (String), CamelRedis.Value (Object)	オブジェクト

キーコマンド	説明	パラメーター	結果
EXISTS	キーが存在するかどうかを判別	CamelRedis.Key (String)	ブール値
DEL	キーの削除	CamelRedis.Keys (String)	void
TYPE	キーに保存されているタイプの特定	CamelRedis.Key (String)	DataType
KEYS	指定のパターンに一致するキーをすべて検索	CamelRedis.Pattern (String)	collection<String>
RANDOMKEY	キースペースからランダムなキーを返します。	CamelRedis.Pattern (String), CamelRedis.Value (String)	文字列

キーコマンド	説明	パラメーター	結果
RENAME	鍵の名前変更	CamelRedis.Key (String)	void
RENAMENX	キーの名前を変更します（新しいキーが存在しない場合のみ）。	CamelRedis.Key (String), CamelRedis.Value (String)	ブール値
EXPIRE	キーの時間を秒単位で設定します。	CamelRedis.Key (String), CamelRedis.Timeout (Long)	ブール値
SORT	リスト、セット、またはソートセット内の要素を並べ替えます。	CamelRedis.Key (String)	List<Object>
永続化	キーから有効期限を削除します。	CamelRedis.Key (String)	ブール値
EXPIREAT	キーの有効期限を UNIX タイムスタンプとして設定します。	CamelRedis.Key (String), CamelRedis.Timestamp (Long)	ブール値
PEXPIRE	キーの時間をミリ秒単位で設定します。	CamelRedis.Key (String), CamelRedis.Timeout (Long)	ブール値
PEXPIREAT	キーの有効期限をミリ秒単位で指定された UNIX タイムスタンプとして設定します。	CamelRedis.Key (String), CamelRedis.Timestamp (Long)	ブール値
TTL	キーの有効期間を取得します。	CamelRedis.Key (String)	Long

キーコマンド	説明	パラメーター	結果
MOVE	鍵を別のデータベースに移動する	CamelRedis.Key (String), CamelRedis.Db (Integer)	ブール値

その他のコマンド	説明	パラメーター	結果
MULTI	トランザクションブロックの開始をマーク付けします。	none	void
DISCARD	MULTI の後に発行されたすべてのコマンドを破棄します。	none	void
EXEC	MULTI の後に発行されたすべてのコマンドを実行します。	none	void
WATCH	指定のキーを確認して、MULTI/EXEC ブロックの実行を確認します。	CamelRedis.Keys (String)	void
UNWATCH	すべての監視された鍵について忘れないようにしてください。	none	void
ECHO	指定された文字列をエコーします。	CamelRedis.Value (String)	文字列
PING	サーバーに ping します。	none	文字列
QUIT	接続を閉じる	none	void

その他のコマンド	説明	パラメーター	結果
PUBLISH	チャネルへのメッセージの投稿	CamelRedis.Channel (String), CamelRedis.Message (Object)	void

301.4. 依存関係

Maven ユーザーは、以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-redis</artifactId>
  <version>${camel-version}</version>
</dependency>
```

ここで、${camel-version} は Camel の実際のバージョン（2.11 以降）に置き換える必要があります。

301.5. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*
- *はじめに*

第302章 SPRING SECURITY

Camel 2.3 の時点で利用可能

camel-spring-security コンポーネントは、Camel ルートにロールベースの承認を提供します。Spring Security が提供する認証およびユーザーサービス（以前の Acegi Security）を活用し、宣言的でロールベースのポリシーシステムを追加して、指定のプリンシパルがルートを実行できるかどうかを制御します。

Spring Security の認証および承認システムに慣れていない場合は、上記のリンク先の SpringSource Web サイトに関する現在のリファレンスドキュメントを参照してください。

302.1. 認証ポリシーの作成

ルートへのアクセスは、SpringSecurityAuthorizationPolicy オブジェクトのインスタンスによって制御されます。ポリシーオブジェクトには、一連のエンドポイントを実行し、現在のプリンシパルにそのロールが割り当てられているかどうかを判断するために使用される Spring Security AuthenticationManager オブジェクトおよび AccessDecisionManager オブジェクトを参照するのに必要な Spring Security authority（ロール）の名前が含まれます。ポリシーオブジェクトは Spring Bean として設定することも、Spring XML で < authorizationPolicy& gt; 要素を使用して設定することもできます。

&lt ;authorizationPolicy> 要素 には以下の属性を含めることができます。

Name	デフォルト値	説明
id	**null**	ルートでポリシーを参照するために使用される一意の Spring Bean 識別子（必須）
access	**null**	アクセスデシジョンマネージャーに渡される Spring セキュリティー認証局名（必須）
authenticationManager	**authenticationManager**	コンテキストの Spring Security **AuthenticationManager** オブジェクトの名前
accessDecisionManager	**accessDecisionManager**	コンテキストの Spring Security **AccessDecisionManager** オブジェクトの名前

Name	デフォルト値	説明
authenticationAdapter	Default Authentication Adapter	Camel 2.4: **javax.security.auth.Subject** を Spring Security **Authentication** インスタンスに変換するために使用されるコンテキストの **camel-spring-security AuthenticationAdapter** オブジェクトの名前。
useThreadSecurityContext	**true**	**javax.security.auth.Subject** が Exchange.AUTHENTICATION の In メッセージヘッダーに見つからない場合は、**Authentication** オブジェクトの Spring Security **SecurityContextHolder** を確認します。
alwaysReauthenticate	**false**	true に設定すると、**SpringSecurityAuthorizationPolicy** はポリシーにアクセスするたびに **AuthenticationManager.authenticate（）** を常に呼び出します。

302.2. CAMEL ルートへのアクセスの制御

このコンポーネントを使用するには、Spring Security AuthenticationManager および AccessDecisionManager が必要です。以下は、Spring Security namespace を使用して Spring XML でこれらのオブジェクトを設定する方法の例になります。

基礎となるセキュリティーオブジェクトが設定されているので、それらを使用して承認ポリシーを設定し、そのポリシーを使用してルートへのアクセスを制御できます。

この例では、エンドポイントの mock:end は、認証の有無が 認証 され、ROLE_ADMIN 認証局が管理者 SpringSecurityAuthorizationPolicy によって配置されない限り実行されません。

302.3. AUTHENTICATION

承認に使用されるセキュリティー認証情報を取得するプロセスは、このコンポーネントで指定されません。必要に応じて、エクスチェンジから認証情報を取得する独自のプロセッサーまたはコンポーネントを作成できます。たとえば、Jetty コンポーネントで作成した HTTP リクエストヘッダーから認証情報を取得するプロセッサーを作成することができます。クレデンシャルを収集する方法に関係なく、Camel Spring Security コンポーネントがそれらにアクセスできるように In メッセージまたは SecurityContextHolder に配置する必要があります。

```
import javax.security.auth.Subject;
import org.apache.camel.*;
```

```
import org.apache.commons.codec.binary.Base64;
import org.springframework.security.authentication.*;


public class MyAuthService implements Processor {
   public void process(Exchange exchange) throws Exception {
      // get the username and password from the HTTP header
      // http://en.wikipedia.org/wiki/Basic_access_authentication
      String userpass = new
String(Base64.decodeBase64(exchange.getIn().getHeader("Authorization", String.class)));
      String[] tokens = userpass.split(":");

      // create an Authentication object
      UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(tokens[0], tokens[1]);

      // wrap it in a Subject
      Subject subject = new Subject();
      subject.getPrincipals().add(authToken);

      // place the Subject in the In message
      exchange.getIn().setHeader(Exchange.AUTHENTICATION, subject);

      // you could also do this if useThreadSecurityContext is set to true
      // SecurityContextHolder.getContext().setAuthentication(authToken);
   }
}
```

SpringSecurityAuthorizationPolicy は、必要に応じて Authentication オブジェクトを自動的に認証します。

Exchange.AUTHENTICATION ヘッダーの代わりに、または SecurityContextHolder を使用する場合は、2 つの問題があります。まず、コンテキストホルダーはスレッドローカル変数を使用して 認証オブジェクトを保持します。seda や jms などのスレッド境界にまたがるルートは、認証 オブジェクトが失われます。次に、Spring Security システムは、コンテキストの Authentication オブジェクトがすでに認証されており、ロールがあることが予想されます（詳細は、Technical Overview セクション を参照してください）。

camel-spring-security のデフォルト動作は、Exchange.AUTHENTICATION ヘッダーで Subject を検索します。この サブジェクト には、org.springframework.security.core.Authentication のサブクラスであるプリンシパルが少なくとも 1 つ含まれる必要があります。Subject から Authentication オブジェクトへのマッピングをカスタマイズするには、org.apache.camel.component.spring.security.AuthenticationAdapter の実装を <authorizationPolicy> Bean に指定します。これは、Spring Security を使用せず、Subject を提供するコンポーネントを使用している場合に役立ちます。現時点では、CXF コンポーネントのみが Exchange.AUTHENTICATION ヘッダーを設定します。

302.4. 認証および認可エラーの処理

SpringSecurityAuthorizationPolicy で認証または承認が失敗すると、CamelAuthorizationException が発生します。これは、例外句などの Camel の標準的な例外処理メソッドを使用して処理できます。CamelAuthorizationException には例外を書き換えたポリシーの ID への参照があり、ポリシーと例外のタイプに基づいてエラーを処理できます。

```
<onException>
  <exception>org.springframework.security.authentication.AccessDeniedException</exception>
  <choice>
    <when>
      <simple>${exception.policyId} == 'user'</simple>
      <transform>
        <constant>You do not have ROLE_USER access!</constant>
      </transform>
    </when>
    <when>
      <simple>${exception.policyId} == 'admin'</simple>
      <transform>
        <constant>You do not have ROLE_ADMIN access!</constant>
      </transform>
    </when>
  </choice>
</onException>
```

302.5. 依存関係

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-security</artifactId>
  <version>2.4.0</version>
</dependency>
```

この依存関係は、org.springframework.security:spring-security-core:3.0.3.RELEASE および org.springframework.security:spring-security-config:3.0.3.RELEASE もプルします。

302.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- *エンドポイント*
- *はじめに*
- *コンポーネント*

第303章 SPRING WEBSERVICE コンポーネント

Camel バージョン 2.6 で利用可能

spring-ws: コンポーネントを使用すると、Spring Web Services と統合できます。Web サービスにアクセスするための クライアント サイドサポート、および独自のコントラクトファーストの Web サービスを作成するための サーバーの両方を提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-ws</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

*INFO:*Dependencies* Camel 2.8 以降、このコンポーネントは Spring 3.0.x を必要とする Spring-WS 2.0.x に同梱されるようになりました。以前の Camel バージョンには、Spring 2.5.x および 3.0.x と互換性のある Spring-WS 1.5.9 が同梱されていました。Spring 2.5.x で以前のバージョンの camel-spring-ws を実行するには、Spring 2.5.x から spring-webmvc モジュールを追加する必要があります。Spring 3.0.x で Spring-WS 1.5.9 を実行するには、このモジュールも Spring 3.0.x から OXM モジュールを除外する必要があります（この後を参照）。*

303.1. URI 形式

このコンポーネントの URI スキームは以下のとおりです。

```
spring-ws:[mapping-type:]address[?options]
```

Web サービス マッピングタイプ を公開するには、以下のいずれかに設定する必要があります。

マッピングタイプ	説明
rootqname	メッセージに含まれるルート要素の修飾名に基づいて Web サービスリクエストをマップするオプションを提供します。

マッピングタイプ	説明
SOAP Action	メッセージのヘッダーに指定された SOAP アクションに基づいて Web サービスリクエストをマッピングするために使用されます。
uri	特定の URI をターゲットとする Web サービスリクエストをマッピングします。
xpathresult	受信メッセージに対して XPath **式** の評価に基づいて Web サービスリクエストをマッピングするために使用されます。評価の結果は、エンドポイント URI に指定された XPath 結果と一致する必要があります。
beanname	**PayloadRootQNameEndpointMapping**、**SoapActionEndpointMapping** など、既存の（レガシー）エンドポイントマッピング と統合するために、**org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher** オブジェクトを参照できます。

コンシューマーとして、アドレス には指定されたマッピングタイプに関連する値が含まれている必要があります（例： SOAP アクション、XPath 式）。プロデューサーとして、アドレスは呼び出した Web サービスの URI に設定する必要があります。

URI にクエリー オプション を追加するには、 ?option=value&option=value&...

303.2. オプション

Spring WebService コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Spring WebService エンドポイントは、URI 構文を使用して設定します。

```
spring-ws:type:lookupKey:webServiceEndpointUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

303.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
type	エンドポイントマッピングが使用される場合のエンドポイントマッピングタイプ。rootqname - メッセージに含まれるルート要素の修飾名に基づいて Web サービス要求をマッピングするオプションを提供します。soapaction - メッセージのヘッダーに指定された SOAP アクションに基づいて Web サービス要求をマッピングするために使用されます。uri: 特定の URI をターゲットにする Web サービスリクエストをマッピングするのに使用します。xpathresult - 受信メッセージに対して XPath 式の評価に基づいて Web サービスリクエストをマッピングするために使用されます。評価の結果は、エンドポイント URI. beanname に指定された XPath 結果と一致する必要があります。これにより、PayloadRootQNameEndpointMapping、SoapActionEndpointMapping、SoapActionEndpointMapping などの既存の（レガシー）エンドポイントマッピングと統合するために org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher オブジェクトを参照できます。		EndpointMappingType
lookupKey	エンドポイントマッピングが使用される場合のエンドポイントマッピングキー		文字列
webServiceEndpointUri	プロデューサーに使用するデフォルトの Web サービスエンドポイント URI。		文字列

303.2.2. クエリーパラメーター（22 パラメーター）：

Name	説明	デフォルト	Type
messageFilter (common)	カスタム MessageFilter を提供するオプション。たとえば、ヘッダーや添付ファイルを処理する場合などです。		MessageFilter

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
endpointDispatcher (consumer)	Spring-WS によって受信されるメッセージを Camel エンドポイントにディスパッチする Spring org.springframework.ws.server.endpoint.MessageEndpoint。PayloadRootQNameEndpointMapping、SoapActionEndpointMapping などの既存の（レガシー）エンドポイントマッピングと統合します。		CamelEndpointDispatcher
endpointMapping (consumer)	Registry/ApplicationContext の org.apache.camel.component.spring.ws.bean.CamelEndpointMapping のインスタンスへの参照。すべての Camel/Spring-WS エンドポイントを提供するには、レジストリーで必要となる Bean は 1 つだけです。この Bean は MessageDispatcher によって自動検出され、エンドポイントに指定された特性（ルート QName、SOAP アクションなど）に基づいてリクエストを Camel エンドポイントにマッピングするために使用されます。		CamelSpringWSEndpoint Mapping
式 （コンシューマー）	type=xpathresult オプションに使用する XPath 式。その場合は、このオプションを設定する必要があります。		文字列
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
allowResponseAttachment Override (producer)	実際のサービスレイヤーからの添付/アウトされたエクスチェンジの soap 応答割り当てを上書きするオプション。呼び出されたサービスが、true に設定されている場合にこのオプションを追加または書き換える場合は、変更した soap アタッチメントをメッセージ添付で上書きできるようにします。	false	boolean

Name	説明	デフォルト	Type
allowResponseHeaderOverride (producer)	実際のサービスレイヤーからのヘッダー情報で、エクスチェンジの soap 応答ヘッダーを上書きするオプション。呼び出されたサービスが true に設定されている場合、このオプションに soap ヘッダーを追加または書き換える場合は、変更した soap ヘッダーをメッセージヘッダーで上書きできるようにします。	false	boolean
faultAction (producer)	メソッドによって提供される faultAction 応答 WS-Addressing Fault Action ヘッダーの値を表します。		URI
faultTo (producer)	メソッドによって提供される faultAction 応答 WS-Addressing FaultTo ヘッダーの値を表します。		URI
messageFactory (producer)	カスタム WebServiceMessageFactory を提供するオプション。たとえば、Apache Axiom で SAAJ の代わりに Web サービスメッセージを処理する場合などがこれに該当します。		WebServiceMessage Factory
messageIdStrategy (producer)	一意のメッセージ ID の生成を制御するカスタム MessageIdStrategy を提供するオプション。		MessageIdStrategy
messageSender (producer)	カスタム WebServiceMessageSender を提供するオプション。たとえば、認証の実行や代替トランスポートの使用		WebServiceMessage Sender
outputAction (producer)	メソッドが提供する応答 WS-Addressing Action ヘッダーの値を表します。		URI
replyTo (producer)	メソッドによって提供される replyTo 応答 WS-Addressing ReplyTo ヘッダーの値を表します。		URI
SOAPAction (producer)	SOAP action to include a SOAP request when accessing remote web services（リモート Web サービスへのアクセス時に SOAP リクエスト内に含める SOAP アクション）		文字列

Name	説明	デフォルト	Type
タイムアウト（プロデューサー）	プロデューサーを使用して web サービスを呼び出す間にソケットの読み取りタイムアウト（ミリ秒単位）を設定します。URLConnection.setReadTimeout（）および CommonsHttpMessageSender.setReadTimeout（）を参照してください。このオプションは、組み込みのメッセージ送信者の実装（CommonsHttpMessageSender および HttpUrlConnectionMessageSender）を使用すると機能します。これらの実装の 1 つは、コンポーネントに提供される Spring WS 設定オプションをカスタマイズしない限り、デフォルトで HTTP ベースのサービスに使用されます。標準以外の送信者を使用している場合は、独自のタイムアウト設定を処理することが想定されます。同梱のメッセージ送信者 HttpComponentsMessageSender は、非推奨となった CommonsHttpMessageSender の代わりに考慮されます。「HttpComponentsMessageSender.setReadTimeout（）」を参照してください。		int
webServiceTemplate (producer)	カスタム WebServiceTemplate を提供するオプション。これにより、カスタムインターセプターの追加やフォールトリゾルバー、メッセージ送信者、メッセージファクトリーの指定など、クライアント側の Web サービス処理を完全に制御できます。		WebServiceTemplate
wsAddressingAction (producer)	Web サービスへのアクセス時に追加する WS-Addressing 1.0 アクションヘッダー。To ヘッダーは、エンドポイント URI で指定される web サービスのアドレスに設定されます（Spring-WS のデフォルト動作）。		URI
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
sslContextParameters (security)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters

303.2.3. メッセージヘッダー

Name	タイプ	説明
CamelSpringWebserviceEndpointUri	文字列	クライアントとしてアクセスしている Web サービスの URI。エンドポイント URI の **アドレス** 部分を上書きします。
CamelSpringWebserviceSoapAction	文字列	メッセージの SOAP アクションを指定するヘッダー。存在する場合は **soapAction** オプションをオーバーライドします。
CamelSpringWebserviceSoapHeader	Source	**Camel 2.11.1:** このヘッダーを使用して、メッセージの SOAP ヘッダーを指定/アクセスします。
CamelSpringWebserviceAddressingAction	URI	このヘッダーを使用してメッセージの WS-Addressing アクションを指定し、**wsAddressingAction** オプションが存在する場合は上書きします。
CamelSpringWebserviceAddressingFaultTo	URI	このヘッダーを使用して WS-Addressing FaultTo を指定し、ある場合は faultTo オプションを上書きします。
CamelSpringWebserviceAddressingReplyTo	URI	このヘッダーを使用して WS-Addressing ReplyTo を指定し、存在する場合は replyTo オプションを上書きします。

Name	タイプ	説明
CamelSpringWebserviceAddressingOutputAction	URI	このヘッダーを使用して WS-Addressing Action を指定し、ある場合は outputAction オプションをオーバーライドします。
CamelSpringWebserviceAddressingFaultAction	URI	このヘッダーを使用して WS-Addressing Fault Action を指定し、ある場合は faultAction オプションを上書きします。

303.3. WEB サービスへのアクセス

http://foo.com/bar で Web サービスを呼び出すには、ルートを定義します。

```
from("direct:example").to("spring-ws:http://foo.com/bar")
```

そして、メッセージを送信しています：

```
template.requestBody("direct:example", "<foobar xmlns=\"http://foo.com\"><msg>test
message</msg></foobar>");
```

SOAP サービスを呼び出す必要がないことを忘れないでください。spring-WS は XML から SOAP のマーシャリングを実行します。

303.4. SOAP および WS-ADDRESSING アクションヘッダーの送信

リモート Web サービスが SOAP アクションを必要とする場合、または WS-Addressing 標準を以下のように定義します。

```
from("direct:example")
.to("spring-ws:http://foo.com/bar?
soapAction=http://foo.com&wsAddressingAction=http://bar.com")
```

オプションで、ヘッダー値でエンドポイントオプションを上書きできます。

```
template.requestBodyAndHeader("direct:example",
"<foobar xmlns=\"http://foo.com\"><msg>test message</msg></foobar>",
SpringWebserviceConstants.SPRING_WS_SOAP_ACTION, "http://baz.com");
```

303.5. SOAP ヘッダーの使用

Camel 2.11.1 から利用可能

spring-ws エンドポイントへメッセージを送信するときに SOAP ヘッダーを Camel Message ヘッダーとして提供できます。たとえば、以下の SOAP ヘッダーが String に指定されます。

```
String body = ...
String soapHeader = "<h:Header xmlns:h=\"http://www.webserviceX.NET/\">
<h:MessageID>1234567890</h:MessageID><h:Nested><h:NestedID>1111</h:NestedID>
</h:Nested></h:Header>";
```

以下のように、Camel Message にボディーとヘッダーを設定できます。

```
exchange.getIn().setBody(body);
exchange.getIn().setHeader(SpringWebserviceConstants.SPRING_WS_SOAP_HEADER,
soapHeader);
```

次に、Exchange を spring-ws エンドポイントに送信し、Web サービスを呼び出します。

同様に、spring-ws コンシューマーも Camel Message を SOAP ヘッダーで補完します。

例は、この ユニットテスト を参照してください。

303.6. ヘッダーおよび添付の伝播

Spring WS Camel は、バージョン 2.10.3 以降の Spring-WS WebServiceMessage 応答へのヘッダーおよび添付の伝搬をサポートします。エンドポイントは、MessageFilter（デフォルト実装は BasicMessageFilter によって提供）と呼ばれるように「hook」を使用して、エクスチェンジヘッダーと添付ファイルを WebServiceMessage 応答に伝播します。では、以下を使用できます。

```
exchange.getOut().getHeaders().put("myCustom","myHeaderValue")
exchange.getIn().addAttachment("myAttachment", new DataHandler(...))
```

注記： パイプラインのエクスチェンジヘッダーにテキストが含まれる場合は、soap ヘッダーに Qname(key)=value 属性を生成します。QName クラスを直接作成し、キーをヘッダーに配置することが推奨されます。

303.7. スタイルシートを使用して SOAP ヘッダーを変換する方法

ヘッダー変換フィルター(HeaderTransformationMessageFilter.java)を使用して、soap 要求の soap ヘッダーを変換できます。ヘッダー変換フィルターを使用する場合は、以下の例を参照してください。

```
<bean id="headerTransformationFilter"
class="org.apache.camel.component.spring.ws.filter.impl.HeaderTransformationMessageFilter">
    <constructor-arg index="0" value="org/apache/camel/component/spring/ws/soap-header-transform.xslt"/>
</bean
```

camel エンドポイントで上記で定義された bead を使用する

```
<route>
    <from uri="direct:stockQuoteWebserviceHeaderTransformation"/>
    <to uri="spring-ws:http://localhost?webServiceTemplate=#webServiceTemplate&amp;soapAction=http://www.stockquotes.edu/GetQuote&amp;messageFilter=#headerTransformationFilter"/>
</route>
```

303.8. MTOM アタッチメントの使用方法

BasicMessageFilter は、MTOM メッセージを生成するために Apache Axiom に必要なすべての情報を提供します。Apache Axiom 内で Apache Camel Spring WS を使用する場合は、以下のようになります。- messageFactory を bellow として定義し、Spring-WS は MTOM ストラテジーを使用して SOAP メッセージに最適化された添付ファイルを設定します。

```
<bean id="axiomMessageFactory"
class="org.springframework.ws.soap.axiom.AxiomSoapMessageFactory">
<property name="payloadCaching" value="false" />
<property name="attachmentCaching" value="true" />
<property name="attachmentCacheThreshold" value="1024" />
</bean>
```

- *pom.xml に以下の依存関係を追加します。*

```
<dependency>
<groupId>org.apache.ws.commons.axiom</groupId>
<artifactId>axiom-api</artifactId>
<version>1.2.13</version>
</dependency>
<dependency>
<groupId>org.apache.ws.commons.axiom</groupId>
<artifactId>axiom-impl</artifactId>
<version>1.2.13</version>
<scope>runtime</scope>
</dependency>
```

- *プロセッサーの実装などを使用して、アタッチメントをパイプラインに追加します。*

```
private class Attachement implements Processor {
public void process(Exchange exchange) throws Exception
{ exchange.getOut().copyFrom(exchange.getIn()); File file = new File("testAttachment.txt");
exchange.getOut().addAttachment("test", new DataHandler(new FileDataSource(file)));  }
}
```

- *エンドポイント（プロデューサー）をユーザ名として定義します。以下に例を示します。*

```
from("direct:send")
.process(new Attachement())
.to("spring-ws:http://localhost:8089/mySoapService?
soapAction=mySoap&messageFactory=axiomMessageFactory");
```

- *これで、プロデューサーは otpmized attachments で MTOM メッセージを生成します。*

303.9. カスタムヘッダーおよび添付のフィルター

ヘッダーまたは添付のいずれかのカスタム処理を提供する必要がある場合は、既存の BasicMessageFilter を拡張し、適切なメソッドを上書きするか、MessageFilter インターフェースのブランドの新しい実装を記述します。
カスタムフィルターを使用するには、以下を Spring コンテキストに追加します。

グローバルメッセージフィルターまたはローカルメッセージフィルターのいずれかを以下のように指定することができます： a）すべての Spring-WS エンドポイントのグローバル設定を提供するグローバルカスタムフィルターを指定できます。

```
<bean id="messageFilter" class="your.domain.myMessageFiler" scope="singleton" />
```

または、以下のようにローカルの messageFilter をエンドポイントに直接指定します。

```
to("spring-ws:http://yourdomain.com?messageFilter=#myEndpointSpecificMessageFilter");
```

詳細は「CAMEL-5724」を参照してください。

独自の MessageFilter を作成する場合は、BasicMessageFilter クラスの MessageFilter のデフォルト実装で以下のメソッドを上書きすることを検討してください。

```
protected void doProcessSoapHeader(Message inOrOut, SoapMessage soapMessage)
{your code /*no need to call super*/ }

protected void doProcessSoapAttachements(Message inOrOut, SoapMessage response)
{ your code /*no need to call super*/ }
```

303.10. カスタム MESSAGESENDER および MESSAGEFACTORY の使用

レジストリーのカスタムメッセージ送信側またはファクトリーは、以下のように参照できます。

```
from("direct:example")
.to("spring-ws:http://foo.com/bar?
messageFactory=#messageFactory&messageSender=#messageSender")
```

Spring の設定：

```
<!-- authenticate using HTTP Basic Authentication -->
<bean id="messageSender"
class="org.springframework.ws.transport.http.HttpComponentsMessageSender">
    <property name="credentials">
        <bean class="org.apache.commons.httpclient.UsernamePasswordCredentials">
            <constructor-arg index="0" value="admin"/>
            <constructor-arg index="1" value="secret"/>
        </bean>
    </property>
</bean>

<!-- force use of Sun SAAJ implementation, http://static.springsource.org/spring-
ws/sites/1.5/faq.html#saaj-jboss -->
<bean id="messageFactory" class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">
    <property name="messageFactory">
        <bean
```

```
class="com.sun.xml.messaging.saaj.soap.ver1_1.SOAPMessageFactory1_1Impl"></bean>
    </property>
</bean>
```

303.11. WEB サービスの公開

このコンポーネントを使用して Web サービスを公開するには、最初に MessageDispatcher を設定して Spring XML ファイルでエンドポイントマッピングを検索する必要があります。サーブレットコンテナー内での実行を計画する場合は、web.xml で設定された MessageDispatcherServlet の使用が必要になる可能性があります。

デフォルトでは、MessageDispatcherServlet は /WEB-INF/spring-ws-servlet.xml という名前の Spring XML を検索します。Spring-WS で Camel を使用するには、その XML ファイルの唯一の必須 Bean は CamelEndpointMapping です。この Bean により、MessageDispatcher は Web サービスリクエストをルートにディスパッチできます。

web.xml

```
<web-app>
    <servlet>
        <servlet-name>spring-ws</servlet-name>
        <servlet-class>org.springframework.ws.transport.http.MessageDispatcherServlet</servlet-
class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring-ws</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

spring-ws-servlet.xml

```
<bean id="endpointMapping"
class="org.apache.camel.component.spring.ws.bean.CamelEndpointMapping" />

<bean id="wsdl" class="org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition">
    <property name="schema">
        <bean class="org.springframework.xml.xsd.SimpleXsdSchema">
            <property name="xsd" value="/WEB-INF/foobar.xsd"/>
        </bean>
    </property>
    <property name="portTypeName" value="FooBar"/>
    <property name="locationUri" value="/"/>
    <property name="targetNamespace" value="http://example.com/"/>
</bean>
```

Spring-WS の設定に関する詳細は、「Con Writing Contract-First Web Services 」を参照してください。基本的には、段落 3.6 の「エンドポイントの実装」は、このコンポーネントによって処理されます（特に段落 3.6.2 「Routing the Message to the Endpoint」は、CamelEndpointMapping が利用されます）。また、Camel ディストリビューションに含まれる Spring Web サービスの例を忘れないでください。

303.12. ルートのエンドポイントマッピング

XML 設定インプレースで、Camel の DSL を使用してエンドポイントによって処理される Web サービスリクエストを定義できるようになりました。

以下のルートは、http://example.com/ namespace 内に「GetFoo」という名前のルート要素を持つすべての Web サービス要求を受信します。

```
from("spring-ws:rootqname:{http://example.com/}GetFoo?
endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

以下のルートは、http://example.com/GetFoo SOAP アクションが含まれる Web サービスリクエストを受信します。

```
from("spring-ws:soapaction:http://example.com/GetFoo?
endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

以下のルートは、http://example.com/foobar に送信されたすべてのリクエストを受信します。

```
from("spring-ws:uri:http://example.com/foobar?endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

以下のルートは、メッセージ内の任意の場所（およびデフォルトの namespace）の要素 < foobar>abc </foobar> が含まれる要求を受信します。

```
from("spring-ws:xpathresult:abc?
expression=//foobar&endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

303.13. 既存のエンドポイントマッピングを使用した代替設定

Registry/ApplicationContext には、対応する 名前の CamelEndpointDispatcher タイプの 1 つの

bean が mapping-type bean が設定されたすべてのエンドポイントに対して必要になります。この Bean は、Camel エンドポイントと PayloadRootQNameEndpointMapping などの既存の エンドポイントマッピング の間のブリッジとして機能します。

注記： beanname mapping-type の使用は、主に Spring-WS を使用し、Spring XML ファイルにエンドポイントマッピングが定義されている（レガシー）状況を対象としています。beanname マッピングタイプを使用すると、Camel ルートを既存のエンドポイントマッピングに接続できます。ゼロから作業を開始する場合、必要少なく、より表現的なため、エンドポイントマッピングを Camel URI（endpointMappingを参照）として定義することが推奨されます。または、アノテーションを用いて vanilla Spring-WS を使用できます。

Beanname を使用するルートの例：

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="spring-ws:beanname:QuoteEndpointDispatcher" />
        <to uri="mock:example" />
    </route>
</camelContext>

<bean id="legacyEndpointMapping"
class="org.springframework.ws.server.endpoint.mapping.PayloadRootQNameEndpointMapping">
    <property name="mappings">
        <props>
            <prop key="{http://example.com/}GetFuture">FutureEndpointDispatcher</prop>
            <prop key="{http://example.com/}GetQuote">QuoteEndpointDispatcher</prop>
        </props>
    </property>
</bean>

<bean id="QuoteEndpointDispatcher"
class="org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher" />
<bean id="FutureEndpointDispatcher"
class="org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher" />
```

303.14. POJO（アン）のマーシャリング

Camel のプラグ可能なデータ形式は、JAXB、XStream、JibX、Castor、XMLBeans などのライブラリーを使用した pojo/xml マーシャリングのサポートを提供します。これらのデータフォーマットをルートで使用して、pojo の送受信が可能です。

Web サービスにアクセス する際には、リクエストをマーシャリングし、レスポンスメッセージをアンマーシャリングすることができます。

```
JaxbDataFormat jaxb = new JaxbDataFormat(false);
```

```
jaxb.setContextPath("com.example.model");

from("direct:example").marshal(jaxb).to("spring-ws:http://foo.com/bar").unmarshal(jaxb);
```

Web サービスを提供する場合と同様に、POJO に XML リクエストをアンマーシャリングし、レスポンスメッセージを XML にマーシャリングすることができます。

```
from("spring-ws:rootqname:{http://example.com/}GetFoo?
endpointMapping=#endpointMapping").unmarshal(jaxb)
.to("mock:example").marshal(jaxb);
```

303.15. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第304章 SQL コンポーネント

Camel バージョン 1.4 で利用可能

sql: コンポーネントを使用すると、JDBC クエリーを使用してデータベースを操作することができます。このコンポーネントと JDBC コンポーネントの相違点は、クエリーがエンドポイントのプロパティーであり、クエリーに渡されるパラメーターとしてメッセージペイロードを使用するためです。

このコンポーネントは、実際の SQL 処理に背後で spring-jdbc を使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sql</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

SQL コンポーネントは以下もサポートします。

- Idempotent Consumer EIP パターンの JDBC ベースのリポジトリー。詳細は以下を参照してください。
- Aggregator EIP パターンの JDBC ベースのリポジトリー。詳細は以下を参照してください。

304.1. URI 形式

警告： Camel 2.11 以降では、このコンポーネントからコンシューマー（例： from（）） およびプロデューサーエンドポイント（例： to（）） の両方を作成できます。以前のバージョンでは、プロデューサーとしてのみ動作できました。

INFO: このコンポーネントは、Transactional Client として使用できます。

SQL コンポーネントは、以下のエンドポイント URI 表記を使用します。

```
sql:select * from table where id=# order by name[?options]
```

Camel 2.11 以降では、以下のように :'#name_of_the_parameter' スタイルを使用すると名前付きパラメーターを使用できます。

```
sql:select * from table where id=:#myId order by name[?options]
```

名前付きパラメーターを使用する場合、Camel は指定の優先順位で名前を検索します。
1. メッセージヘッダーから java.util.Map
2. からメッセージボディーから名前を検索します。

名前付きパラメーターを解決できない場合、例外が発生します。

Camel 2.14 以降 では、以下のように Simple 式をパラメーターとして使用できます。

```
sql:select * from table where id=:#${property.myId} order by name[?options]
```

SQL クエリーのパラメーターを示す標準の ? 記号は、エンドポイントのオプションを指定するために ? 記号が使用されるため、# 記号に置き換えられます。? 記号の置換はエンドポイントベースに設定できます。

Camel 2.17 以降では、以下のように SQL クエリーをクラスパスまたはファイルシステムのファイルに外部化できます。

```
sql:classpath:sql/myquery.sql[?options]
```

myquery.sql ファイルはクラスパスにあり、プレーンテキストです。

```
-- this is a comment
select *
from table
where
  id = :#${property.myId}
order by
  name
```

ファイルで複数行を使用し、必要に応じて SQL をフォーマットできます。また、- dash 行などのコメントも使用します。

URI にクエリーオプションを追加するには、?option=value&option=value&...

304.2. オプション

SQL コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
dataSource (common)	データベースとの通信に使用する DataSource を設定します。		DataSource
usePlaceholder (advanced)	プレースホルダーを使用するかどうかを設定し、すべてのプレースホルダー文字を SQL クエリーの署名に置き換えるかどうかを設定します。このオプションはデフォルトの true です。	true	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SQL エンドポイントは、URI 構文を使用して設定されます。

```
sql:query
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

304.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
query	実行する SQL クエリーの設定が **必要です**。file: または classpath: をプレフィックスとして使用し、ファイルの場所を指定することで、クエリーを外部化できます。		文字列

304.2.2. クエリーパラメーター（45 パラメーター）：

Name	説明	デフォルト	Type
allowNamedParameters (common)	クエリーで名前付きパラメーターを使用できるかどうか。	true	boolean
dataSource (common)	データベースとの通信に使用する DataSource を設定します。		DataSource
dataSourceRef (common)	**非推奨**。データベースとの通信に使用する DataSource への参照をレジストリーから参照するように設定します。		文字列
outputClass (common)	outputType=SelectOne の変換として使用する完全なパッケージおよびクラス名を指定します。		文字列
outputHeader (common)	クエリー結果をメッセージのボディーの代わりにヘッダーに保存します。デフォルトでは、outputHeader == null とクエリー結果はメッセージボディーに保存され、メッセージボディー内の既存のコンテンツは破棄されます。outputHeader が設定されている場合、値はクエリー結果を保存するヘッダーの名前として使用され、元のメッセージボディーは保持されます。		文字列
outputType (common)	コンシューマーまたはプロデューサーの出力を Map の List として選択するか、または以下のように SelectOne を単一の Java オブジェクトとして選択します。a）クエリーに列が 1 つしかない場合は、JDBC Column オブジェクトが返されます（SELECT COUNT（）FROM PROJECT は Long オブジェクトを返します。b）クエリーに複数の列がある場合、その結果の Map を返します。c）outputClass が設定されている場合は、その結果のマップを返します。 次に、列名に一致するすべてのセッターを呼び出して、クエリー結果を Java Bean オブジェクトに変換します。これは、クラスにインスタンスを作成するためのデフォルトのコンストラクターを持つと仮定します。d）クエリーが複数の行で作成された場合は、一意でない結果例外をスローします。	SelectList	SqlOutputType
セパレーター（共通）	パラメーター値がメッセージボディー（本文が String タイプである場合）から取得されるときに使用する区切り文字。プレースホルダーに挿入されます。名前付きパラメーターを使用する場合は、代わりに Map タイプが使用されます。デフォルト値は comma です。	,	char

Name	説明	デフォルト	Type
breakBatchOnConsumeFail (consumer)	消費に失敗した場合にバッチを壊すかどうかを設定します。	false	boolean
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
expectedUpdateCount (consumer)	onConsume の使用時に、予想される更新数を検証するように設定します。	-1	int
maxMessagesPerPoll (consumer)	ポーリングするメッセージの最大数を設定します。		int
onConsume (consumer)	各行の処理後、Exchange が正常に処理されたときにこのクエリーを実行できます。たとえば、行を処理済みとしてマークします。クエリーにはパラメーターを指定できます。		文字列
onConsumeBatchComplete (consumer)	バッチ全体を処理したら、このクエリーを実行して行を一括更新することができます。クエリーにパラメーターを含めることはできません。		文字列
onConsumeFailed (consumer)	各行の処理後、エクスチェンジが失敗した場合、行の失敗をマークするなど、このクエリーを実行できます。クエリーにはパラメーターを指定できます。		文字列
routeEmptyResultSet (consumer)	空の結果セットを次のホップに送信できるようにするかどうかを設定します。デフォルトは false です。そのため、空の結果セットが除外されます。	false	boolean
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
トランザクション（コンシューマー）	トランザクションを有効または無効にします。有効にすると、エクスチェンジの処理に失敗した場合、コンシューマーは追加のエクスチェンジの処理を中断してロールバック Eager をトリガーします。	false	boolean

Name	説明	デフォルト	Type
useIterator (consumer)	resultset をルートに配信する方法を設定します。配信をリストまたは個別オブジェクトのいずれかとして指定します。デフォルトは true です。	true	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
processingStrategy (consumer)	コンシューマーが行/バッチを処理した場合に、プラグインでカスタムの org.apache.camel.component.sql.SqlProcessingStrategy を使用してクエリーを実行できます。		SqlProcessingStrategy
batch （プロデューサー）	バッチモードを有効または無効にします。	false	boolean
noop (producer)	設定されている場合、SQL クエリーの結果を無視し、既存の IN メッセージを処理の継続に OUT メッセージとして使用します。	false	boolean
useMessageBodyForSql (producer)	メッセージボディーを SQL として使用し、パラメーターのヘッダーとして使用するかどうか。このオプションを有効にすると、URI の SQL は使用されません。	false	boolean

Name	説明	デフォルト	Type
alwaysPopulateStatement (producer)	有効にすると、org.apache.camel.component.sql.SqlPrepareStatementStrategy の populateStatement メソッドは常に呼び出されます。また、準備が予想されるパラメーターがない場合も常に呼び出されます。false の場合、1 つ以上の想定パラメーターが設定されている場合にのみ populateStatement が呼び出されます。たとえば、パラメーターを指定せずに SQL クエリーのメッセージボディー/ヘッダーを読み取ることが回避されます。	false	boolean
parametersCount (producer)	ゼロより大きい場合、Camel は JDBC メタデータ API 経由でクエリーを実行する代わりに、このパラメーター値を使用して置き換えるパラメーターの数を使用します。これは、JDBC ベンダーが正しいパラメーター数を返しないとユーザーが上書きできる場合に便利です。		int
placeholder (advanced)	SQL クエリーで置き換える文字を指定します。これは単純な String.replaceAll（）操作であり、SQL 解析が行われないことに注意してください（引用符付きの文字列も変更されます）。	#	文字列
prepareStatementStrategy (advanced)	プラグインがカスタムの org.apache.camel.component.sql.SqlPrepareStatementStrategy を使用して、クエリーおよび準備済みステートメントの準備を制御できます。		SqlPrepareStatement Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
templateOptions (advanced)	マップのキー/値で Spring JdbcTemplate を設定します。		マップ
usePlaceholder (advanced)	プレースホルダーを使用するかどうかを設定し、すべてのプレースホルダー文字を SQL クエリーの署名に置き換えるかどうかを設定します。このオプションはデフォルトの true です。	true	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int

Name	説明	デフォルト	Type
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit

Name	説明	デフォルト	Type
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

304.3. メッセージボディーの処理

SQL コンポーネントはメッセージボディーを java.util.Iterator タイプのオブジェクトに変換しようとし、このイテレーターを使用してクエリーパラメーター（各クエリーパラメーターがエンドポイント URI の # 記号（または設定されたプレースホルダー）で表される）を入力します。メッセージボディーが配列またはコレクションではない場合、変換により、ボディー自体である 1 つのオブジェクトのみを繰り返すイテレーターが生成されます。

たとえば、メッセージボディーが java.util.List のインスタンスである場合、リストの最初の項目は SQL クエリーの最初の # に置換されます。

batch が true に設定されている場合、インバウンドメッセージボディーの解釈は、パラメーターイテレーターが含まれるイテレーターの代わりに若干変わります。コンポーネントは、パラメーターイテレーターを含むイテレーターを想定します。外部イテレーターのサイズはバッチサイズを決定します。

Camel 2.16 以降では、メッセージボディーを SQL ステートメントとして使用できるようにする useMessageBodyForSql オプションを使用し、SQL パラメーターは SqlConstants.SQL_PARAMETERS キーのあるヘッダーに指定する必要があります。これにより、SQL クエリーがメッセージボディーからのものであるため、SQL コンポーネントはより動的に動作します。

304.4. クエリーの結果

選択 する操作の場合、結果は JdbcTemplate.queryFor List（）メソッドによって返される List <Map<String, Object >> タイプのインスタンスになります。更新 操作の場合、結果は更新された行数で、整数として返され ます。

デフォルトでは、結果はメッセージボディーに配置されます。 outputHeader パラメーターを設定すると、結果はヘッダーに配置されます。 これは、完全なメッセージ補完パターンを使用してヘッダーを追加する代わりに、シーケンスや他の小さな値をヘッダーにクエリーするための簡潔な構文を提供します。 outputHeader と outputType を一緒に使用すると便利です。

```
from("jms:order.inbox")
    .to("sql:select order_seq.nextval from dual?
```

```
outputHeader=OrderId&outputType=SelectOne")
    .to("jms:order.booking");
```

304.5. STREAMLIST の使用

Camel 2.18 以降、プロデューサーはイテレーターを使用してクエリーの出力をストリーミングする outputType=StreamList をサポートします。これにより、Splitter EIP が各行を一度に処理し、必要に応じてデータベースからデータをロードできるストリーミング方式でデータを処理できます。

```
from("direct:withSplitModel")
        .to("sql:select * from projects order by id?
outputType=StreamList&outputClass=org.apache.camel.component.sql.ProjectModel")
        .to("log:stream")
        .split(body()).streaming()
            .to("log:row")
            .to("mock:result")
        .end();
```

304.6. ヘッダーの値

更新 操作の実行時に、SQL コンポーネントは更新数を以下のメッセージヘッダーに保存します。

ヘッダー	説明
CamelSqlUpdateCount	**Integer** オブジェクトとして返される、**更新** 操作用に更新された行数。outputType=StreamList を使用する場合、このヘッダーは提供されません。
CamelSqlRowCount	**Integer** オブジェクトとして返される、**選択** 操作のために返される行の数。outputType=StreamList を使用する場合、このヘッダーは提供されません。
CamelSqlQuery	Camel 2.8: 実行するクエリー。このクエリーは、エンドポイント URI で指定されたクエリーよりも優先されます。ヘッダーのクエリーパラメーターは、**#** 記号で **は** なく **?** で表されることに注意してください。

挿入 操作の実行時に、SQL Component は生成されたキーおよびこれらの行番号を以下のメッセージヘッダーに格納します（Camel 2.12.4、2.13.1 の時点で利用可能）。

ヘッダー	説明
CamelSqlGeneratedKeysRowCount	生成されたキーが含まれるヘッダーの行数。
CamelSqlGeneratedKeyRows	生成されたキー（キーのマッピングのリスト）が含まれる行。

304.7. 生成される鍵

** Camel 2.12.4、2.13.1、および 2.14 の時点で利用可能 **

SQL INSERT を使用してデータを挿入すると、automatic が自動生成された鍵をサポートする可能性があります。SQL プロデューサーに対し、ヘッダーで生成されたキーを返すように指示できます。そのためには、ヘッダー CamelSqlRetrieveGeneratedKeys=true を設定します。次に、生成されたキーは上記の表に記載されているキーと共にヘッダーとして提供されます。

この ユニットテスト で詳細を確認できます。

304.8. 設定

URI で DataSource への参照を直接設定できるようになりました。

```
sql:select * from table where id=# order by name?dataSource=myDS
```

304.9. 例

以下の例では、クエリーを実行して行の 一覧 として結果を取得します。各行は Map<String, Object で、キーは列名です。

まず、サンプルに使用するテーブルを設定します。これはユニットテストをベースとしているため、java で行います。

SQL スクリプト createAndPopulateDatabase.sql は、以下のように実行します。

次に、ルートと sql コンポーネントを設定します。sql エンドポイントの前に direct エンドポイントを使用していることに注意してください。これにより、URI(direct:simple)で ダイレクト エンドポイントにエクスチェンジを送信できます。これは、クライアントが長い sql: URI よりも簡単に使用できます。DataSource はレジストリーで検索されるため、標準の Spring XML を使用して DataSource を設定できます。

そして、メッセージをデータベースをクエリーする sql コンポーネントにルーティングする direct エンドポイントで実行します。

以下のように Spring XML で DataSource を設定できます。

```
<jee:jndi-lookup id="myDS" jndi-name="jdbc/myDataSource"/>
```

304.9.1. 名前付きパラメーターの使用

Camel 2.11 から利用可能

以下の指定のルートでは、プロジェクトテーブルからすべてのプロジェクトを取得します。SQL クエリーにはパラメーター :#lic と :#min の 2 つの名前があることに留意してください。
その後、Camel はメッセージボディーまたはメッセージヘッダーからこれらのパラメーターを検索します。上記の例では、名前付きパラメーターに定数値
を使用して 2 つのヘッダーを設定しています。

```
from("direct:projects")
  .setHeader("lic", constant("ASF"))
  .setHeader("min", constant(123))
  .to("sql:select * from projects where license = :#lic and id > :#min order by id")
```

メッセージボディーが java.util.Map の場合、名前付きパラメーターはボディーから取得されます。

```
from("direct:projects")
  .to("sql:select * from projects where license = :#lic and id > :#min order by id")
```

304.9.2. 式パラメーターの使用

Camel 2.14 から利用可能

以下の指定のルートでは、データベースからすべてのプロジェクトを取得します。これはエクスチェンジのボディーを使用してライセンスを定義し、2 番目のパラメーターとしてプロパティーの値を使用します。

```
from("direct:projects")
  .setBody(constant("ASF"))
  .setProperty("min", constant(123))
  .to("sql:select * from projects where license = :#${body} and id > :#${property.min} order by id")
```

304.9.3. 動的値での IN クエリーの使用

Camel 2.17 から利用可能

Camel 2.17 以降では、SQL プロデューサーは IN 値が動的に計算される IN ステートメントで SQL クエリーを使用できます。たとえば、メッセージボディーやヘッダーなどです。

IN を使用するには、以下を行う必要があります。

- パラメーター名 の前に以下を付けます。
- パラメーターの周りの追加 （）

例として、これが改善されています。以下のクエリーが使用されます。

```
-- this is a comment
select *
from projects
where project in (:#in:names)
order by id
```

以下のルートで以下を行います。

```
from("direct:query")
   .to("sql:classpath:sql/selectProjectsIn.sql")
   .to("log:query")
   .to("mock:query");
```

次に IN クエリーはキー名を持つヘッダーを動的値で使用することができます。以下に例を示します。

```
// use an array
template.requestBodyAndHeader("direct:query", "Hi there!", "names", new String[]{"Camel",
"AMQ"});


// use a list
List<String> names = new ArrayList<String>();
names.add("Camel");
names.add("AMQ");

template.requestBodyAndHeader("direct:query", "Hi there!", "names", names);


// use a string separated values with comma
template.requestBodyAndHeader("direct:query", "Hi there!", "names", "Camel,AMQ");
```

クエリーを外部化する代わりにエンドポイントで指定することもできます（外部化により SQL クエリーの管理が容易になります）。

```
from("direct:query")
    .to("sql:select * from projects where project in (:#in:names) order by id")
    .to("log:query")
    .to("mock:query");
```

304.10. JDBC ベースのべき等リポジトリーの使用

Camel 2.7 で利用可能： このセクションでは、JDBC ベースのべき等リポジトリーを使用し ます。

TIP:*Abstract class* From Camel 2.9 以降では、カスタム JDBC のべき等リポジトリーを作成するために拡張できる抽象クラス org.apache.camel.processor.idempotent.jdbc.AbstractJdbcMessageIdRepository があります。

まず、べき等リポジトリーによって使用されるデータベーステーブルを作成する必要があります。Camel 2.7 では、以下のスキーマを使用します。

```
CREATE TABLE CAMEL_MESSAGEPROCESSED ( processorName VARCHAR(255),
messageId VARCHAR(100) )
```

Camel 2.8 では、作成されたAt 列を追加しました。

```
CREATE TABLE CAMEL_MESSAGEPROCESSED ( processorName VARCHAR(255),
messageId VARCHAR(100), createdAt TIMESTAMP )
```

警告： SQL Server TIMESTAMP タイプは、固定長のバイナリー文字列タイプです。JDBC 時間タイプ DATE、TIME、または TIMESTAMP にはマッピングされません。

JdbcMessageIdRepository のカスタマイズ

Camel 2.9.1 以降、必要に応じて org.apache.camel.processor.idempotent.jdbc.JdbcMessageIdRepository を調整するいくつかのオプションがあります。

パラメーター	デフォルト値	説明
createTableIfNotExists	true	Camel が存在しない場合に、テーブルを作成しようとするかどうかを定義します。
tableExistsString	CAMEL_MESSAGEPROCESSED から 1 を選択します。ここで、1=0	このクエリーは、テーブルがすでに存在しているかどうかを確認するために使用されます。テーブルが存在しないことを示すために例外をスローする必要があります。

パラメーター	デフォルト値	説明
createString	CREATE TABLE CAMEL_MESSAGEPROCESSED（processorName VARCHAR(255), messageId VARCHAR(100), createdAt TIMESTAMP）	テーブルの作成に使用されるステートメント。
queryString	SELECT COUNT(*) FROM CAMEL_MESSAGEPROCESSED WHERE processorName = ? AND messageId = ?	メッセージがリポジトリーにすでに存在するかどうかを判断するために使用されるクエリー（結果は「0」と等しくありません）。これは 2 つのパラメーターを取ります。この最初の 1 つはプロセッサー名（**文字列**）で、2 つ目はメッセージ ID（**文字列**）です。

パラメーター	デフォルト値	説明
insertString	INSERT INTO CAMEL_MESSAGEPROCESSED(processorName, messageId, createdAt)VALUES(?, ?, ?)	テーブルにエントリーを追加するために使用されるステートメント。3 つのパラメーターを取ります。1 つ目はプロセッサー名（**文字列**）で、2 つ目はメッセージ ID（**文字列**）で、3 つ目はこのエントリーがリポジトリーに追加される際のタイムスタンプ（**java.sql.Timestamp**)です。
deleteString	DELETE FROM CAMEL_MESSAGEPROCESSED WHERE processorName = ? AND messageId = ?	データベースからエントリーを削除するために使用されるステートメント。これには 2 つのパラメーターが使用されます。この最初の 1 つはプロセッサー名（**文字列**）で、2 つ目はメッセージ ID（**文字列**）です。

JDBC ベースの集約リポジトリーの使用

Camel 2.6 で利用可能

INFO: Using JdbcAggregationRepository in Camel 2.6

Camel 2.6 では、JdbcAggregationRepository は camel-jdbc-aggregator コンポーネントで提供されます。Camel 2.7 以降、JdbcAggregationRepository は camel-sql コンポーネントで提供されます。

JdbcAggregationRepository は AggregationRepository で、集約されたメッセージを即座に永続化します。これにより、デフォルトのアグリゲーターは AggregationRepository のみのメモリーで使用されるため、メッセージを緩めることはありません。JdbcAggregationRepository を使用すると、Camel とともに Aggregator の永続的なサポートを提供できます。

Exchange が正常に処理された場合にのみ、完了とマークされます。これは、確認 メソッドが AggregationRepository で呼び出されると発生します。つまり、同じエクスチェンジが再び失敗すると、成功するまで再試行されます。

maximumRedeliveries オプションを使用して、特定のリカバリーされたエクスチェンジの再配信の最大試行回数を制限できます。また、maximumRedeliveries に達したときに Camel が Exchange を送信する場所を認識できるように deadLetterUri オプションも設定する必要があります。

この テストなど、camel-sql のユニットテストにはいくつかの例があります。

データベース

稼働にするために、各Aggregator は集約と完了の 2 つのテーブルを使用します。規則により、完了した名前は 「_COMPLETED」 のサフィックスが付いた集約名と同じです。名前は、RepositoryName プロパティーで Spring Bean で設定する必要があります。以下の例では、集約が使用されます。

両方のテーブルのテーブル構造の定義は同じです。いずれの場合も、String 値はキー(id)として使用されますが、B Blob にはバイトアレイでシリアライズされたエクスチェンジが含まれます。
ただし、1 つの違いに注意してください。id フィールドには、テーブルによっては同じコンテンツがありません。
集約テーブル ID では、メッセージを集約するためにコンポーネントによって使用される相関 ID を保持します。完了したテーブルの id は、対応する blob フィールドに保存されたエクスチェンジの ID を保持します。

これは、テーブルの作成に使用される SQL クエリーです。「aggregation」 を、お使いのアグリゲーターリポジトリー名に置き換えます。

```
CREATE TABLE aggregation ( id varchar(255) NOT NULL, exchange blob NOT
NULL, constraint aggregation_pk PRIMARY KEY (id) ); CREATE TABLE
aggregation_completed ( id varchar(255) NOT NULL, exchange blob NOT
NULL, constraint aggregation_completed_pk PRIMARY KEY (id) );
```

本文とヘッダーのテキストとしての保存

Camel 2.11 から利用可能

JdbcAggregationRepository を設定して、メッセージボディーを保存し、別の列で String として選択することができます。たとえば、本文を保存し、以下の 2 つのヘッダー companyName および accountName を保存するには、以下の SQL を使用します。

```
CREATE TABLE aggregationRepo3 ( id varchar(255) NOT NULL, exchange blob
NOT NULL, body varchar(1000), companyName varchar(1000), accountName
varchar(1000), constraint aggregationRepo3_pk PRIMARY KEY (id) ); CREATE
TABLE aggregationRepo3_completed ( id varchar(255) NOT NULL, exchange
blob NOT NULL, body varchar(1000), companyName varchar(1000),
accountName varchar(1000), constraint aggregationRepo3_completed_pk
PRIMARY KEY (id) );
```

次に、以下のようにこの動作を有効にするためにリポジトリーを設定します。

```
<bean id="repo3"
class="org.apache.camel.processor.aggregate.jdbc.JdbcAggregationRepository">
<property name="repositoryName" value="aggregationRepo3"/> <property
name="transactionManager" ref="txManager3"/> <property name="dataSource"
ref="dataSource3"/> <!-- configure to store the message body and
following headers as text in the repo --> <property
name="storeBodyAsText" value="true"/> <property
name="headersToStoreAsText"> <list> <value>companyName</value>
<value>accountName</value> </list> </property> </bean>
```

Codec（シリアライズ）

どの種類のペイロードも含まれる可能性があるため、Exchange は設計でシリアライズできません。これは、データベース BLOB フィールドに保存されるバイト配列に変換されます。これらの変換はすべて JdbcCodec クラスによって処理されます。コードの詳細は、ClassLoadingAwareObjectInputStream に注意してください。

ClassLoadingAwareObjectInputStream は Apache ActiveMQ プロジェクトから再利用されました。ObjectInputStream をラップし、現在のThread ではなく ContextClassLoader で使用します。この利点は、他のバンドルによって公開されるクラスをロードできることです。これにより、エクスチェンジボディーおよびヘッダーにカスタムタイプオブジェクト参照を持たせることができます。

Transaction

トランザクションのオーケストレーションには Spring PlatformTransactionManager が必要です。

サービス(Start/Stop)

start メソッドは、データベースの接続と、必要なテーブルが存在することを確認します。何らかの問題が発生した場合は、起動時に失敗します。

Aggregator の設定

対象の環境によっては、Aggregator に一部の設定が必要になる場合があります。すでに分かるように、各アグリゲーターには、独自のリポジトリー（データベース内に作成されたテーブルのペアあり）とデータソースが必要です。デフォルトの lobHandler がデータベースシステムに適応されない場合は、lobHandler プロパティーで挿入できます。

以下は、Oracle の宣言です。

```
<bean id="lobHandler"
class="org.springframework.jdbc.support.lob.OracleLobHandler"> <property
name="nativeJdbcExtractor" ref="nativeJdbcExtractor"/> </bean> <bean
id="nativeJdbcExtractor"
class="org.springframework.jdbc.support.nativejdbc.CommonsDbcpNativeJdbcExtractor"/>
<bean id="repo"
class="org.apache.camel.processor.aggregate.jdbc.JdbcAggregationRepository">
<property name="transactionManager" ref="transactionManager"/> <property
name="repositoryName" value="aggregation"/> <property name="dataSource"
ref="dataSource"/> <!-- Only with Oracle, else use default --> <property
name="lobHandler" ref="lobHandler"/> </bean>
```

楽観的ロック

Camel 2.12 以降では、optistic Locking をオンにし、複数の Camel アプリケーションが集約リポジトリーと同じデータベースを共有するクラスター環境でこの JDBC ベースの集約リポジトリーを使用できます。競合状態がある場合、JDBC ドライバーは、JdbcAggregationRepository が反応できるベンダー固有の例外をスローします。JDBC ドライバーから発生した例外は、最適なロックエラーとして考慮されたかを知るには、マッパーが必要です。そのため、org.apache.camel.processor.aggregate.jdbc.JdbcOptimisticLockingExceptionMapper を使用すると、必要に応じてカスタムロジックを実装できます。デフォルトの実装

org.apache.camel.processor.aggregate.jdbc.DefaultJdbcOptimisticLockingExceptionMapper は以下のようになります。

以下のチェックが行われます。

発生した例外が SQLException の場合、SQLState は 23 で始まる場合にチェックされます。

発生した例外が DataIntegrityViolationExceptionの場合

原因となった例外クラス名が、名前に "ConstraintViolation" である場合。

FQN クラス名のオプションの確認は、クラス名が設定されているかどうかに一致します。

FQN クラス名を追加でき、原因となった例外（またはネストされたもの）が FQN クラス名のいずれにも等しい場合は、楽観的ロックエラーになります。

以下は、JDBC ベンダーから 2 つの追加の FQN クラス名を定義する例です。

```
<bean id="repo"
class="org.apache.camel.processor.aggregate.jdbc.JdbcAggregationRepository">
<property name="transactionManager" ref="transactionManager"/> <property
name="repositoryName" value="aggregation"/> <property name="dataSource"
ref="dataSource"/> <property name"jdbcOptimisticLockingExceptionMapper"
ref="myExceptionMapper"/> </bean> <!-- use the default mapper with extra
FQN class names from our JDBC driver --> <bean id="myExceptionMapper"
class="org.apache.camel.processor.aggregate.jdbc.DefaultJdbcOptimisticLockingExceptionMapper">
<property name="classNames"> <util:set>
<value>com.foo.sql.MyViolationExceptoion</value>
<value>com.foo.sql.MyOtherViolationExceptoion</value> </util:set>
</property> </bean>
```

304.11. CAMEL SQL STARTER

spring-boot ユーザーはスターターモジュールを利用できます。スターターを使用する場合は、spring-boot プロパティーを使用して DataSource を直接設定できます。

```
# Example for a mysql datasource
spring.datasource.url=jdbc:mysql://localhost/test
```

```
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

この機能を使用するには、以下の依存関係を Spring ブートの pom.xml ファイルに追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sql-starter</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version -->
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
  <version>${spring-boot-version}</version>
</dependency>
```

必要に応じて、特定のデータベースドライバーも追加する必要があります。

304.12. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

SQL ストアドプロシージャ

JDBC

第305章 SQL ストアドプロシージャーコンポーネント

Camel バージョン 2.17 から利用可能

sql-stored: コンポーネントを使用すると、JDBC ストアドプロシージャークエリーを使用してデータベースを操作することができます。このコンポーネントは SQL コンポーネントのエクステンションですが、ストアドプロシージャー の呼び出しに特化されています。

このコンポーネントは、実際の SQL 処理に背後で spring-jdbc を使用します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sql</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

305.1. URI 形式

SQL コンポーネントは、以下のエンドポイント URI 表記を使用します。

```
sql-stored:template[?options]
```

ここでの template はストアドプロシージャーテンプレートで、ここではストアドプロシージャーの名前、INOUT 引数、および OUT 引数を宣言します。

また、以下のようにファイルシステムまたはクラスパス上の外部ファイルでテンプレートを参照することもできます。

```
sql-stored:classpath:sql/myprocedure.sql[?options]
```

sql/myprocedure.sql は、以下のようにテンプレートを持つクラスパス内のプレーンテキストファイルです。

```
SUBNUMBERS(
```

```
  INTEGER ${headers.num1},
  INTEGER ${headers.num2},
  INOUT INTEGER ${headers.num3} out1,
  OUT INTEGER out2
)
```

URI にクエリーオプションを追加するには、?option=value&option=value&...

305.2. オプション

SQL Stored Procedure コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
dataSource (producer)	データベースとの通信に使用する DataSource を設定します。		DataSource
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SQL Stored Procedure エンドポイントは、URI 構文を使用して設定します。

```
sql-stored:template
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

305.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
template	実行する StoredProcedure テンプレートの設定が **必要**		文字列

305.2.2. クエリーパラメーター（7 パラメーター）：

Name	説明	デフォルト	Type
batch （プロデューサー）	バッチモードを有効または無効にします。	false	boolean
dataSource (producer)	データベースとの通信に使用する DataSource を設定します。		DataSource
関数 （プロデューサー）	この呼び出しが関数であるかどうか。	false	boolean
noop (producer)	設定されている場合は、テンプレートの結果を無視し、既存の IN メッセージを処理の継続用の OUT メッセージとして使用します。	false	boolean
outputHeader (producer)	テンプレートを保存すると、メッセージボディーではなくヘッダーになります。デフォルトでは、outputHeader == null とテンプレートの結果はメッセージボディーに保存され、メッセージボディー内の既存のコンテンツは破棄されます。outputHeader が設定されている場合、値はテンプレートの結果を保存するヘッダーの名前として使用され、元のメッセージボディーは保持されます。		文字列
useMessageBodyForTemplate (producer)	メッセージボディーをテンプレートとして使用し、パラメーターのヘッダーを使用するかどうか。このオプションを有効にすると、URI のテンプレートは使用されません。	false	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

305.3. ストアドプロシージャーテンプレートの宣言

テンプレートは、Java メソッド署名と似た構文を使用して宣言されます。ストアドプロシージャーの名前、次に括弧で囲まれた引数。これを説明する例を以下に示します。

```
<to uri="sql-stored:STOREDSAMPLE(INTEGER ${headers.num1},INTEGER
${headers.num2},INOUT INTEGER ${headers.num3} result1,OUT INTEGER result2)"/>
```

引数は型によって宣言され、Simple 式を使用して Camel メッセージへのマッピングが宣言されます。この例では、最初の 2 つのパラメーターは INTEGER タイプの IN 値であり、メッセージヘッダーにマッピングされます。3 番目のパラメーターは INOUT で、INTEGER を受け入れ、別の INTEGER 結果を返します。最後のパラメーターは OUT 値であり、INTEGER タイプでもあります。

SQL 用語では、ストアドプロシージャーは以下のように宣言できます。

```
CREATE PROCEDURE STOREDSAMPLE(VALUE1 INTEGER, VALUE2 INTEGER, INOUT
RESULT1 INTEGER, OUT RESULT2 INTEGER)
```

305.3.1. IN パラメーター

IN パラメーターは、パラメーター名、SQL タイプ（スケール）、型 name および value source の 4 つの部分をスペースで区切って使用します。

パラメーター名は任意で、指定されていない場合には自動生成されます。これは、引用符(')間で指定する必要があります。

SQL 型が必要であり、整数（正または負の）または一部のクラスの整数フィールドへの参照を使用できます。SQL タイプにドットが含まれる場合、コンポーネントはそのクラスを解決し、指定のフィールドを読み込もうとします。たとえば、SQL タイプの com.Foo.INTEGER は、クラス com.Foo の INTEGER フィールドから読み取られます。タイプにコンマが含まれていない場合、整数値を解決するクラスは java.sql.Types になります。タイプは、スケールスケールで必要になる場合があります。たとえば、DECIMAL(10)は、スケール 10 の java.sql.Types.DECIMAL を意味します。

タイプ名はオプションで、引用符(')の間で指定する必要があります。

値ソース が必要です。値ソース は、エクスチェンジからパラメーター値を生成します。Simple 式またはヘッダーの場所（:#<header name> など）のいずれかになります。たとえば、Simple expression ${header.val} は、パラメーター値がヘッダー「val」から読み取られることを意味します。ヘッダーの場所式 :#val は同じ効果を持ちます。

```
<to uri="sql-stored:MYFUNC('param1' org.example.Types.INTEGER(10) ${header.srcValue})"/>
```

URI は、ストアドプロシージャーがパラメーター名「param1」で呼び出されます。この SQL タイプは、クラス org.example.Types の INTEGER から読み取られ、scale は 10 に設定されます。パラメーターの入力値はヘッダー「srcValue」から渡されます。

```
<to uri="sql-stored:MYFUNC('param1' 100 'mytypename' ${header.srcValue})"/>
```

SQLtype は 100 で、タイプ name は "mytypename" 以外は、URI は以前のものと同じです。

実際の呼び出しは、org.springframework.jdbc.core.SqlParameter を使用して行われます。

305.3.2. OUT パラメーター

OUT パラメーターは同じ IN パラメーターで動作し、SQL タイプ（スケールあり）、タイプ name および output パラメーター名の 3 つの部分が含まれます。

SQL タイプは IN パラメーターと同じように動作します。

タイプ名はオプションで、IN パラメーターと同じように機能します。

出力パラメーター名は、OUT パラメーター名や、結果が保存されるヘッダー名に使用されます。

```
<to uri="sql-stored:MYFUNC(OUT org.example.Types.DECIMAL(10) outheader1)"/>
```

URI は OUT パラメーターの名前が "outheader1" であることを意味し、結果はヘッダー「outheader1"」になります。

```
<to uri="sql-stored:MYFUNC(OUT org.example.Types.NUMERIC(10) 'mytype' outheader1)"/>
```

これは以前のものと同じですが、型名は「mytype」になります。

実際の呼び出しは、org.springframework.jdbc.core.SqlOutParameter を使用して行われます。

305.3.3. INOUT パラメーター

INOUT パラメーターは上記のすべての組み合わせです。エクスチェンジから値を受信し、結果をメッセージヘッダーとして保存します。唯一の注意点は、IN パラメーターの「name」が省略されていることです。代わりに、OUT パラメーターの「name」は、SQL パラメーター名と結果ヘッダー名の両方を定義します。

```
<to uri="sql-stored:MYFUNC(INOUT DECIMAL(10) ${headers.inheader} outheader)"/>
```

Actual call will be done using org.springframework.jdbc.core.SqlInOutParameter.

305.4. CAMEL SQL STARTER

spring-boot ユーザーはスターターモジュールを利用できます。スターターを使用する場合は、spring-boot プロパティーを使用して DataSource を直接設定できます。

```
# Example for a mysql datasource
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

この機能を使用するには、以下の依存関係を Spring ブートの pom.xml ファイルに追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sql-starter</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version --
>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
  <version>${spring-boot-version}</version>
</dependency>
```

必要に応じて、特定のデータベースドライバーも追加する必要があります。

305.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに
- SQL コンポーネント

第306章 SSH コンポーネント

Camel バージョン 2.10 で利用可能

SSH コンポーネントを使用すると、SSH コマンドを送信して応答を処理できるように SSH サーバーにアクセスできます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ssh</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

306.1. URI 形式

```
ssh:[username[:password]@]host[:port][?options]
```

306.2. オプション

SSH コンポーネントは、以下に示す 12 個のオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有 SSH 設定を使用するには、以下を実行します。		SshConfiguration
ホスト （共通）	リモート SSH サーバーのホスト名を設定します。		文字列
ポート （共通）	リモート SSH サーバーのポート番号を設定します。		int
ユーザー名 （セキュリティー）	リモート SSH サーバーへのログインに使用するユーザー名を設定します。		文字列
パスワード （セキュリティー）	リモート SSH サーバーへの接続に使用するパスワードを設定します。keyPairProvider を null に設定する必要があります。		文字列

Name	説明	デフォルト	Type
pollCommand (common)	すべてのポーリングサイクル中にリモート SSH サーバーに送信するコマンド文字列を設定します。コンシューマーとして使用される camel-ssh コンポーネントとのみ機能します（つまり from(ssh://...)。コマンドを改行して終了する必要がある場合があります。URL でエンコードされる %0A を指定する必要があります。		文字列
keyPairProvider (security)	証明書を使用してリモート SSH サーバーに接続する際に使用する KeyPairProvider 参照を設定します。		KeyPairProvider
keyType (security)	認証の一部として KeyPairProvider に渡すキータイプを設定します。KeyPairProvider.loadKey(...)はこの値を渡します。デフォルトは ssh-rsa です。		文字列
タイムアウト (common)	リモート SSH サーバー接続の確立を待つタイムアウトをミリ秒単位で設定します。デフォルトは 30000 ミリ秒です。		Long
certFilename (security)	**非推奨**: 認証に使用する証明書のリソースパスを設定します。		文字列
certResource (security)	認証に使用する証明書のリソースパスを設定します。ResourceHelperKeyPairProvider を使用してファイルベースの証明書を解決し、keyType 設定により異なります。		文字列
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

SSH エンドポイントは URI 構文を使用します。

```
ssh:host:port
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

306.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
host	**必須**。リモート SSH サーバーのホスト名を設定します。		文字列
port	リモート SSH サーバーのポート番号を設定します。	22	int

306.2.2. クエリーパラメーター（28 パラメーター）：

Name	説明	デフォルト	Type
failOnUnknownHost (common)	不明なホストへの接続に失敗するかどうかを指定します。この値は、knownHosts プロパティーが設定されている場合にのみチェックされます。	false	boolean
knownHostsResource (common)	known_hosts ファイルのリソースパスを設定します。		文字列
タイムアウト (common)	リモート SSH サーバー接続の確立を待つタイムアウトをミリ秒単位で設定します。デフォルトは 30000 ミリ秒です。	30000	Long
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
pollCommand (consumer)	すべてのポーリングサイクル中にリモート SSH サーバーに送信するコマンド文字列を設定します。コンシューマーとして使用される camel-ssh コンポーネントとのみ機能します（例： from(ssh://...)。コマンドラインでコマンドを終了する必要があり、URL エンコード %0A にする必要があります。		文字列
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean

Name	説明	デフォルト	Type
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean
certResource (security)	認証に使用する証明書のリソースパスを設定します。ResourceHelperKeyPairProvider を使用してファイルベースの証明書を解決し、keyType 設定により異なります。		文字列
keyPairProvider (security)	証明書を使用してリモート SSH サーバーに接続する際に使用する KeyPairProvider 参照を設定します。		KeyPairProvider
keyType (security)	認証の一部として KeyPairProvider に渡すキータイプを設定します。KeyPairProvider.loadKey(...)はこの値を渡します。デフォルトは ssh-rsa です。	ssh-rsa	文字列
パスワード（セキュリティー）	リモート SSH サーバーへの接続に使用するパスワードを設定します。keyPairProvider を null に設定する必要があります。		文字列

Name	説明	デフォルト	Type
ユーザー名 （セキュリティー）	リモート SSH サーバーへのログインに使用するユーザー名を設定します。		文字列

306.3. プロデューサーエンドポイントとしての使用

SSH コンポーネントがプロデューサー（.to("ssh://...")として使用される場合、メッセージボディーはリモート SSH サーバーで実行するコマンドとして送信されます。

*以下は、XML DSL 内での例です。コマンドには XML でエンコードされた改行(
)があります。*

```
<route id="camel-example-ssh-producer">
  <from uri="direct:exampleSshProducer"/>
  <setBody>
    <constant>features:list&amp;#10;</constant>
  </setBody>
  <to uri="ssh://karaf:karaf@localhost:8101"/>
  <log message="${body}"/>
</route>
```

306.4. AUTHENTICATION

SSH コンポーネントは、公開鍵証明書またはユーザー名/パスワードのいずれかを使用して、リモートの SSH サーバーに対して認証できます。SSH コンポーネントの認証方法は、オプションの設定方法およびオプションを基にしています。

1. *まず、certResource オプションが設定されているかどうかを確認し、ある場合はこれを使用して参照される公開鍵証明書を特定し、その証明書を認証に使用するようにします。*

2. *certResource が設定されていない場合、keyPairProvider が設定されているかどうかを確認し、その場合は証明書ベースの認証に使用します。*

3. *certResource および keyPairProvider が設定されていないと、認証に ユーザー名 および パスワード オプションが使用されます。ユーザー名 と パスワード はエンドポイント設定に提供され、SshConstants.USERNAME_HEADER (CamelSshUsername)および SshConstants.PASSWORD_HEADER (CamelSshPassword)で設定されたヘッダーで提供されますが、ヘッダーに設定されたエンドポイント設定が使用されます。*

以下のルートフラグメントは、クラスパスの証明書を使用した SSH ポーリングコンシューマーを示しています。

XML DSL の場合：

```
<route>
  <from uri="ssh://scott@localhost:8101?
certResource=classpath:test_rsa&amp;useFixedDelay=true&amp;delay=5000&amp;pollCommand=features:list%0A"/>
  <log message="${body}"/>
</route>
```

Java DSL の場合

```
from("ssh://scott@localhost:8101?
certResource=classpath:test_rsa&useFixedDelay=true&delay=5000&pollCommand=features:list%0A")
    .log("${body}");
```

公開鍵認証の使用例は、examples/camel-example-ssh-security にあります。

証明書の依存関係

証明書ベースの認証を使用する場合は、追加のランタイム依存関係を追加する必要があります。表示される依存関係バージョンは Camel 2.11 の時点で、使用している Camel のバージョンによって、新しいバージョンを使用する必要がある場合があります。

```
<dependency>
  <groupId>org.apache.sshd</groupId>
  <artifactId>sshd-core</artifactId>
  <version>0.8.0</version>
</dependency>
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcpg-jdk15on</artifactId>
  <version>1.47</version>
</dependency>
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcpkix-jdk15on</artifactId>
  <version>1.47</version>
</dependency>
```

306.5. 例

Camel ディストリビューションの examples/camel-example-ssh および examples/camel-example-ssh-security を参照してください。

306.6. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第307章 STAX コンポーネント

Camel バージョン 2.9 で利用可能

StAX コンポーネントを使用すると、SAX ContentHandler を介してメッセージを処理できます。このコンポーネントのもう 1 つの機能は、Splitter EIP の使用などを使用して、StAX を使用して JAXB レコードを繰り返し処理できるようにすることです。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-stax</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

307.1. URI 形式

```
stax:content-handler-class
```

例：

```
stax:org.superbiz.FooContentHandler
```

Camel 2.11.1 以降では、以下のように # 構文を使用して、レジストリーから org.xml.sax.ContentHandler Bean を検索できます。

```
stax:#myHandler
```

307.2. オプション

StAX コンポーネントにはオプションがありません。

StAX エンドポイントは URI 構文を使用します。

```
stax:contentHandlerClass
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

307.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
contentHandlerClass	使用する ContentHandler 実装の FQN クラス名が **必要です**。		文字列

307.2.2. クエリーパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

307.3. STAX パーサーとしてのコンテンツハンドラーの使用

処理後のメッセージボディーはハンドラー自体になります。

以下に例を示します。

```
from("file:target/in")
  .to("stax:org.superbiz.handler.CountingHandler")
  // CountingHandler implements org.xml.sax.ContentHandler or extends
org.xml.sax.helpers.DefaultHandler
  .process(new Processor() {
    @Override
    public void process(Exchange exchange) throws Exception {
        CountingHandler handler = exchange.getIn().getBody(CountingHandler.class);
        // do some great work with the handler
    }
  });
```

307.4. JAXB および STAX を使用してコレクションを反復処理します。

まず、JAXB オブジェクトがあるとします。

たとえば、ラッパーオブジェクトのレコードのリストの場合：

```
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "records")
public class Records {
    @XmlElement(required = true)
    protected List<Record> record;

    public List<Record> getRecord() {
        if (record == null) {
            record = new ArrayList<Record>();
        }
        return record;
    }
}
```

および

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "record", propOrder = { "key", "value" })
public class Record {
    @XmlAttribute(required = true)
    protected String key;

    @XmlAttribute(required = true)
    protected String value;

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
```

```
        this.value = value;
    }
}
```

次に、処理する XML ファイルを取得します。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<records>
  <record value="v0" key="0"/>
  <record value="v1" key="1"/>
  <record value="v2" key="2"/>
  <record value="v3" key="3"/>
  <record value="v4" key="4"/>
  <record value="v5" key="5"/>
</record>
```

StAX コンポーネントは StAXBuilder を提供します。これは、XML 要素を Camel Splitter で反復するときに使用できます。

```
from("file:target/in")
    .split(stax(Record.class)).streaming()
        .to("mock:records");
```

stax は、Java コードで静的インポートできる org.apache.camel.component.stax.StAXBuilder の静的メソッドです。stax ビルダーはデフォルトで、使用する XMLReader で認識されます。Camel 2.11.1 以降では、以下のように boolean パラメーターを false に設定すると、これをオフにすることができます。

```
from("file:target/in")
    .split(stax(Record.class, false)).streaming()
        .to("mock:records");
```

307.4.1. XML DSL を使用した以前の例

上記の例は、XML DSL で以下のように実装できます。

307.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- *エンドポイント*
- *はじめに*

第308章 STOMP コンポーネント

Camel バージョン 2.12 から利用可能

stomp: コンポーネントは、Apache ActiveMQ や ActiveMQ Apolloなどの Stomp 準拠のメッセージブローカーとの通信に使用されます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-stomp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

308.1. URI 形式

```
stomp:queue:destination[?options]
```

destination はキューの名前です。

308.2. オプション

Stomp コンポーネントは、以下に示す 8 個のオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有された stomp 設定を使用するには、以下を実行します。		StompConfiguration
brokerURL (common)	接続する Stomp ブローカーの URI。		文字列
Login （セキュリティー）	ユーザー名		文字列
passcode （セキュリティー）	パスワード		文字列

Name	説明	デフォルト	Type
ホスト（共通）	仮想ホスト		文字列
useGlobalSslContext Parameters (security)	グローバル SSL コンテキストパラメーターの使用を有効にします。	false	boolean
headerFilterStrategy (filter)	カスタムの org.apache.camel.spi.HeaderFilterStrategy を使用して、Camel メッセージとの間でヘッダーをフィルターします。		HeaderFilterStrategy
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Stomp エンドポイントは URI 構文を使用して設定します。

```
stomp:destination
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

308.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
destination	キューの **必要な** 名前		文字列

308.2.2. クエリーパラメーター（10 パラメーター）：

Name	説明	デフォルト	Type
brokerURL (common)	接続する Stomp ブローカーの URI が **必要です**。	tcp://localhost:61613	文字列
ホスト（共通）	仮想ホスト名		文字列

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
headerFilterStrategy (advanced)	カスタム HeaderFilterStrategy を使用して Camel メッセージに対してヘッダーをフィルターします。		HeaderFilterStrategy
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
Login （セキュリティー）	ユーザー名		文字列
passcode （セキュリティー）	パスワード		文字列
sslContextParameters (security)	SSLContextParameters を使用したセキュリティーの設定		SSLContextParameters

URI にクエリーオプションを追加するには、 ?option=value&option=value&...

308.3. サンプル

メッセージの送信：

```
from("direct:foo").to("stomp:queue:test");
```

メッセージの消費：

```
from("stomp:queue:test").transform(body().convertToString()).to("mock:result")
```

308.4. エンドポイント

Camel は Endpoint インターフェースを使用した Message Endpoint パターンをサポートします。通常、エンドポイントは Component および Endpoints によって作成されます。これは通常、URI 経由で DSL で参照されます。

エンドポイントから以下のメソッドを使用できます。

* createProducer() は、メッセージエクスチェンジをエンドポイントに送信するプロデューサーを作成します。* createConsumer() は、Consumer * create PollingConsumer() は、PollingConsumer 経由でエンドポイントからメッセージエクスチェンジを消費するためのイベント駆動コンシューマーパターンを実装します。http://camel.apache.org/maven/current/camel-core/apidocs/org/apache/camel/Processor.html

308.5. 関連項目

- Configuring Camel (Camel の設定)
- Message Endpoint パターン
- URI
- コンポーネントの作成

第309章 ストリームコンポーネント

Camel バージョン 1.3 で利用可能

stream: コンポーネントは、System.in、System.out、および System.err ストリームへのアクセスを提供し、ファイルおよび URL のストリーミングを可能にします。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-stream</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

309.1. URI 形式

```
stream:in[?options]
stream:out[?options]
stream:err[?options]
stream:header[?options]
```

さらに、File および url エンドポイント URI もサポートされます。

```
stream:file?fileName=/foo/bar.txt
stream:url[?options]
```

stream:header URI が指定されている場合は、ストリーム ヘッダーを使用して、書き込むストリームを検索します。このオプションは、ストリームプロデューサーにのみ利用できます（つまり、from（）には表示されません）。

URI にクエリーオプションを追加するには、?option=value&option=value&…

309.2. オプション

Stream コンポーネントにはオプションがありません。

Stream エンドポイントは URI 構文を使用します。

```
stream:kind
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

309.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
kind	**System**.in または System.out など、使用するストリームの Kind。		文字列

309.2.2. クエリーパラメーター（18 パラメーター）：

Name	説明	デフォルト	Type
encoding (common)	テキストベースのストリームを使用するようにエンコーディング（文字セット名）を設定できます（メッセージボディーは String オブジェクトです）。指定されていない場合、Camel は JVM デフォルトの Charset を使用します。		文字列
fileName (common)	stream:file URI 形式を使用する場合、このオプションはストリーム先となる/元のファイル名を指定します。		文字列
URL (common)	stream:url URI 形式を使用する場合、このオプションはストリーム先となる/元の URL を指定します。入出力ストリームは、JDK URLConnection 機能を使用して開きます。		文字列
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
groupLines (consumer)	コンシューマーの X 行数をグループ化します。たとえば、10 行をグループ化し、行ごとに 1 つの Exchange ではなく、10 行だけを区切るには、エクスチェンジを 1 行ずつ実行します。		int
groupStrategy (consumer)	カスタム GroupStrategy を使用して行をグループ化する方法を制御できます。		GroupStrategy
initialPromptDelay (consumer)	メッセージプロンプトを表示するまでの初期の遅延（ミリ秒単位）。この遅延は 1 回のみ発生します。システムの起動時に使用して、システムに他のロギングが行われている間にメッセージプロンプトが書き込まれないようにすることができます。	2000	Long
promptDelay (consumer)	メッセージプロンプトを表示する前の任意の遅延（ミリ秒単位）。		Long
promptMessage (consumer)	stream:in からの読み取り時に使用するメッセージプロンプトです。たとえば、これを Enter a コマンドに設定できます。		文字列
Retry (consumer)	上書きされた場合はファイルを開く再試行します（tail --retry など）。	false	boolean
scanStream (consumer)	unix tail コマンドなどのストリームを継続的に読み取るために使用します。	false	boolean
scanStreamDelay (consumer)	scanStream の使用時の読み取り試行の間隔（ミリ秒単位）。		Long
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
autoCloseCount (producer)	プロデューサー側でストリームを閉じる前に処理するメッセージの数。デフォルトではストリームを閉じないでください（プロデューサーが停止した場合のみ）。追加のメッセージが送信されると、ストリームが別の autoCloseCount バッチに対して再度開きます。		int

Name	説明	デフォルト	Type
closeOnDone (producer)	このオプションは Splitter と併用され、同じファイルにストリーミングされます。パフォーマンスを向上させるために、ストリームを開いた状態にし、Splitter が実行された場合にのみ閉じられることです。これは、2 つ以上のファイルではなく、同じファイルにストリーミングする必要がある点に注意してください。	false	boolean
遅延 （プロデューサー）	ストリームを生成する前の初期遅延（ミリ秒単位）。		Long
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

309.3. メッセージの内容

stream: コンポーネントはストリームに書き込むために String または byte[] のいずれかをサポートします。String または byte[] コンテンツを message.in.body に追加します。バイナリーモードで stream: プロデューサーに送信されたメッセージでは、（String メッセージではなく）改行文字の後には改行文字が続きません。null ボディーを持つメッセージは出力ストリームに追加されません。カスタム出力ストリームには、特別な stream:header URI が使用されます。java.io.OutputStream オブジェクトをキーヘッダーの message.in. header に追加します。
サンプルについては、「サンプル」を参照してください。

309.4. サンプル

以下の例では、direct:in エンドポイントから System.out ストリームにメッセージをルーティングします。

```
// Route messages to the standard output.
from("direct:in").to("stream:out");

// Send String payload to the standard output.
// Message will be followed by the newline.
template.sendBody("direct:in", "Hello Text World");

// Send byte[] payload to the standard output.
// No newline will be added after the message.
template.sendBody("direct:in", "Hello Bytes World".getBytes());
```

以下の例は、ヘッダータイプを使用して、使用するストリームを判別する方法を示しています。この例では、独自の出力ストリーム MyOutputStream を使用します。

以下の例は、ファイルストリームを継続的に読み取る方法を示しています（UNIX tail コマンドに似ています）。

```
from("stream:file?
fileName=/server/logs/server.log&scanStream=true&scanStreamDelay=1000").to("bean:logSer
vice?method=parseLogLine");
```

scanStream（Camel 2.7 以前）または scanStream + retry が含まれる 1 つの gotcha は、scanStreamDelay の各反復でファイルを再度開き、スキャンします。NIO2 が利用可能になるまで、ファイルの削除/再作成時に確実に検出できません。

309.5. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント
- エンドポイント
- はじめに

第310章 STRING ENCODING DATAFORMAT

Camel バージョン 2.12 から利用可能

String Data Format はエンコーディングをサポートするテキストベースの形式です。

310.1. オプション

String Encoding データ形式は、以下に示す 2 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
charset		文字列	使用するエンコーディングを設定します。デフォルトでは、JVM プラットフォームのデフォルト文字セットを使用します。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

310.2. マーシャリング

この例では、ファイルの内容を UTF-8 エンコーディングで String オブジェクトにマーシャルします。

```
from("file://data.csv").marshal().string("UTF-8").to("jms://myqueue");
```

310.3. アンマーシャリング

この例では、newOrder プロセッサーで処理される前に、UTF-8 エンコーディングを使用して JMS キューから String オブジェクトにペイロードをアンマーシャリングします。

```
from("jms://queue/order").unmarshal().string("UTF-8").processRef("newOrder");
```

310.4. 依存関係

このデータ形式は camel-core で提供されるため、追加の依存関係は必要ありません。

第311章 文字列テンプレートコンポーネント

Camel バージョン 1.2 で利用可能

string-template: コンポーネントを使用すると、String Template を使用してメッセージを処理できます。これは、Templating を使用してリクエストの応答を生成する場合に便利です。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-stringtemplate</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

311.1. URI 形式

```
string-template:templateName[?options]
```

templateName は、呼び出すテンプレートのクラスパスローカル URI またはリモートテンプレートの完全な URL です。

URI にクエリーオプションを追加するには、?option=value&option=value&…

311.2. オプション

String Template コンポーネントにはオプションがありません。

String Template エンドポイントは、URI 構文を使用して設定されます。

```
string-template:resourceUri
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

311.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
resourceUri	リソースへの **必須** パス。プレフィックス：classpath、file、http、ref、または bean. classpath、file、http は、これらのプロトコルを使用してリソースをロードします（classpath はデフォルト）。ref はレジストリーのリソースを検索します。Bean はリソースとして使用される Bean でメソッドを呼び出します。Bean には、ドットの後にメソッド名を指定できます（例：bean:myBean.myMethod）。		文字列

311.2.2. クエリーパラメーター（4 パラメーター）：

Name	説明	デフォルト	Type
contentCache (producer)	リソースコンテンツキャッシュを使用するかどうかを設定します。	false	boolean
delimiterStart (producer)	変数の開始区切り文字	<	char
delimiterStop (producer)	変数終了区切り文字	>	char
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

311.3. HEADERS

Camel は、キー org.apache.camel.stringtemplate.resource を持つメッセージヘッダーにリソースへの参照を保存します。Resource は org.springframework.core.io.Resource オブジェクトです。

311.4. ホットリロード

文字列テンプレートリソースは、デフォルトでファイルとクラスパスリソース（展開 jar）に対してホットリロードが可能です。contentCache=true を設定すると、Camel はリソースを 1 回だけロードし、ホットリロードはできません。このシナリオは、リソースが変更されなかった場合に実稼働で使用できます。

311.5. STRINGTEMPLATE 属性

Camel 2.14 以降、以下のコードのようにメッセージヘッダー「CamelStringTemplateVariableMap」を設定して、カスタムコンテキストマップを定義できます。

```
Map<String, Object> variableMap = new HashMap<String, Object>();
Map<String, Object> headersMap = new HashMap<String, Object>();
headersMap.put("name", "Willem");
variableMap.put("headers", headersMap);
variableMap.put("body", "Monday");
variableMap.put("exchange", exchange);
exchange.getIn().setHeader("CamelStringTemplateVariableMap", variableMap);
```

311.6. サンプル

たとえば、メッセージへの応答を形成するために、以下のように文字列テンプレートを使用できます。

```
from("activemq:My.Queue").
  to("string-template:com/acme/MyResponse.tm");
```

311.7. メールサンプル

この例では、文字列テンプレートを使用して注文確認メールを送信します。メールテンプレートは StringTemplate でレイアウトされます。この例では camel 2.11.0 で動作します。camel のバージョンが 2.11.0 未満の場合は、変数を開始して $ で終了する必要があります。

```
Dear <headers.lastName>, <headers.firstName>

Thanks for the order of <headers.item>.

Regards Camel Riders Bookstore
<body>
```

java コードは以下のようになります。

311.8. 関連項目

- Configuring Camel (Camel の設定)
- コンポーネント

- *エンドポイント*
- *はじめに*

第312章 STUB コンポーネント

Camel バージョン 2.10 で利用可能

stub: component は、開発またはテスト中に物理エンドポイントをスタブアウトする簡単な方法を提供します。これにより、特定の SMTP または Http エンドポイントに実際に接続せずにルートを実行できます。エンドポイントをスタブアウトするには、エンドポイント URI の前に stub: を追加します。

Stub コンポーネント内で、VM エンドポイントを作成します。Stub と VM の主な相違点は、仮想 マシン が URI と指定したパラメーターを検証することです。そのため、クエリー引数を持つ通常の URI の前に vm: を置くと通常失敗します。ただし、スタブは基本的にすべてのクエリーパラメーターを無視し、ルート内の 1 つ以上のエンドポイントを一時的にスタブアウトできるようにします。

312.1. URI 形式

```
stub:someUri
```

someUri は、クエリーパラメーターを持つ URI にすることができます。

312.2. オプション

Stub コンポーネントは、以下に示す 4 つのオプションをサポートします。

Name	説明	デフォルト	Type
queueSize (advanced)	SEDA キューのデフォルト最大容量（つまり、保持できるメッセージの数）を設定します。		int
concurrentConsumers (consumer)	エクスチェンジを処理するデフォルトの同時スレッド数を設定します。	1	int
defaultQueueFactory (advanced)	デフォルトのキューファクトリーを設定します。		Exchange>
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Stub エンドポイントは URI 構文を使用して設定します。

```
stub:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

312.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	**必要な** キューの名前		文字列

312.2.2. クエリーパラメーター（16 パラメーター）：

Name	説明	デフォルト	Type
size (common)	SEDA キューの最大容量（つまり、保持できるメッセージの数）。	2147483647	int
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
concurrentConsumers (consumer)	エクスチェンジを処理する同時スレッドの数。	1	int
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern

Name	説明	デフォルト	Type
limitConcurrentConsumers (consumer)	concurrentConsumer の数を最大 500 に制限するかどうか。デフォルトでは、エンドポイントが数値が大きい場合に例外が発生します。このオプションをオフにして、チェックを無効にすることができます。	true	boolean
multipleConsumers (consumer)	複数のコンシューマーを許可するかどうかを指定します。有効にすると、Publish-Subscribe messaging に SEDA を使用できます。つまり、SEDA キューにメッセージを送信し、各コンシューマーがメッセージのコピーを受け取ることができます。このオプションを有効にすると、すべてのコンシューマーエンドポイントでこのオプションを指定する必要があります。	false	boolean
pollTimeout (consumer)	ポーリング時に使用されるタイムアウト。タイムアウトが発生すると、コンシューマーは実行を継続できるかどうかを確認できます。値を低く設定すると、シャットダウン時にコンシューマーがより迅速に反応できるようになります。	1000	int
purgeWhenStopping (consumer)	コンシューマー/ルートを停止する際にタスクキューをパージするかどうか。これにより、キューで保留中のメッセージが破棄されるため、より早く停止することができます。	false	boolean
blockWhenFull (producer)	メッセージを完全な SEDA キューに送信するスレッドが、キューの容量が使い切られるまでブロックするかどうか。デフォルトでは、キューが満杯であることを示す例外がスローされます。このオプションを有効にすると、呼び出しスレッドがブロックされ、メッセージが受け入れられるまで待機します。	false	boolean
discardIfNoConsumers (producer)	アクティブなコンシューマーのないキューに送信するときに、プロデューサーがメッセージを破棄するかどうか（メッセージをキューに追加しないでください）。同時に有効にできるオプションは discardIfNoConsumers および failIfNoConsumers の 1 つのみです。	false	boolean
failIfNoConsumers (producer)	アクティブなコンシューマーのないキューに送信するときに、プロデューサーが例外を発生させて失敗するかどうか。同時に有効にできるオプションは discardIfNoConsumers および failIfNoConsumers の 1 つのみです。	false	boolean

Name	説明	デフォルト	Type
タイムアウト（プロデューサー）	SEDA プロデューサーが非同期タスクの完了の待機を停止するまでのタイムアウト（ミリ秒単位）。タイムアウトを無効にするには、0 または負の値を使用します。	30000	Long
waitForTaskToComplete (producer)	非同期タスクが完了するまで待機すべきかどうかを指定するオプション。続行します。Always、Never、または IfReplyExpected の 3 つのオプションがサポートされます。最初の 2 つの値は self-explany です。IfReplyExpected の最後の値は、メッセージが Request Reply based の場合にのみ待機します。デフォルトオプションは IfReplyExpected です。	IfReplyExpected	WaitForTaskToComplete
queue（詳細）	エンドポイントによって使用されるキューインスタンスを定義します。このオプションは、カスタムキューインスタンスを使用するまれなユースケースのみとなります。		BlockingQueue
同期（詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

312.3. 例

スタブエンドポイント URI のサンプルをいくつか示します。

```
stub:smtp://somehost.foo.com?user=whatnot&something=else
stub:http://somehost.bar.com/something
```

第313章 SWAGGER JAVA コンポーネント

Camel 2.16 から利用可能

Rest DSL は、Swagger を使用して REST サービスおよびその API を公開するために使用される camel-swagger-java モジュールと統合できます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

Camel 2.16 以降、swagger コンポーネントは純粋に Java をベースとし、

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-swagger-java</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

camel-swagger-java モジュールは REST コンポーネントから使用できます（サーブレットは必要ありません）。

例は、Apache Camel ディストリビューションの examples ディレクトリーの camel-example-swagger-cdi を参照してください。

313.1. REST-DSL での SWAGGER の使用

以下のように apiContextPath dsl を設定して rest-dsl から swagger api を有効にできます。

```
public class UserRouteBuilder extends RouteBuilder {
  @Override
  public void configure() throws Exception {
    // configure we want to use servlet as the component for the rest DSL
    // and we enable json binding mode
    restConfiguration().component("netty4-http").bindingMode(RestBindingMode.json)
      // and output using pretty print
      .dataFormatProperty("prettyPrint", "true")
      // setup context path and port number that netty will use
      .contextPath("/").port(8080)
      // add swagger api-doc out of the box
      .apiContextPath("/api-doc")
```

```
            .apiProperty("api.title", "User API").apiProperty("api.version", "1.2.3")
            // and enable CORS
            .apiProperty("cors", "true");

        // this user REST service is json only
        rest("/user").description("User rest service")
            .consumes("application/json").produces("application/json")
            .get("/{id}").description("Find user by id").outType(User.class)
                .param().name("id").type(path).description("The id of the user to
get").dataType("int").endParam()
                .to("bean:userService?method=getUser(${header.id})")
            .put().description("Updates or create a user").type(User.class)
                .param().name("body").type(body).description("The user to update or
create").endParam()
                .to("bean:userService?method=updateUser")
            .get("/findAll").description("Find all users").outTypeList(User.class)
                .to("bean:userService?method=listUsers");
    }
}
```

313.2. オプション

swagger モジュールは、以下のオプションを使用して設定できます。サーブレットを使用して設定するには、上記のように init-param を使用します。rest-dsl で直接設定する場合は、enableCORS、host、contextPath、dsl などの適切なメソッドを 使用します。api.xxx のオプションは、apiProperty dsl を使用して設定されます。

オプション	型	説明
cors	ブール値	CORS を有効にするかどうか。これにより、api ブラウザーの CORS のみが有効になり、REST サービスへの実際のアクセスは有効になりません。デフォルトは false です。CorsFilte の使用が推奨されます。詳細は以下を参照してください。
swagger.version	文字列	Swagger 仕様のバージョン。デフォルトは 2.0 です。
host	文字列	ホスト名を設定します。camel-swagger-java を設定しないと、名前を localhost ベースとして計算されます。
スキーマ	文字列	使用するプロトコルスキーム。複数の値は、"http,https" のようにコンマで区切ることができます。デフォルト値は「http」です。このオプションは、名前が schemes であるため、Camel 2.17 以降では **非推奨** となっています。

オプション	型	説明
schemes	文字列	**Camel 2.17:** 使用するプロトコルスキーム。複数の値は、"http,https" のようにコンマで区切ることができます。デフォルト値は「http」です。
base.path	文字列	**必須**: REST サービスが利用できるベースパスを設定します。パスは相対（http/https で開始しない）で、camel-swagger-java はランタイム時に絶対パスを算出します。これは **protocol://host:port/context-path/base.path になります。**
api.path	文字列	API が利用できるパスを設定します（例： /api-docs）。パスは相対（http/https で開始しない）で、camel-swagger-java はランタイム時に絶対ベースパスを算出します。これは **protocol://host:port/context-path/api.path です。** そのため、相対パスの使用ははるかに簡単です。例は、上記を参照してください。
api.version	文字列	API のバージョン。デフォルトは 0.0.0 です。
api.title	文字列	アプリケーションのタイトル。
api.description	文字列	アプリケーションの簡単な説明。
api.termsOfService	文字列	API 利用規約への URL。
api.contact.name	文字列	連絡先する個人または組織の名前
api.contact.email	文字列	API 関連の連絡先に使用するメール。
api.contact.url	文字列	詳細については、Web サイトへの URL。
api.license.name	文字列	API に使用されるライセンス名。
api.license.url	文字列	API に使用されるライセンスの URL。
apiContextIdListing	boolean	REST サービスを持つ JVM 内のすべての CamelContext 名を一覧表示できるようにするかどうか。有効にすると、api-doc のルートパスはすべてのコンテキストを一覧表示します。無効にするとコンテキスト ID が一覧表示されず、api-doc のルートパスは現在の CamelContext を一覧表示します。デフォルトは false です。

オプション	型	説明
apiContextIdPattern	文字列	コンテキストリストに表示される CamelContext 名のフィルターを可能にするパターン。このパターンは正規表現と * をワイルドカードとして使用します。Intercept で使用されるのと同じパターン一致

313.3. CONTEXTIDLISTING ENABLED

contextIdListing を有効にすると、同じ JVM で実行中の CamelContext をすべて検出します。これらのコンテキストはルートパスにリストされます（例： json 形式の名前の単純な一覧として /api-docs ）。swagger ドキュメントにアクセスするには、api-docs/myCamel などの Camel コンテキスト ID で context-path を追加する必要があります。apiContextIdPattern オプションを使用して、この一覧の名前をフィルタリングすることができます。

313.4. JSON または YAML

Camel 2.17 から利用可能

camel-swagger-java モジュールは、追加設定なしの JSon と Yaml の両方をサポートします。いずれかのエンドポイントに /swagger.json または /swagger.yaml を使用して返される要求 URL を指定できます。指定がない場合は、HTTP Accept ヘッダーを使用して json または yaml を受け入れることができるかどうかを検出します。両方が受け入れられるか、または none が受け入れられないと、json がデフォルトの形式として返されます。

313.5. 例

Apache Camel ディストリビューションでは、この Swagger コンポーネントを使用する camel-example-swagger-cdi および camel-example-swagger-java が同梱されています。

第314章 SYSLOG DATAFORMAT

Camel バージョン 2.6 で利用可能

syslog データ形式は、RFC3164 および RFC5424 メッセージの使用に使用されます。

このコンポーネントは以下をサポートします。

- syslog メッセージの UDP 消費
- プレーンな String オブジェクトまたは SyslogMessage モデルオブジェクトを使用した非依存データフォーマット。
- Type Converter from/to SyslogMessage and String
- camel-mina コンポーネントとの統合。
- camel-netty コンポーネントとの統合。
- Camel 2.14: camel-netty コンポーネントの Encoder および decoder。
- Camel 2.14: RFC5424 のサポート

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-syslog</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

314.1. RFC3164 SYSLOG プロトコル

syslog は、UDP（ユーザーデータグラムプロトコル）[1] を基盤のトランスポート層メカニズムとして使用します。syslog に割り当てられた UDP ポートは 514 です。

Syslog リスナーサービスを公開するには、既存の camel-mina コンポーネントまたは camel-netty を再利用します。この場合、Rfc3164SyslogDataFormat を使用してメッセージをマーシャリングおよびアンマーシャリングします。Camel 2.14 以降では、syslog データフォーマットの名前が SyslogDataFormat に変更されていることに注意してください。

314.2. オプション

Syslog データフォーマットは、以下に示す 1 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

314.3. RFC5424 SYSLOG プロトコル

Camel 2.14 から利用可能

Syslog リスナーサービスを公開するには、 SyslogDataFormat を使用してメッセージをマーシャリングおよびアンマーシャリングする既存の camel-mina コンポーネントまたは camel-netty を再利用します。

314.3.1. Syslog リスナーの公開

Spring XML ファイルでは、ポート 10514 で udp メッセージをリッスンするようにエンドポイントを設定します。netty では defaultCodec を無効にし、この は NettyTypeConverter へのフォールバックを許可し、メッセージを InputStream として提供します。

```
<camelContext id="myCamel" xmlns="http://camel.apache.org/schema/spring">

  <dataFormats>
    <syslog id="mySyslog"/>
```

```
    </dataFormats>

    <route>
        <from uri="netty:udp://localhost:10514?sync=false&amp;allowDefaultCodec=false"/>
        <unmarshal ref="mySyslog"/>
        <to uri="mock:stop1"/>
    </route>

</camelContext>
```

camel-minaを使用した同じルート

```
<camelContext id="myCamel" xmlns="http://camel.apache.org/schema/spring">

    <dataFormats>
        <syslog id="mySyslog"/>
    </dataFormats>

    <route>
        <from uri="mina:udp://localhost:10514"/>
        <unmarshal ref="mySyslog"/>
        <to uri="mock:stop1"/>
    </route>

</camelContext>
```

314.3.2. syslog メッセージのリモート宛先への送信

```
<camelContext id="myCamel" xmlns="http://camel.apache.org/schema/spring">

    <dataFormats>
        <syslog id="mySyslog"/>
    </dataFormats>

    <route>
        <from uri="direct:syslogMessages"/>
        <marshal ref="mySyslog"/>
        <to uri="mina:udp://remotehost:10514"/>
    </route>

</camelContext>
```

314.4. 関連項目

- *Configuring Camel (Camel の設定)*
- *コンポーネント*

- ***エンドポイント***

- ***はじめに***

第315章 TAR FILE DATAFORMAT

Camel バージョン 2.16 から利用可能

Tar File Data Format は、メッセージ圧縮と圧縮解除形式です。メッセージは、1 つのエントリーを含む Tar ファイルにマーシャル（圧縮）することができ、単一のエントリーを含む Tar ファイルは、元のファイルのコンテンツにアンマーシャリング（展開）できます。

また、複数のメッセージを単一の Tar ファイルに統合できる集約ストラテジーもあります。

315.1. TAR ファイルオプション

Tar File データフォーマットは、以下に示す 4 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
usingIterator	**false**	**ブール値**	tar ファイルに複数のエントリーがある場合、このオプションを true に設定すると、Splitter EIP との作業が可能になり、ストリーミングモードでイテレーターを使用してデータを分割できます。
allowEmptyDirectory	**false**	**ブール値**	tar ファイルに複数のエントリーがある場合は、このオプションを true に設定すると、ディレクトリーが空であってもイテレーターを取得できます。
preservePathElements	**false**	**ブール値**	ファイル名にパス要素が含まれる場合は、このオプションを true に設定すると、パスを tar ファイルで維持できます。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

315.2. マーシャリング

この例では、Tar File 圧縮を使用して通常のテキスト/XML ペイロードを圧縮ペイロードにマーシャリングし、MY_QUEUE という ActiveMQ キューに送信します。

```
from("direct:start").marshal().tarFile().to("activemq:queue:MY_QUEUE");
```

作成された Tar File 内の Tar エントリーの名前は、ファイルコンポーネントによって使用される標準メッセージヘッダーである、受信 CamelFileName メッセージヘッダーに基づいています。さらに、発信 CamelFileName メッセージヘッダーは、".tar" 接尾辞が付いた受信 CamelFileName メッセージヘッダーの値に自動的に設定されます。たとえば、次のルートが入力ディレクトリーで「test.txt」という名前のファイルを見つけると、出力は "test.txt" という名前の単一の Tar エントリーが含まれる "test.txt.tar" という名前の Tar ファイルになります。

```
from("file:input/directory?antInclude=*/.txt").marshal().tarFile().to("file:output/directory");
```

受信 CamelFileName メッセージヘッダー（ファイルコンポーネントがコンシューマーでない場合など）がない場合、メッセージ ID はデフォルトで使用されます。通常、メッセージ ID は一意の ID であるため、ID-MACHINENAME-2443-1211718892437-1-0.tar などのファイル名で終わります。この動作をオーバーライドする場合は、ルートに CamelFileName ヘッダーの値を明示的に設定できます。

```
from("direct:start").setHeader(Exchange.FILE_NAME,
constant("report.txt")).marshal().tarFile().to("file:output/directory");
```

このルートにより、「report.txt」という名前の単一の Tar エントリーが含まれ、出力ディレクトリーの「report.txt.tar」という名前の Tar ファイルが作成されます。

315.3. アンマーシャリング

この例では、MY_QUEUE という ActiveMQ キューから元の形式に Tar File ペイロードをアンマーシャリングし、これを UnTarpedMessageProcessor に転送します。

```
from("activemq:queue:MY_QUEUE").unmarshal().tarFile().process(new
UnTarpedMessageProcessor());
```

Tar File に複数のエントリーがある場合は、TarFileDataFormat の usingIterator オプションを true に、Splitter を使用して追加の作業を行うことができます。

```
  TarFileDataFormat tarFile = new TarFileDataFormat();
  tarFile.setUsingIterator(true);
  from("file:src/test/resources/org/apache/camel/dataformat/tarfile/?
consumer.delay=1000&noop=true")
    .unmarshal(tarFile)
    .split(body(Iterator.class))
        .streaming()
          .process(new UnTarpedMessageProcessor())
    .end();
```

または、TarSplitter を、このように Splitter の式として直接使用できます。

```
    from("file:src/test/resources/org/apache/camel/dataformat/tarfile?
consumer.delay=1000&noop=true")
      .split(new TarSplitter())
        .streaming()
        .process(new UnTarpedMessageProcessor())
      .end();
```

315.4. AGGREGATE

INFO:Please note. this aggregation strategy requires that this aggregation completion check to work correctly.

この例では、入力ディレクトリーにあるすべてのテキストファイルを、output ディレクトリーに保存される単一の Tar ファイルに統合します。

```
    from("file:input/directory?antInclude=*/.txt")
      .aggregate(new TarAggregationStrategy())
        .constant(true)
        .completionFromBatchConsumer()
        .eagerCheckCompletion()
    .to("file:output/directory");
```

発信 CamelFileName メッセージヘッダーは、java.io.File.createTempFile と ".tar" 接尾辞を使用して作成されます。 この動作をオーバーライドする場合は、ルートに CamelFileName ヘッダーの値を明示的に設定できます。

```
    from("file:input/directory?antInclude=*/.txt")
      .aggregate(new TarAggregationStrategy())
        .constant(true)
        .completionFromBatchConsumer()
        .eagerCheckCompletion()
      .setHeader(Exchange.FILE_NAME, constant("reports.tar"))
    .to("file:output/directory");
```

315.5. 依存関係

Camel ルートで Tar Files を使用するには、このデータ形式を実装する camel-tarfile の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加できます。バージョン番号は最新の最新のリリースに置き換えてください（最新バージョンのダウンロードページを参照）。

```
<dependency>
```

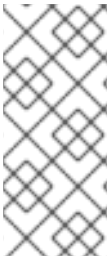
```
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-tarfile</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第316章 TELEGRAM コンポーネント

Camel バージョン 2.18 から利用可能

Telegram コンポーネントは Telegram Bot API へのアクセスを提供します。これにより、Camel ベースのアプリケーションは Bot として動作することでメッセージを送受信できます。これは、通常のユーザー、プライベート、パブリックグループまたはチャネルとの直接対話に参加します。

Telegram Bot developers のホームの手順にしたがって、このコンポーネントを使用する前に Telegram Bot を作成する必要があります。新しい Bot が作成されると、BotFather は Bot に対応する承認トークン を提供します。承認トークンは camel-telegram エンドポイントに必須のパラメーターです。



注記

Bot がグループまたはチャンネル内で交換されたすべてのメッセージを受信できるようにするには、/setprivacy コマンドを使用して BotFather に プライバシーモードを無効 にするよう依頼します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-telegram</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

316.1. URI 形式

```
telegram:type/authorizationToken[?options]
```

URI にクエリーオプションを追加するには、?option=value&option=value&…

316.2. オプション

Telegram コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
authorizationToken (security)	情報がエンドポイントで提供されない場合に使用されるデフォルトの Telegram 承認トークン。		文字列
resolvePropertyPlaceholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Telegram エンドポイントは URI 構文を使用して設定されます。

```
telegram:type/authorizationToken
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

316.2.1. パスパラメーター（2 パラメーター）：

Name	説明	デフォルト	Type
type	**必要な** エンドポイントタイプ。現時点では、「bots」タイプのみがサポートされています。		文字列
authorizationToken	ボットを使用するための認証トークン（BotFather マスク）、eg. 654321531:HGF_dTra456323dHuOedsE343211fqr3t-H です。		文字列

316.2.2. クエリーパラメーター（22 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean

Name	説明	デフォルト	Type
制限 （コンシューマー）	1 つのポーリング要求で受信できる更新の数を制限します。	100	整数
sendEmptyMessageWhenIdle (consumer)	ポーリングコンシューマーがファイルをポーリングしなかった場合、このオプションを有効にして、代わりに空のメッセージ (ボディーなし) を送信できます。	false	boolean
タイムアウト （コンシューマー）	長いポーリングのタイムアウト（秒単位）。短いポーリングの場合は 0 を、長いポーリングの数値が大きいようにします。ポーリングが長いと応答時間が短くなります。	30	整数
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
pollStrategy (consumer)	プラグ可能な org.apache.camel.PollingConsumerPollingStrategy を使用すると、エクスチェンジが作成され、Camel でルーティングされる前に、通常はポーリング操作中に発生するエラー処理を制御するカスタム実装が提供できます。		PollingConsumerPoll Strategy
chatId (producer)	生成されたメッセージを受信するチャットの識別子。チャット ID は、最初に受信メッセージから取得できます（たとえば、テレグラムユーザーがボットと対話を開始すると、クライアントはチャット ID を含む '/start' メッセージを自動的に送信します）。チャット ID は（本文またはヘッダーを使用して）送信メッセージごとに動的に設定できるので、オプションのパラメーターです。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
backoffErrorThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のエラーポーリング (エラーによって失敗した) の数。		int

Name	説明	デフォルト	Type
backoffIdleThreshold (scheduler)	backoffMultipler が開始する前に発生する必要がある後続のアイドルポーリングの数。		int
backoffMultiplier (scheduler)	後続のアイドル状態/エラーが連続して発生した場合に、スケジュールされたポーリングコンシューマーのバックオフを許可します。乗数は、実際に次の試行が行われる前にスキップされるポーリングの数です。このオプションが使用されている場合は、backoffIdleThreshold や backoffErrorThreshold も設定する必要があります。		int
遅延 （スケジューラー）	次のポーリングまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	500	Long
greedy (scheduler)	greedy が有効で、以前の実行が 1 つ以上のメッセージをポーリングした場合、ScheduledPollConsumer は即座に再度実行されます。	false	boolean
initialDelay (scheduler)	最初のポーリングが開始されるまでの時間 (ミリ秒単位)。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
runLoggingLevel (scheduler)	コンシューマーはポーリング時に開始/完了のログ行を記録します。このオプションを使用すると、ログレベルを設定できます。	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	コンシューマーに使用するカスタム/共有スレッドプールを設定できます。デフォルトでは、各コンシューマーに独自の単一スレッドのスレッドプールがあります。		ScheduledExecutor Service
scheduler (scheduler)	camel-spring または camel-quartz2 コンポーネントからの cron スケジューラーを使用するには、以下を実行します。	none	ScheduledPollConsumer Scheduler
schedulerProperties (scheduler)	カスタムスケジューラーまたは Quartz2、Spring ベースのスケジューラーを使用する場合に追加のプロパティーを設定します。		マップ
startScheduler (scheduler)	スケジューラーを自動起動するかどうか。	true	boolean

Name	説明	デフォルト	Type
timeUnit (scheduler)	initialDelay および delay オプションの時間単位。	ミリ秒	TimeUnit
useFixedDelay (scheduler)	固定遅延または固定レートを使用するかどうかを制御します。詳細は、JDK の ScheduledExecutorService を参照してください。	true	boolean

316.3. メッセージヘッダー

Name	説明
CamelTelegramChatId	このヘッダーは、メッセージを受信するチャット ID を解決するためにプロデューサーエンドポイントによって使用されます。Recipient chat id は、メッセージボディー、**CamelTelegramChatId** ヘッダー、またはエンドポイント設定（**chatId** オプション）で配置できます。このヘッダーはすべての受信メッセージにも存在します。
CamelTelegramMediaType	このヘッダーは、送信メッセージが純粋なバイナリーデータで構成される場合にメディアタイプを識別するために使用されます。使用できる値は、**org.apache.camel.component.telegram.TelegramMediaType** 列挙に属する文字列または enum の値です。
CamelTelegramMediaTitleCaption	このヘッダーは、送信バイナリーメッセージのキャプションまたはタイトルを提供するために使用されます。
CamelTelegramParseMode	このヘッダーは、HTML またはマークダウンを使用してテキストメッセージをフォーマットするために使用されます（**org.apache.camel.component.telegram.TelegramParseMode**を参照）。

316.4. 用途

Telegram コンポーネントは、コンシューマーおよびプロデューサーエンドポイントの両方をサポートします。リアクティブなチャットバッキングモード（メッセージの消費、生成に 使用）でも使用することができます。

316.5. プロデューサーの例

以下は、Telegram Bot API 経由で Telegram チャットにメッセージを送信する基本的な例です。

in Java DSL

```
from("direct:start").to("telegram:bots/123456789:insertAuthorizationTokenHere");
```

または Spring XML で行う

```
<route>
    <from uri="direct:start"/>
    <to uri="telegram:bots/123456789:insertAuthorizationTokenHere"/>
<route>
```

123456789:insertAuthorizationTokenHere コードは Bot に対応する 認証トークン です。

チャット ID オプションを指定せずにプロデューサーエンドポイントを使用する場合、メッセージの本文またはヘッダーに含まれる情報を使用してターゲットチャットが識別されます。プロデューサーエンドポイントには、以下のメッセージ本文が許可されます（タイプ OutgoingXXXMessage のメッセージはパッケージ org.apache.camel.component.telegram.modelに属します）。

Java タイプ	説明
OutgoingTextMessage	テキストメッセージをチャットに送信する
OutgoingPhotoMessage	写真（JPG、PNG）をチャットに送信する
OutgoingAudioMessage	mp3 音声をチャットに送信する
OutgoingVideoMessage	mp4 ビデオをチャットに送信する
OutgoingDocumentMessage	ファイルをチャット（任意のメディアタイプ）に送信する
byte[]	サポートされているメディアタイプを送信する。**CamelTelegramMediaType** ヘッダーを適切なメディアタイプに設定する必要があります。
文字列	テキストメッセージをチャットに送信する。自動的に **OutgoingTextMessage**に変換されます。

316.6. コンシューマーの例

以下は、テレグラムユーザーが設定済みの Bot に送信しているすべてのメッセージを受信する基本的な例です。Java DSL の場合

```
from("telegram:bots/123456789:insertAuthorizationTokenHere")
.bean(ProcessorBean.class)
```

または Spring XML で行う

```
<route>
    <from uri="telegram:bots/123456789:insertAuthorizationTokenHere"/>
    <bean ref="myBean" />
<route>

<bean id="myBean" class="com.example.MyBean"/>
```

MyBean は、メッセージを受信する簡単な Bean です。

```
public class MyBean {

    public void process(String message) {
        // or Exchange, or org.apache.camel.component.telegram.model.IncomingMessage (or
both)

        // do process
    }

}
```

受信メッセージでサポートされるタイプは次のとおりです。

Java タイプ	説明
IncomingMessage	受信メッセージの完全なオブジェクト表現
文字列	メッセージの内容（テキストメッセージのみ）

316.7. リアクティブ CHAT-BOT の例

reactive chat-bot モードは、Camel コンポーネントを使用して Telegram ユーザーから受信したチャットメッセージに直接返信する単純なチャットボットをビルドする簡単な方法です。

以下は、Java DSL における chat-bot の基本設定です。

```
from("telegram:bots/123456789:insertAuthorizationTokenHere")
.bean(ChatBotLogic.class)
.to("telegram:bots/123456789:insertAuthorizationTokenHere");
```

または Spring XML で行う

```
<route>
    <from uri="telegram:bots/123456789:insertAuthorizationTokenHere"/>
    <bean ref="chatBotLogic" />
    <to uri="telegram:bots/123456789:insertAuthorizationTokenHere"/>
<route>

<bean id="chatBotLogic" class="com.example.ChatBotLogic"/>
```

ChatBotLogic は、一般的な String-to-String メソッドを実装する単純な Bean です。

```
public class ChatBotLogic {

    public String chatBotProcess(String message) {
        if( "do-not-reply".equals(message) ) {
            return null; // no response in the chat
        }

        return "echo from the bot: " + message; // echoes the message
    }

}
```

chatBotProcess メソッドによって返されるすべての null でない文字列は、リクエストの発信されたチャットに自動的にルーティングされます（ CamelTelegramChatId ヘッダーはメッセージのルーティングに使用されます）。

316.8. CHAT ID の取得

イベント発生時にメッセージを特定の Telegram チャットにプッシュする場合は、対応するチャット ID を取得する必要があります。現在、チャット ID は telegram クライアントには表示されませんが、単純なルートを使用して取得できます。

まず、ボットをメッセージのプッシュ先のチャットに追加し、以下のようなルートを実行します。

```
from("telegram:bots/123456789:insertAuthorizationTokenHere")
.to("log:INFO?showHeaders=true");
```

ボットが受信したメッセージは、チャット（CamelTelegramChatId ヘッダー）に関する情報と共にログにダンプされます。

チャット ID を取得すると、以下のサンプルルートを使用してメッセージをプッシュできます。

```
from("timer:tick")
.setBody().constant("Hello")
to("telegram:bots/123456789:insertAuthorizationTokenHere?chatId=123456")
```

対応する URI パラメーターは単に chatId であることに注意してください。

第317章 テストコンポーネント

Camel バージョン 1.3 で利用可能

分散および非同期処理のテストは非常に困難です。**Mock**、**Test**、**DataSet** エンドポイントは、Camel テストフレームワークにより優れた機能を提供し、*エンタープライズ統合パターンと Camel の多くのコンポーネントと強力な Bean 統合* とのさまざまなコンポーネントを使用して、ユニットテストと統合テストを簡素化します。

テスト コンポーネントは **Mock** コンポーネントを拡張し、起動時に別のエンドポイントからのメッセージプルをサポートし、基礎となる **Mock** エンドポイントで想定されるメッセージ本文を設定します。つまり、ルートでテストエンドポイントを使用し、到達したメッセージは暗黙的に、他の場所から抽出される想定されるメッセージと比較されます。

したがって、たとえば、予想されるメッセージボディーをファイルとして使用できます。これにより、適切に設定された **Mock** エンドポイントが設定されます。これは、受信したメッセージが想定されるメッセージの数と一致する場合にのみ有効で、メッセージペイロードが等しくなります。

Camel 2.8 以前を使用している場合は、Maven ユーザーはこのコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Camel 2.9 以降では、**Test** コンポーネントは camel-core に直接提供されます。

317.1. URI 形式

```
test:expectedMessagesEndpointUri
```

expectedMessagesEndpointUri は、テストを開始する前に想定されるメッセージ本文がプルされるその他のコンポーネント URI を参照します。

317.2. URI オプション

Test コンポーネントにはオプションがありません。

Test エンドポイントは、URI 構文を使用して設定します。

```
test:name
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

317.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
name	テストに使用されるメッセージのポーリングに使用するレジストリーで検索するエンドポイントの名前。		文字列

317.2.2. クエリーパラメーター（14 パラメーター）：

Name	説明	デフォルト	Type
anyOrder (producer)	予想されるメッセージが同じ順序で到達するか、または任意の順序で受信できるかどうか。	false	boolean
assertPeriod (producer)	モックエンドポイントが再アサートされる猶予期間を設定し、事前のアサーションが有効であることを確認します。これは、たとえば多数のメッセージが到着することをアサートするために使用されます。たとえば、expectMessageCount(int)が 5 に設定されている場合、5 つ以上のメッセージが到達するとアサーションが満たされます。5 つのメッセージが到着するようにするには、さらにメッセージが到達しないように、少し待機する必要があります。これは、このリンク setAssertPeriod(long)メソッドを使用できます。デフォルトでは、この期間は無効です。	0	Long
delimiter (producer)	分割が有効な場合に使用する分割区切り文字。デフォルトでは、区切り文字は改行ベースです。区切り文字は正規表現になります。		文字列

Name	説明	デフォルト	Type
expectedCount (producer)	このエンドポイントによって受信されるべきメッセージエクスチェンジの数を指定します。注意： 0 のメッセージを想定したい場合は、テストの開始時に 0 がマッチするので、アサート期間を設定して、メッセージが到達しないようにする必要があります。そのため、リンク setAssertPeriod(long)を使用します。別の方法として、NotifyBuilder を使用して、モックでリンク assertIsSatisfied（）メソッドを呼び出す前に、Camel がメッセージをルーティングするタイミングを把握することができます。これにより、テスト時間を短縮するために、固定されたアサート期間を使用できません。n 番目のメッセージがこのモックエンドポイントに到達することをアサートする場合は、詳細については、リンク setAssertPeriod(long)メソッドも参照してください。	-1	int
reportGroup (producer)	サイズのグループに基づいてスループットロギングを有効にするために使用される数。		int
resultMinimumWaitTime (producer)	リンクアサートIsSatisfied（）が満たされるまで、リンクアサートが待機する最小時間（ミリ秒単位）を設定します。	0	Long
resultWaitTime (producer)	リンクアサート IsSatisfied（）が満たされるまでリンクアサートが待機する最大時間（ミリ秒単位）を設定します。	0	Long
retainFirst (producer)	受信されたエクスチェンジの最初の数だけを保持するように指定します。これは、このモックエンドポイントを受け取るすべての Exchange のコピーを保存せず、大量のデータでテストする際に使用され、メモリー消費を削減します。重要： この制限を使用すると、リンク getReceivedCounter（）は受信したエクスチェンジの実際の数を返します。たとえば、5000 Exchange を受け取ったため、最初の 10 Exchange のみを保持するように設定している場合は、リンク getReceivedCounter（）は引き続き 5000 を返しますが、リンク getExchanges（）とリンク getReceivedExchanges（）メソッドの最初の 10 エクスチェンジのみがあります。この方法を使用する場合は、想定される他の方法の一部はサポートされません。たとえば、リンク expectBodiesReceived(Object...)は受信される最初の本文数に期待を設定します。リンク setRetainFirst(int)メソッドおよびリンク setRetainLast(int)メソッドの両方を設定して、最初に受信した最初と最後に受信した両方を制限できます。	-1	int

Name	説明	デフォルト	Type
retainLast (producer)	最後に受信したエクスチェンジの数のみを保持するように指定します。これは、このモックエンドポイントを受け取るすべての Exchange のコピーを保存せず、大量のデータでテストする際に使用され、メモリー消費を削減します。重要： この制限を使用すると、リンク getReceivedCounter（）は受信したエクスチェンジの実際の数を返します。たとえば、5000 Exchange を受け取ったときに最後の 20 Exchange のみを保持するよう設定されている場合、リンク getReceivedCounter（）は引き続き 5000 を返しますが、リンク getExchanges（）とリンク getReceivedExchanges（）メソッドの最後の 20 Exchange のみが存在します。この方法を使用する場合は、想定される他の方法の一部はサポートされません。たとえば、リンク expectBodiesReceived(Object...)は受信される最初の本文数に期待を設定します。リンク setRetainFirst(int)メソッドおよびリンク setRetainLast(int)メソッドの両方を設定して、最初に受信した最初と最後に受信した両方を制限できます。	-1	int
sleepForEmptyTest (producer)	expectedMessageCount(int)がゼロで呼び出されると、このエンドポイントが実際に空であることを確認できるようスリープを指定できます。	0	Long
split (producer)	有効にすると、test エンドポイントからロードされたメッセージは新しい行区切り文字を使用して分割され、各行が想定されるメッセージになります。たとえば、File エンドポイントを使用して、各行が想定されるメッセージであるファイルを読み込みます。	false	boolean
タイムアウト （プロデューサー）	URI からのメッセージボディーのポーリング時に使用するタイムアウト。	2000	Long
copyOnExchange (producer)	このモックエンドポイントで受信されたときに受信エクスチェンジのディープコピーを作成するかどうかを設定します。デフォルトは true です。	true	boolean
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

317.3. 例

たとえば、以下のようにテストケースを作成できます。

```
from("seda:someEndpoint").
  to("test:file://data/expectedOutput?noop=true");
```

テストが MockEndpoint.assertIsSatisfied(camelContext)メソッド を呼び出すと、テストケースが必要なアサーションを実行します。

テストエンドポイントに他の期待値を設定する方法は、「Mock コンポーネント」を参照してください。

317.4. 関連項目

- Spring テスト

第318章 THRIFT コンポーネント

Camel バージョン 2.20 で利用可能

Thrift コンポーネントを使用すると、Apache Thrift バイナリー通信プロトコルおよびシリアライズメカニズムを使用して Remote Procedure Call(RPC)サービスを呼び出すことができます。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-thrift</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

318.1. URI 形式

```
thrift://service[?options]
```

318.2. エンドポイントオプション

Thrift コンポーネントは、以下に示す 2 つのオプションをサポートします。

Name	説明	デフォルト	Type
useGlobalSslContext Parameters (security)	thrift コンポーネントがグローバル SSL コンテキストパラメーターを使用しているかどうかを判別	false	boolean
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Thrift エンドポイントは、URI 構文を使用して設定します。

```
thrift:host:port/service
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

318.2.1. パスパラメーター（3 パラメーター）：

Name	説明	デフォルト	Type
host	Thrift サーバーのホスト名。これは、プロデューサーの使用時にコンシューマーまたはリモートサーバーのホスト名である場合に localhost または 0.0.0.0（定義されていない場合）です。		文字列
port	**必須**: Thrift サーバーポート		int
サービス	thrift 記述子ファイルに **必要な** 完全修飾サービス名（パッケージドットサービス定義名）		文字列

318.2.2. クエリーパラメーター（12 パラメーター）：

Name	説明	デフォルト	Type
compressionType (common)	プロトコル圧縮メカニズムのタイプ	NONE	ThriftCompressionType
exchangeProtocol (common)	Exchange プロトコルのシリアライズタイプ	BINARY	ThriftExchangeProtocol
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。	false	boolean
clientTimeout (consumer)	コンシューマーのクライアントタイムアウト		int
maxPoolSize (consumer)	Thrift サーバーコンシューマーの最大サイズ	10	int
poolSize (consumer)	Thrift サーバーコンシューマーの初期スレッドプールのサイズ	1	int

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN または ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	コンシューマーがエクスチェンジを作成する際に交換パターンを設定します。		ExchangePattern
メソッド （プロデューサー）	Thrift 呼び出しメソッド名		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
negotiationType (security)	セキュリティーネゴシエーションのタイプ	PLAIN TEXT	ThriftNegotiationType
sslParameters (security)	SSL/TLS セキュリティーネゴシエーションの設定パラメーター		SSLContextParameters

318.3. THRIFT メソッドパラメーターのマッピング

呼び出された手順のパラメーターは、メッセージボディー内のオブジェクトのリストとして渡す必要があります。プリミティブは、すぐにオブジェクトから変換されます。対応するメソッドを正しく検出するには、値に関係なくすべてのタイプを送信する必要があります。以下で、Camel ボディーを使用して異なるパラメーターをメソッドに渡す方法を参照してください。

```
List requestBody = new ArrayList();

requestBody.add((boolean)true);
requestBody.add((byte)THRIFT_TEST_NUM1);
requestBody.add((short)THRIFT_TEST_NUM1);
requestBody.add((int)THRIFT_TEST_NUM1);
requestBody.add((long)THRIFT_TEST_NUM1);
requestBody.add((double)THRIFT_TEST_NUM1);
requestBody.add("empty"); // String parameter
requestBody.add(ByteBuffer.allocate(10)); // binary parameter
requestBody.add(new Work(THRIFT_TEST_NUM1, THRIFT_TEST_NUM2,
Operation.MULTIPLY)); // Struct parameter
requestBody.add(new ArrayList<Integer>()); // list paramater
requestBody.add(new HashSet<String>()); // set parameter
```

```
requestBody.add(new HashMap<String, Long>()); // map parameter

Object responseBody = template.requestBody("direct:thrift-alltypes", requestBody);
```

サービスコンシューマーの受信パラメーターは、オブジェクトのリストとしてメッセージボディーにも渡されます。

318.4. THRIFT コンシューマーヘッダー（コンシューマーの呼び出し後にインストールされる）

ヘッダー名	説明	可能な値
CamelThriftMethodName	コンシューマーサービスによって処理されるメソッド名	

318.5. 例

以下は、ホストおよびポートパラメーターで呼び出される簡単な同期メソッドです。

```
from("direct:thrift-calculate")
.to("thrift://localhost:1101/org.apache.camel.component.thrift.generated.Calculator?
method=calculate&synchronous=true");
```

以下は、XML DSL 設定に対して呼び出される簡単な同期メソッドです。

```
<route>
    <from uri="direct:thrift-add" />
    <to uri="thrift://localhost:1101/org.apache.camel.component.thrift.generated.Calculator?
method=add&synchronous=true"/>
</route>
```

非同期通信のある Thrift サービスコンシューマー

```
from("thrift://localhost:1101/org.apache.camel.component.thrift.generated.Calculator")
.to("direct:thrift-service");
```

thrift-maven-plugin を使用して .thrift ファイルの Java コード生成を自動化することができますが、オペレーティングシステム用の thrift コンパイラーバイナリーディストリビューションを実行中のホストに存在する必要があります。

318.6. 詳細情報は、これらのリソースを参照してください。

Thrift プロジェクト GitHub link https://thrift.apache.org/tutorial/java [Apache Thrift Java tutorial]

318.7. 関連項目

- *はじめに*
- *Configuring Camel (Camel の設定)*
- *コンポーネント*
- *エンドポイント*

第319章 THRIFT DATAFORMAT

Camel バージョン 2.20 で利用可能

Camel は、Java と Apache Thrift の間でシリアライズする Data Format を提供します。https://thrift.apache.org/ を希望する理由のプロジェクトのサイトの詳細。Apache Thrift は言語に依存しないプラットフォームに依存しないため、Camel ルートによって生成されたメッセージは他の言語実装によって消費される可能性があります。

Apache Thrift 実装

319.1. THRIFT オプション

Thrift データフォーマットは、以下に示す 3 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
instanceClass		文字列	無効化解除時に使用するクラスの名前
contentTypeFormat	バイナリー	文字列	thrift メッセージが Java からシリアライズ/デシリアライズされるコンテンツタイプ形式を定義します。形式は、ネイティブバイナリーのthrift、json、または単純な json フィールド表現のいずれかの場合は native または json のいずれかになります。デフォルト値は binary です。
contentTypeHeader	false	ブール値	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

319.2. コンテンツタイプの形式

JSON メッセージを解析して Thrift 形式に変換し、ネイティブユーティリティーコンバーターを使用して解析し直すことができます。このオプションを使用するには、contentTypeFormat の値を 'json' に設定するか、2 番目のパラメーターで thrift を呼び出すこともできます。デフォルトインスタンスが指定されていない場合は、必ずネイティブバイナリー Thrift 形式を使用してください。単純な JSON 形式は書き込み専用(marshal)で、スクリプト言語による解析に適した単純な出力形式を生成します。サンプルコードは、以下を示しています。

```
from("direct:marshal")
    .unmarshal()
    .thrift("org.apache.camel.dataformat.thrift.generated.Work", "json")
    .to("mock:reverse");
```

319.3. THRIFT の概要

Thrift の使用方法に関する簡単な概要詳細は、完全なチュートリアルを参照してください。

319.4. THRIFT 形式の定義

最初のステップでは、エクスチェンジのボディーの形式を定義します。これは、以下のように .thrift ファイルで定義されます。

tutorial.thrift

```
namespace java org.apache.camel.dataformat.thrift.generated

enum Operation {
  ADD = 1,
  SUBTRACT = 2,
  MULTIPLY = 3,
  DIVIDE = 4
}

struct Work {
  1: i32 num1 = 0,
  2: i32 num2,
  3: Operation op,
  4: optional string comment,
}
```

319.5. JAVA クラスの生成

Apache Thrift は、.thrift ファイルで定義した形式の Java クラスを生成するコンパイラーを提供します。

手動で必要となる追加の対応言語に対してコンパイラーを実行することもできます。

```
thrift -r --gen java -out ../java/ ./tutorial-dataformat.thrift
```

これにより、.thrift ファイル（struct または enum など）で定義されている各タイプに個別の Java クラスが生成されます。生成されたクラスは、シリアライゼーションメカニズムで必要な org.apache.thrift.TBase を実装します。このため、これらのクラスのみがエクスチェンジのボディーで使用されることが重要です。org.apache.thrift.TBase を実装しないクラスを使用するように Data Format に指示すると、Camel はルートの作成時に例外をスローします。

319.6. JAVA DSL

ThriftDataFormat インスタンスを作成して、Camel DataFormat のマーシャリングおよびアンマーシャリング API に渡すことができます。

```
ThriftDataFormat format = new ThriftDataFormat(new Work());

from("direct:in").marshal(format);
from("direct:back").unmarshal(format).to("mock:reverse");
```

または、DSL thrift（）を使用して、デフォルトのインスタンスまたはデフォルトのインスタンスクラス名をアンマーシャリングします。

```
// You don't need to specify the default instance for the thrift marshaling
from("direct:marshal").marshal().thrift();
from("direct:unmarshalA").unmarshal()
    .thrift("org.apache.camel.dataformat.thrift.generated.Work")
    .to("mock:reverse");

from("direct:unmarshalB").unmarshal().thrift(new Work()).to("mock:reverse");
```

319.7. SPRING DSL

以下の例は、Trift を使用して Thrift データ型を設定する thrift を使用してアンマーシャリングする方法を示しています。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <unmarshal>
      <thrift instanceClass="org.apache.camel.dataformat.thrift.generated.Work" />
    </unmarshal>
    <to uri="mock:result"/>
  </route>
</camelContext>
```

319.8. 依存関係

camel ルートで Thrift を使用するには、このデータ形式を実装する camel-thrift の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-thrift</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第320章 TIDYMARKUP DATAFORMAT

Camel バージョン 2.0 で利用可能

TidyMarkup は、TagSoup を使用して HTML を連携させるデータ形式です。不安定な HTML を解析し、非常に適切に形式の HTML を返すために使用できます。

camel eats our own -dog food- soap

PDF マニュアルにいくつかの問題があり、いくつかのレジョナルシンボルを用意しました。そのため Jonathan はこのデータ形式を使用して、pdf マニュアルのレンダリングのベースとして使用される wiki html ページを連携させます。そして、mysterious symbol vanished.

TidyMarkup は アンマーシャリング 操作のみをサポートします。これは、HTML の公式な HTML に変換したくないためです。

320.1. TIDYMARKUP オプション

TidyMarkup データフォーマットは、以下に示す 3 つのオプションをサポートします。

Name	デフォルト	Java タイプ	説明
dataObjectType	**org.w3c.dom.Node**	**文字列**	アンマーシャリングするデータ型は、org.w3c.dom.Node または java.lang.String のいずれかになります。デフォルトは org.w3c.dom.Node です。
omitXmlDeclaration	**false**	**ブール値**	String を返す場合は、上部の XML 宣言を省略します。
contentTypeHeader	**false**	**ブール値**	データフォーマットがデータ形式を実行できる場合に、データ形式がデータ形式の型で Content-Type ヘッダーを設定するかどうか。たとえば、XML へのデータフォーマットの application/xml、または JSon へのデータフォーマットの application/json など。

320.2. JAVA DSL の例

コンシューマーが一部の HTML を提供する例

```
from("file://site/inbox").unmarshal().tidyMarkup().to("file://site/blogs");
```

320.3. SPRING XML の例

以下の例は、Spring を使用して TidyMarkup を使用してアンマーシャリングする方法を示しています。

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="file://site/inbox"/>
    <unmarshal>
      <tidyMarkup/>
    </unmarshal>
    <to uri="file://site/blogs"/>
  </route>
</camelContext>
```

320.4. 依存関係

camel ルートで TidyMarkup を使用するには、このデータ形式を実装する camel-tagsoup の依存関係を追加する必要があります。

Maven を使用する場合は、以下を pom.xml に追加するだけで、最新かつ最大のリリースのバージョン番号を置き換えます（最新バージョンのダウンロードページを参照）。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-tagsoup</artifactId>
  <version>x.x.x</version>
</dependency>
```

第321章 TIKA コンポーネント

Camel バージョン 2.19 から利用可能

Tika: コンポーネントは、Apache Tika でドキュメントを検出して解析する機能を提供します。このコンポーネントは、ドキュメントを操作する基礎となるライブラリーとして Apache Tika を使用します。

Tika コンポーネントを使用するには、Maven ユーザーは以下の依存関係を pom.xml に追加する必要があります。

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-tika</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

TIKA コンポーネントはプロデューサーエンドポイントのみをサポートします。

321.1. オプション

Tika コンポーネントにはオプションがありません。

Tika エンドポイントは、URI 構文を使用して設定します。

```
tika:operation
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

321.1.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
operation	**必要な** Tika 操作。解析または検出		TikaOperation

321.1.2. クエリーパラメーター（5 パラメーター）：

Name	説明	デフォルト	Type
tikaConfig (producer)	tika Config		TikaConfig
tikaConfigUri (producer)	tika Config Uri: tika-config.xml の URI		文字列
tikaParseOutputEncoding (producer)	Tka Parse Output Encoding: 解析された出力の文字エンコーディングを指定するために使用されます。Defaults to Charset.defaultCharset() .		文字列
tikaParseOutputFormat (producer)	tika 出力形式。サポートされる出力形式。xml: 解析したコンテンツを XML として返します。html: 解析したコンテンツを HTML として返します。textMain: boilerpipe ライブラリーを使用して、Web ページからメインコンテンツを自動的に抽出します。	xml	TikaParseOutputFormat
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean

321.2. ファイルの MIME タイプを検出

ファイルは Body に配置する必要があります。

```
from("direct:start")
        .to("tika:detect");
```

321.3. ファイルを解析する

ファイルは Body に配置する必要があります。

```
from("direct:start")
        .to("tika:parse");
```

第322章 タイマーコンポーネント

Camel バージョン 1.0 で利用可能

timer: コンポーネントは、タイマーが実行するとメッセージ交換を生成するために使用されます。このエンドポイントからのみイベントを消費できます。

322.1. URI 形式

```
timer:name[?options]
```

name は、エンドポイント全体で作成され、共有される Timer オブジェクトの名前です。そのため、すべてのタイマーエンドポイントに同じ名前を使用する場合は、1 つの Timer オブジェクトとスレッドのみが使用されます。

URI にクエリーオプションを追加するには、 ?option=value&option=value&…

注記： 生成されたエクスチェンジの IN ボディーは null です。So exchange.getIn().getBody() returns null.

TIP:*Advanced Scheduler* より高度なスケジューリングに対応する Quartz コンポーネントも参照してください。

TIP:* 人間が平易な形式で時間を指定* Camel 2.3 以降では、人間が解読しやすい構文 で時間を指定することができます。

322.2. オプション

Timer コンポーネントにはオプションがありません。

Timer エンドポイントは、URI 構文を使用して設定します。

```
timer:timerName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

322.2.1. パスパラメーター（1 パラメーター）：

Name	説明	デフォルト	Type
timerName	**必要な** タイマーの名前		文字列

322.2.2. クエリーパラメーター（12 パラメーター）：

Name	説明	デフォルト	Type
bridgeErrorHandler (consumer)	コンシューマーの Camel ルーティングエラーハンドラーへのブリッジを許可します。よって、コンシューマーが受信メッセージなどの取得を試行している間に発生した例外は、メッセージとして処理され、ルーティングエラーハンドラーによって処理されます。デフォルトでは、コンシューマーは org.apache.camel.spi.ExceptionHandler を使用して例外に対応し、WARN/ERROR レベルでログに記録され無視されます。	false	boolean
遅延 （コンシューマー）	最初のイベントが生成されるまで待機する時間（ミリ秒単位）。time オプションと併用しないでください。デフォルト値は 1000 です。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
fixedRate (consumer)	イベントは、指定した期間で区切られた、約一定間隔で実行されます。	false	boolean
期間 （コンシューマー）	0 を超える場合は、期間（ミリ秒単位）ごとに定期的なイベントを生成します。デフォルト値は 1000 です。また、60 秒（60 秒）、5m30s（5 分と 30 秒）、および 1h（1 時間）などの単位を使用して時間の値を指定することもできます。	1000	Long
repeatCount （コンシューマー）	実行の最大数を指定します。したがって、これを 1 に設定すると、タイマーは 1 度だけ実行されます。これを 5 に設定した場合、5 回だけ実行されます。0 または負の値を設定すると、無制限に実行されます。	0	Long

Name	説明	デフォルト	Type
exceptionHandler (consumer)	コンシューマーによるカスタム ExceptionHandler の使用を許可します。bridgeErrorHandler オプションが有効な場合は、このオプションは使用されていないことに注意してください。デフォルトでは、コンシューマーは例外に対応し、WARN/ERROR レベルでログに記録され、無視されます。		ExceptionHandler
exchangePattern (consumer)	エクスチェンジの作成時にデフォルトの交換パターンを設定します。		ExchangePattern
デーモン （詳細）	タイマーエンドポイントに関連付けられたスレッドがデーモンとして実行されるかどうかを指定します。デフォルト値は true です。	true	boolean
パターン （詳細）	URI 構文を使用して time オプションの設定に使用するカスタムの Date パターンを指定できます。		文字列
同期 （詳細）	同期処理を厳密に使用するか、Camel が非同期処理を使用できるようにするかを設定します（サポートされている場合）。	false	boolean
時間 （詳細）	最初のイベントを生成する java.util.Date。URI を使用する場合、想定されるパターンは yyyy-MM-dd HH:mm:ss または yyyy-MM-dd'T'HH:mm:ss です。		Date
timer (advanced)	カスタムタイマーの使用		Timer

322.3. エクスチェンジプロパティー

タイマーが実行されると、エクスチェンジに以下の情報がプロパティーとして追加されます。

Name	タイプ	説明
Exchange.TIMER_NAME	**文字列**	**name** オプションの値。
Exchange.TIMER_TIME	**Date**	**time** オプションの値。

Name	タイプ	説明
Exchange.TIMER_PERIOD	**Long**	**period** オプションの値。
Exchange.TIMER_FIRED_TIME	**Date**	コンシューマーが実行される時間。
Exchange.TIMER_COUNTER	**Long**	Camel 2.8: 現在の実行カウンター。1 から始まります。

322.4. 例

60 秒ごとにイベントを生成するルートを設定するには、以下を実行します。

```
from("timer://foo?fixedRate=true&period=60000").to("bean:myBean?
method=someMethodName");
```

ヒント

60000 の代わりに、period=60s を使用できます。これはより読みやすいものになります。

上記のルートはイベントを生成し、JNDI や Spring などのレジストリーで myBean という Bean で someMethodName メソッドを呼び出します。

Spring DSL のルートの場合：

```
<route>
  <from uri="timer://foo?fixedRate=true&amp;period=60000"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

322.5. できるだけ早く実行

Camel 2.17 から利用可能

Camel ルートでメッセージをできるだけ早く実行する場合は、負の遅延を使用できます。

```
<route>
  <from uri="timer://foo?delay=-1"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

これにより、タイマーはメッセージを即座に実行します。

固定の数に達した後にメッセージの発生を停止するために、負の遅延とともに **repeatCount** パラメーターを指定することもできます。

repeatCount を指定しない場合、タイマーはルートが停止するまでメッセージを実行し続けます。

322.6. 実行は 1 回のみ

Camel 2.8 から利用可能

ルートの起動時など、Camel ルートでメッセージを 1 度だけ実行したい場合があります。これには、以下のように **repeatCount** オプションを指定します。

```
<route>
  <from uri="timer://foo?repeatCount=1"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

322.7. 関連項目

- スケジューラー
- **Quartz**

第323章 TWILIO コンポーネント

Camel バージョン 2.20 で利用可能

Twilio コンポーネントは、Twilio Java SDK を使用してアクセス可能な Twilio REST API のバージョン 2010-04-01 へのアクセスを提供します。

Maven ユーザーは、このコンポーネントの pom.xml に以下の依存関係を追加する必要があります。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-twilio</artifactId>
  <version>${camel-version}</version>
</dependency>
```

323.1. TWILIO オプション

Twilio コンポーネントは、以下に示す 3 つのオプションをサポートします。

Name	説明	デフォルト	Type
Configuration (advanced)	共有設定の使用		TwilioConfiguration
restClient (advanced)	共有 REST クライアントの使用		TwilioRestClient
resolveProperty Placeholders (advanced)	起動時にコンポーネント自体がプロパティープレースホルダーを解決するかどうか。String タイプのプロパティーのみがプロパティープレースホルダーを使用できます。	true	boolean

Twilio エンドポイントは URI 構文を使用して設定します。

```
twilio:apiName/methodName
```

以下の path パラメーターおよびクエリーパラメーターを使用します。

323.1.1. パスパラメーター（2 パラメーター）：

Name	説明		

323.1.2.

323.2.

```
twilio://endpoint-prefix/endpoint?[options]
```

-
-
-

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

-
-
-
-
-
-
-
-
-
-
-
-

323.3.

323.4.

323.5.

323.6.

第324章

-
-
-
-

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-twitter</artifactId>
  <version>${camel-version}</version>
</dependency>
```

324.1.

324.2.

324.3.

324.4.

324.5.

注記

324.5.1.

```
from("direct:foo")
  .to("twitter-timeline://user?consumerKey=[s]&consumerSecret=[s]&accessToken=
[s]&accessTokenSecret=[s]);
```

324.5.2.

```
from("twitter-timeline://home?type=polling&delay=60&consumerKey=[s]&consumerSecret=
[s]&accessToken=[s]&accessTokenSecret=[s]")
  .to("bean:blah");
```

324.5.3.

```
from("twitter-search://foo?type=polling&keywords=camel&consumerKey=
[s]&consumerSecret=[s]&accessToken=[s]&accessTokenSecret=[s]")
  .to("bean:blah");
```

324.5.4.

```
from("direct:foo")
  .to("twitter-search://foo?keywords=camel&consumerKey=[s]&consumerSecret=
[s]&accessToken=[s]&accessTokenSecret=[s]");
```

324.5.5.

```
from("direct:foo")
  .setHeader("CamelTwitterKeywords", header("bar"))
  .to("twitter-search://foo?consumerKey=[s]&consumerSecret=[s]&accessToken=
[s]&accessTokenSecret=[s]");
```

324.6.

324.7.

-
-

-
-
-

第325章

325.1.

325.2.

twitter-directmessage:user

325.2.1.

325.2.2.

第326章

326.1.

326.2.

```
twitter-search:keywords
```

326.2.1.

326.2.2.

第327章

327.1.

327.2.

twitter-streaming:streamingType

327.2.1.

327.2.2.

第328章

328.1.

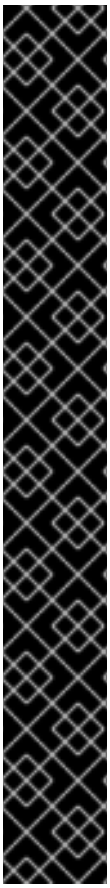
328.2.

```
twitter-timeline:timelineType
```

328.2.1.

328.2.2.

第329章



重要

-
  -
  -
  -
  -

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-twitter</artifactId>
  <version>${camel-version}</version>
</dependency>
```

329.1.

```
twitter://endpoint[?options]
```

329.2.

329.3.

329.4.

329.5.

> *twitter:kind*

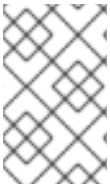
329.5.1.

329.5.2.

329.6.

329.7.

329.8.



注記

329.8.1.

```
from("direct:foo")
  .to("twitter://timeline/user?consumerKey=[s]&consumerSecret=[s]&accessToken=
[s]&accessTokenSecret=[s]);
```

329.8.2.

```
from("twitter://timeline/home?type=polling&delay=60&consumerKey=[s]&consumerSecret=
[s]&accessToken=[s]&accessTokenSecret=[s]")
  .to("bean:blah");
```

329.8.3.

```
from("twitter://search?type=polling&keywords=camel&consumerKey=[s]&consumerSecret=
[s]&accessToken=[s]&accessTokenSecret=[s]")
  .to("bean:blah");
```

329.8.4.

```
from("direct:foo")
  .to("twitter://search?keywords=camel&consumerKey=[s]&consumerSecret=
[s]&accessToken=[s]&accessTokenSecret=[s]");
```

329.8.5.

```
from("direct:foo")
  .setHeader("CamelTwitterKeywords", header("bar"))
  .to("twitter://search?consumerKey=[s]&consumerSecret=[s]&accessToken=
[s]&accessTokenSecret=[s]");
```

329.9.

329.10.

-
-
-
-
-

第330章

ヒント

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-undertow</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

330.1.

```
undertow:http://hostname[:port][/resourceUri][?options]
undertow:https://hostname[:port][/resourceUri][?options]
undertow:ws://hostname[:port][/resourceUri][?options]
undertow:wss://hostname[:port][/resourceUri][?options]
```

330.2.

undertow:httpURI

330.2.1.

330.2.2.

330.3.

330.4.

```
from("direct:start")
    .to("undertow:http://www.google.com");
```

```
<route>
    <from uri="direct:start"/>
    <to uri="undertow:http://www.google.com"/>
<route>
```

330.5.

```
<route>
  <from uri="undertow:http://localhost:8080/myapp/myservice"/>
  <to uri="bean:myBean"/>
</route>
```

330.6.

```
<route>
  <from uri="undertow:ws://localhost:8080/myapp/mysocket"/>
  <transform><simple>Echo ${body}</simple></transform>
  <to uri="undertow:ws://localhost:8080/myapp/mysocket"/>
</route>
```

330.7.

330.8.

```
<subsystem xmlns="urn:jboss:domain:undertow:4.0">
    <buffer-cache name="default" />
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true" />
```

```
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true" />
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content" />
            <filter-ref name="server-header" />
            <filter-ref name="x-powered-by-header" />
            <http-invoker security-realm="ApplicationRealm" />
        </host>
    </server>
</subsystem>
```

-
-

```
from("undertow:http://somehost:1234/path/to/resource")
```

```
[org.wildfly.extension.camel] (pool-2-thread-1) Ignoring configured host:
http://somehost:1234/path/to/resource
```

330.8.1.

330.8.2.

第331章

-
-
-

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-univocity-parsers</artifactId>
  <version>x.x.x</version>
</dependency>
```

331.1.

331.2.

331.3.

-
-

331.3.1.

```
<route>
    <from uri="direct:input"/>
```

```
    <marshal>
      <univocity-csv/>
    </marshal>
    <to uri="mock:result"/>
</route>
```

331.3.2.

```
<route>
    <from uri="direct:input"/>
    <marshal>
      <univocity-fixed padding="_">
        <univocity-header length="5"/>
        <univocity-header length="5"/>
        <univocity-header length="5"/>
      </univocity-fixed>
    </marshal>
    <to uri="mock:result"/>
</route>
```

331.3.3.

```
<route>
    <from uri="direct:input"/>
    <marshal>
      <univocity-tsv/>
    </marshal>
    <to uri="mock:result"/>
</route>
```

331.4.

-
-

-
-

331.4.1.

```
<route>
    <from uri="direct:input"/>
    <unmarshal>
        <univocity-csv headerExtractionEnabled="true" asMap="true"/>
    </unmarshal>
    <to uri="mock:result"/>
</route>
```

331.4.2.

```
<route>
    <from uri="direct:input"/>
    <unmarshal>
        <univocity-fixed>
            <univocity-header length="5"/>
            <univocity-header length="5"/>
            <univocity-header length="5"/>
        </univocity-fixed>
    </unmarshal>
    <to uri="mock:result"/>
</route>
```

第332章

-
-
-

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-univocity-parsers</artifactId>
  <version>x.x.x</version>
</dependency>
```

332.1.

332.2.

332.3.

-
-

332.3.1.

```
<route>
    <from uri="direct:input"/>
```

```
    <marshal>
        <univocity-csv/>
    </marshal>
    <to uri="mock:result"/>
</route>
```

332.3.2.

```
<route>
    <from uri="direct:input"/>
    <marshal>
        <univocity-fixed padding="_">
            <univocity-header length="5"/>
            <univocity-header length="5"/>
            <univocity-header length="5"/>
        </univocity-fixed>
    </marshal>
    <to uri="mock:result"/>
</route>
```

332.3.3.

```
<route>
    <from uri="direct:input"/>
    <marshal>
        <univocity-tsv/>
    </marshal>
    <to uri="mock:result"/>
</route>
```

332.4.

-
-

-
-

332.4.1.

```
<route>
    <from uri="direct:input"/>
    <unmarshal>
        <univocity-csv headerExtractionEnabled="true" asMap="true"/>
    </unmarshal>
    <to uri="mock:result"/>
</route>
```

332.4.2.

```
<route>
    <from uri="direct:input"/>
    <unmarshal>
        <univocity-fixed>
            <univocity-header length="5"/>
            <univocity-header length="5"/>
            <univocity-header length="5"/>
        </univocity-fixed>
    </unmarshal>
    <to uri="mock:result"/>
</route>
```

第333章

-
-
-

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-univocity-parsers</artifactId>
  <version>x.x.x</version>
</dependency>
```

333.1.

333.2.

333.3.

-
-

333.3.1.

```
<route>
    <from uri="direct:input"/>
    <marshal>
        <univocity-csv/>
```

```
    </marshal>
    <to uri="mock:result"/>
</route>
```

333.3.2.

```
<route>
    <from uri="direct:input"/>
    <marshal>
        <univocity-fixed padding="_">
            <univocity-header length="5"/>
            <univocity-header length="5"/>
            <univocity-header length="5"/>
        </univocity-fixed>
    </marshal>
    <to uri="mock:result"/>
</route>
```

333.3.3.

```
<route>
    <from uri="direct:input"/>
    <marshal>
        <univocity-tsv/>
    </marshal>
    <to uri="mock:result"/>
</route>
```

333.4.

-
-
-

-

333.4.1.

```
<route>
    <from uri="direct:input"/>
    <unmarshal>
        <univocity-csv headerExtractionEnabled="true" asMap="true"/>
    </unmarshal>
    <to uri="mock:result"/>
</route>
```

333.4.2.

```
<route>
    <from uri="direct:input"/>
    <unmarshal>
        <univocity-fixed>
            <univocity-header length="5"/>
            <univocity-header length="5"/>
            <univocity-header length="5"/>
        </univocity-fixed>
    </unmarshal>
    <to uri="mock:result"/>
</route>
```

第334章

-
-

334.1.

```
validator:someLocalOrRemoteResource
```

-
-
-
-

```
<dependency>
```

```
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-spring</artifactId>
    <version>x.x.x</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

334.2.


```
validator:resourceUri
```

334.2.1.

334.2.2.

334.3.

334.4.

第335章

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

335.1.

```
velocity:templateName[?options]
```

335.2.


```
velocity:resourceUri
```

335.2.1.

335.2.2.

335.3.


```
$in.setHeader("fruit", "Apple")
```

335.4.


```
VelocityContext velocityContext = new VelocityContext(variableMap);
exchange.getIn().setHeader("CamelVelocityContext", velocityContext);
```

335.5.

335.6.

335.7.

```
from("activemq:My.Queue").
  to("velocity:com/acme/MyResponse.vm");
```

```
from("activemq:My.Queue").
  to("velocity:com/acme/MyResponse.vm").
  to("activemq:Another.Queue");
```

```
from("activemq:My.Queue").
  to("velocity:com/acme/MyResponse.vm?contentCache=true").
  to("activemq:Another.Queue");
```

```
from("activemq:My.Queue").
  to("velocity:file://myfolder/MyResponse.vm?contentCache=true").
  to("activemq:Another.Queue");
```

```
from("direct:in").
  setHeader("CamelVelocityResourceUri").constant("path/to/my/template.vm").
  to("velocity:dummy");
```

```
from("direct:in").
  setHeader("CamelVelocityTemplate").constant("Hi this is a velocity template that can do
templating ${body}").
  to("velocity:dummy");
```

335.8.

> *Dear ${headers.lastName}, ${headers.firstName}*
>
> *Thanks for the order of ${headers.item}.*
>
> *Regards Camel Riders Bookstore*
> *${body}*

335.9.

-
-
-
-

第336章

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-vertx</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

336.1.

```
vertx:channelName[?options]
```

336.2.

> *vertx:address*

336.2.1.

336.2.2.

> *You can append query options to the URI in the following format, ?option=value&option=value&...*

336.3.

```
Vertx vertx = ...;
VertxComponent vertxComponent = new VertxComponent();
```

```
vertxComponent.setVertx(vertx);
camelContext.addComponent("vertx", vertxComponent);
```

336.4.

-
-
-
-

第337章

337.1.

```
vm:queueName[?options]
```

```
from("direct:foo").to("vm:bar?concurrentConsumers=5");

from("vm:bar?concurrentConsumers=5").to("file://output");
```

```
from("direct:foo").to("vm:bar");

from("vm:bar?concurrentConsumers=5").to("file://output");
```

337.2.

vm:name

337.2.1.

337.2.2.

337.3.

```
from("direct:in").bean(MyOrderBean.class).to("vm:order.email");
```

```
from("vm:order.email").bean(MyOrderEmailSender.class);
```

337.4.

-

第338章

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-weather</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

338.1.

```
weather://<unused name>[?options]
```

338.2.

338.3.

338.4.

> *weather:name*

338.4.1.

338.4.2.

338.5.

338.6.

338.7.

```
from("weather:foo?location=Madrid,Spain&period=7
days&appid=APIKEY&geolocationAccessKey=IPSTACK_ACCESS_KEY&geolocationRequest
HostIP=LOCAL_IP").to("jms:queue:weather");
```

```
from("weather:foo?
appid=APIKEY&geolocationAccessKey=IPSTACK_ACCESS_KEY&geolocationRequestHostIP
=LOCAL_IP").to("jms:queue:weather");
```

```
from("direct:start")
  .to("weather:foo?
location=Madrid,Spain&appid=APIKEY&geolocationAccessKey=IPSTACK_ACCESS_KEY&geo
locationRequestHostIP=LOCAL_IP");
```

```
  String json = template.requestBodyAndHeader("direct:start", "", "CamelWeatherLocation",
"Paris,France&appid=APIKEY", String.class);
```

```
  String json = template.requestBodyAndHeader("direct:start", "", "CamelWeatherLocation",
"current&appid=APIKEY", String.class);
```

第339章

339.1.

> **websocket**://hostname[:port][/resourceUri][?options]

339.2.

websocket:host:port/resourceUri

339.2.1.

339.2.2.

339.3.

339.4.

```
from("activemq:topic:newsTopic")
  .routeId("fromJMStoWebSocket")
  .to("websocket://localhost:8443/newsTopic?
sendToAll=true&staticResources=classpath:webapp");
```

339.5.

339.5.1.

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

TrustManagersParameters tmp = new TrustManagersParameters();
tmp.setKeyStore(ksp);

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);
scp.setTrustManagers(tmp);

CometdComponent commetdComponent = getContext().getComponent("cometds",
CometdComponent.class);
commetdComponent.setSslContextParameters(scp);
```

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
    <camel:keyManagers
       keyPassword="keyPassword">
      <camel:keyStore
         resource="/users/home/server/keystore.jks"
         password="keystorePassword"/>
    </camel:keyManagers>
    <camel:trustManagers>
      <camel:keyStore
         resource="/users/home/server/keystore.jks"
         password="keystorePassword"/>
    </camel:trustManagers>
  </camel:sslContextParameters>...
...
  <to uri="websocket://127.0.0.1:8443/test?sslContextParameters=#sslContextParameters"/>...
```

```
...
    protected RouteBuilder createRouteBuilder() throws Exception {
        return new RouteBuilder() {
            public void configure() {

                String uri = "websocket://127.0.0.1:8443/test?
sslContextParameters=#sslContextParameters";

                from(uri)
                    .log(">>> Message received from WebSocket Client : ${body}")
                    .to("mock:client")
                    .loop(10)
                        .setBody().constant(">> Welcome on board!")
                        .to(uri);
...
```

339.6.

-
-
-
-
-
-
-

第340章

340.1.

wordpress:operationDetail

340.1.1.

340.1.2.

> *wordpress:post?criteria.perPage=10&criteria.orderBy=author&criteria.categories=camel,dozer,json*

340.1.3.

```
public void configure() {
    final WordpressConfiguration configuration = new WordpressConfiguration();
    final WordpressComponentConfiguration component = new
WordpressComponentConfiguration();
    configuration.setApiVersion("2");
    configuration.setUrl("http://yoursite.com/wp-json/");
    component.setConfiguration(configuration);
    getContext().addComponent("wordpress", component);

    from("wordpress:post?id=1")
      .to("mock:result");
}
```

340.1.4.

-
-

340.1.5.

-
-
-

340.2.

第341章

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xchange</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

341.1.

```
xchange://exchange?options
```

341.2.

```
xchange:name
```

341.2.1.

341.2.2.

341.3.

```
##
# This file MUST NEVER be commited to source control.
# It is therefore added to .gitignore.
#
apiKey = GuRW0*********
secretKey = nKLki************
```

341.4.

<TODO><title></title>

```
from("direct:ticker").to("xchange:binance?
service=market&method=ticker&currencyPair=BTC/USDT")
```

</TODO>

第342章

```
from("activemq:My.Queue").
  unmarshal().xmlBeans().
  to("mqseries:Another.Queue");
```

342.1.

342.2.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xmlbeans</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第343章

-
-

343.1.

343.2.

343.2.1.

```
XmlJsonDataFormat xmlJsonFormat = new XmlJsonDataFormat();
```

```
XmlJsonDataFormat xmlJsonFormat = new XmlJsonDataFormat();
xmlJsonFormat.setEncoding("UTF-8");
xmlJsonFormat.setForceTopLevelObject(true);
xmlJsonFormat.setTrimSpaces(true);
xmlJsonFormat.setRootName("newRoot");
xmlJsonFormat.setSkipNamespaces(true);
xmlJsonFormat.setRemoveNamespacePrefixes(true);
xmlJsonFormat.setExpandableProperties(Arrays.asList("d", "e"));
```

```
// from XML to JSON
from("direct:marshal").marshal(xmlJsonFormat).to("mock:json");
// from JSON to XML
from("direct:unmarshal").unmarshal(xmlJsonFormat).to("mock:xml");
```

343.2.2.

```
// from XML to JSON - inline dataformat
from("direct:marshalInline").marshal().xmljson().to("mock:jsonInline");
// from JSON to XML - inline dataformat
from("direct:unmarshalInline").unmarshal().xmljson().to("mock:xmlInline");
```

```
Map<String, String> xmlJsonOptions = new HashMap<String, String>();
xmlJsonOptions.put(org.apache.camel.model.dataformat.XmlJsonDataFormat.ENCODING,
"UTF-8");
xmlJsonOptions.put(org.apache.camel.model.dataformat.XmlJsonDataFormat.ROOT_NAME,
"newRoot");
```

```
xmlJsonOptions.put(org.apache.camel.model.dataformat.XmlJsonDataFormat.SKIP_NAMESPACES, "true");
xmlJsonOptions.put(org.apache.camel.model.dataformat.XmlJsonDataFormat.REMOVE_NAMESPACE_PREFIXES, "true");
xmlJsonOptions.put(org.apache.camel.model.dataformat.XmlJsonDataFormat.EXPANDABLE_PROPERTIES, "d e");

// from XML to JSON - inline dataformat w/ options
from("direct:marshalInlineOptions").marshal().xmljson(xmlJsonOptions).to("mock:jsonInlineOptions");
// form JSON to XML - inline dataformat w/ options
from("direct:unmarshalInlineOptions").unmarshal().xmljson(xmlJsonOptions).to("mock:xmlInlineOptions");
```

343.3.

```
<dataFormats>
   <xmljson id="xmljson"/>
   <xmljson id="xmljsonWithOptions" forceTopLevelObject="true" trimSpaces="true"
rootName="newRoot" skipNamespaces="true"
         removeNamespacePrefixes="true" expandableProperties="d e"/>
</dataFormats>
```

```
<route>
   <from uri="direct:marshal"/>
   <marshal ref="xmljson"/>
   <to uri="mock:json" />
</route>

<route>
   <from uri="direct:unmarshalWithOptions"/>
   <unmarshal ref="xmljsonWithOptions"/>
   <to uri="mock:xmlWithOptions"/>
</route>
```

343.4.

```
{ "pref1:a": "value1", "pref2:b": "value2" }
```

```
XmlJsonDataFormat namespacesFormat = new XmlJsonDataFormat();
List<XmlJsonDataFormat.NamespacesPerElementMapping> namespaces = new
ArrayList<XmlJsonDataFormat.NamespacesPerElementMapping>();
namespaces.add(new XmlJsonDataFormat.
                NamespacesPerElementMapping("",
"|ns1|http://camel.apache.org/test1||http://camel.apache.org/default|"));
namespaces.add(new XmlJsonDataFormat.
                NamespacesPerElementMapping("surname",
"|ns2|http://camel.apache.org/personalData|" +
                    "ns3|http://camel.apache.org/personalData2|"));
namespacesFormat.setNamespaceMappings(namespaces);
namespacesFormat.setRootElement("person");
```

343.4.1.

```
{ "name": "Raul", "surname": "Kripalani", "f": true, "g": null}
```

```
<person xmlns="http://camel.apache.org/default" xmlns:ns1="http://camel.apache.org/test1">
   <f>true</f>
   <g null="true"/>
   <name>Raul</name>
   <surname xmlns:ns2="http://camel.apache.org/personalData"
xmlns:ns3="http://camel.apache.org/personalData2">Kripalani</surname>
</person>
```

343.5.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xmljson</artifactId>
  <version>x.x.x</version>
  <!-- Use the same version as camel-core, but remember that this component is only available
from 2.10 onwards -->
</dependency>

<!-- And also XOM must be included. XOM cannot be included by default due to an
incompatible
license with ASF; so add this manually -->
<dependency>
  <groupId>xom</groupId>
```

```
  <artifactId>xom</artifactId>
  <version>1.2.5</version>
</dependency>
```

343.6.

-
-

第344章

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xmlsecurity</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

344.1.

```
<[parent element]>
  ... <!-- Signature element is added as last child of the parent element-->
  <Signature Id="generated_unique_signature_id">
    <SignedInfo>
      <Reference URI="">
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        (<Transform>)* <!-- By default "http://www.w3.org/2006/12/xml-c14n11" is added to
the transforms -->
        <DigestMethod>
        <DigestValue>
      </Reference>
      (<Reference URI="#[keyinfo_Id]">
        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <DigestMethod>
        <DigestValue>
```

```
            </Reference>)?
            <!-- further references possible, see option 'properties' below -->
        </SignedInfo>
        <SignatureValue>
        (<KeyInfo Id="[keyinfo_id]">)?
        <!-- Object elements possible, see option 'properties' below -->
    </Signature>
</[parent element]>
```

```
<Signature Id="generated_unique_signature_id">
  <SignedInfo>
        <Reference URI="#generated_unique_object_id" type="[optional_type_value]">
            (<Transform>)* <!-- By default "http://www.w3.org/2006/12/xml-c14n11" is added to the
transforms -->
            <DigestMethod>
            <DigestValue>
        </Reference>
        (<Reference URI="#[keyinfo_id]">
            <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            <DigestMethod>
            <DigestValue>
        </Reference>)?
         <!-- further references possible, see option 'properties' below  -->
  </SignedInfo>
  <SignatureValue>
  (<KeyInfo Id="[keyinfo_id]">)?
  <Object Id="generated_unique_object_id"/> <!-- The Object element contains the in-message
body; the object ID can either be generated or set by the option parameter "contentObjectId" -
->
  <!-- Further Object elements possible, see option 'properties' below -->
</Signature>
```

```
(<[signed element] Id="[id_value]">
<!-- signed element must have an attribute of type ID -->
      ...

</[signed element]>
<other sibling/>*
<!-- between the signed element and the corresponding signature element, there can be other
siblings.
 Signature element is added as last sibling. -->
<Signature Id="generated_unique_ID">
   <SignedInfo>
      <CanonicalizationMethod>
      <SignatureMethod>
      <Reference URI="#[id_value]" type="[optional_type_value]">
      <!-- reference URI contains the ID attribute value of the signed element -->
```

```
        (<Transform>)* <!-- By default "http://www.w3.org/2006/12/xml-c14n11" is added to the
transforms -->
        <DigestMethod>
        <DigestValue>
    </Reference>
    (<Reference URI="#[generated_keyinfo_Id]">
        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <DigestMethod>
        <DigestValue>
    </Reference>)?
  </SignedInfo>
  <SignatureValue>
  (<KeyInfo Id="[generated_keyinfo_id]">)?
</Signature>)+
```

344.2.

```
xmlsecurity:sign:name[?options]
xmlsecurity:verify:name[?options]
```

-
-
-

344.3.

```
from("direct:enveloping").to("xmlsecurity:sign://enveloping?keyAccessor=#accessor",
                    "xmlsecurity:verify://enveloping?keySelector=#selector",
                    "mock:result")
```

```
<from uri="direct:enveloping" />
    <to uri="xmlsecurity:sign://enveloping?keyAccessor=#accessor" />
    <to uri="xmlsecurity:verify://enveloping?keySelector=#selector" />
<to uri="mock:result" />
```

344.4.

344.5.

```
xmlsecurity:command:name
```

344.5.1.

344.5.2.

344.5.3.

-
  -
  -
  -
-

```
<Signature>
    <SignedInfo>
      <Reference URI="#object"/>
      <!-- further references possible but they must not point to an Object or Manifest
containing an object reference -->
      ...
    </SignedInfo>

    <Object Id="object">
       <!-- contains one XML element which is extracted to the message body -->
    <Object>
    <!-- further object elements possible which are not referenced-->
    ...
    (<KeyInfo>)?
</Signature>
```

```
<Signature>
    <SignedInfo>
      <Reference URI="#manifest"/>
      <!-- further references  are possible but they must not point to an Object or other
manifest containing an object reference -->
      ...
    </SignedInfo>

    <Object >
      <Manifest Id="manifest">
        <Reference URI=#object/>
      </Manifest>
    </Objet>
    <Object Id="object">
      <!-- contains the DOM node which is extracted to the message body -->
    </Object>
     <!-- further object elements possible which are not referenced -->
    ...
    (<KeyInfo>)?
</Signature>
```

344.6.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
   <A ID="IDforA">
      <B>
         <C ID="IDforC">
            <D>dvalue</D>
         </C>
         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            Id="_6bf13099-0568-4d76-8649-faf5dcb313c0">
            <ds:SignedInfo>
               <ds:CanonicalizationMethod
                  Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
               <ds:SignatureMethod
                  Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
               <ds:Reference URI="#IDforC">
                  ...
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>aUDFmiG71</ds:SignatureValue>
         </ds:Signature>
      </B>
   </A>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"Id="_6b02fb8a-30df-42c6-ba25-
76eba02c8214">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
         <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
         <ds:Reference URI="#IDforA">
            ...
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>q3tvRoGgc8cMUqUSzP6C21zb7tt04riPnDuk=</ds:SignatureValue>
   </ds:Signature>
<root>
```

```
from("direct:detached")
  .to("xmlsecurity:sign://detached?
keyAccessor=#keyAccessorBeant&xpathsToIdAttributes=#xpathsToIdAttributesBean&schem
aResourceUri=Test.xsd")
  .to("xmlsecurity:verify://detached?
```

```
keySelector=#keySelectorBean&schemaResourceUri=org/apache/camel/component/xmlsecurity/Test.xsd")
  .to("mock:result");
```

```
<bean id="xpathsToIdAttributesBean" class="java.util.ArrayList">
    <constructor-arg type="java.util.Collection">
        <list>
            <bean
                class="org.apache.camel.component.xmlsecurity.api.XmlSignatureHelper"
                factory-method="getXpathFilter">
                <constructor-arg type="java.lang.String"
                    value="/ns:root/a/@ID" />
                <constructor-arg>
                    <map key-type="java.lang.String" value-type="java.lang.String">
                        <entry key="ns" value="http://test" />
                    </map>
                </constructor-arg>
            </bean>
        </list>
    </constructor-arg>
 </bean>
...
 <from uri="direct:detached" />
    <to
        uri="xmlsecurity:sign://detached?
keyAccessor=#keyAccessorBean&amp;xpathsToIdAttributes=#xpathsToIdAttributesBean&amp;schemaResourceUri=Test.xsd" />
    <to
        uri="xmlsecurity:verify://detached?
keySelector=#keySelectorBean&amp;schemaResourceUri=Test.xsd" />
    <to uri="mock:result" />
```

344.7.

```
<QualifyingProperties Target>
   <SignedProperties>
      <SignedSignatureProperties>
         (SigningTime)?
         (SigningCertificate)?
         (SignaturePolicyIdentifier)
         (SignatureProductionPlace)?
         (SignerRole)?
      </SignedSignatureProperties>
      <SignedDataObjectProperties>
         (DataObjectFormat)?
         (CommitmentTypeIndication)?
      </SignedDataObjectProperties>
   </SignedProperties>
</QualifyingProperties>
```

```
 Keystore keystore = ... // load a keystore
DefaultKeyAccessor accessor = new DefaultKeyAccessor();
accessor.setKeyStore(keystore);
accessor.setPassword("password");
accessor.setAlias("cert_alias"); // signer key alias

DefaultXAdESSignatureProperties props = new DefaultXAdESSignatureProperties();
props.setNamespace("http://uri.etsi.org/01903/v1.3.2#"); // sets the namespace for the XAdES
elements; the namspace is related to the XAdES version, default value is
"http://uri.etsi.org/01903/v1.3.2#", other possible values are "http://uri.etsi.org/01903/v1.1.1#"
and "http://uri.etsi.org/01903/v1.2.2#"
props.setPrefix("etsi"); // sets the prefix for the XAdES elements, default value is "etsi"

// signing certificate
props.setKeystore(keystore));
props.setAlias("cert_alias"); // specify the alias of the signing certificate in the keystore =
signer key alias
props.setDigestAlgorithmForSigningCertificate(DigestMethod.SHA256); // possible values for
the algorithm are "http://www.w3.org/2000/09/xmldsig#sha1",
"http://www.w3.org/2001/04/xmlenc#sha256", "http://www.w3.org/2001/04/xmldsig-
more#sha384", "http://www.w3.org/2001/04/xmlenc#sha512", default value is
"http://www.w3.org/2001/04/xmlenc#sha256"
props.setSigningCertificateURIs(Collections.singletonList("http://certuri"));

// signing time
props.setAddSigningTime(true);

// policy
```

```
props.setSignaturePolicy(XAdESSignatureProperties.SIG_POLICY_EXPLICIT_ID);
// also the values XAdESSignatureProperties.SIG_POLICY_NONE ("None"), and
XAdESSignatureProperties.SIG_POLICY_IMPLIED ("Implied")are possible, default value is
XAdESSignatureProperties.SIG_POLICY_EXPLICIT_ID ("ExplicitId")
// For "None" and "Implied" you must not specify any further policy parameters
props.setSigPolicyId("urn:oid:1.2.840.113549.1.9.16.6.1");
props.setSigPolicyIdQualifier("OIDAsURN"); //allowed values are empty string, "OIDAsURI",
"OIDAsURN"; default value is empty string
props.setSigPolicyIdDescription("invoice version 3.1");
props.setSignaturePolicyDigestAlgorithm(DigestMethod.SHA256);// possible values for the
algorithm are "http://www.w3.org/2000/09/xmldsig#sha1",
http://www.w3.org/2001/04/xmlenc#sha256", "http://www.w3.org/2001/04/xmldsig-
more#sha384", "http://www.w3.org/2001/04/xmlenc#sha512", default value is
http://www.w3.org/2001/04/xmlenc#sha256"
props.setSignaturePolicyDigestValue("Ohixl6upD6av8N7pEvDABhEL6hM=");
// you can add  qualifiers for the signature policy either by specifying text or an XML fragment
with the root element "SigPolicyQualifier"
props.setSigPolicyQualifiers(Arrays
    .asList(new String[] {
        "<SigPolicyQualifier xmlns=\"http://uri.etsi.org/01903/v1.3.2#\">
<SPURI>http://test.com/sig.policy.pdf</SPURI><SPUserNotice><ExplicitText>display
text</ExplicitText>"
            + "</SPUserNotice></SigPolicyQualifier>", "category B" }));
props.setSigPolicyIdDocumentationReferences(Arrays.asList(new String[]
{"http://test.com/policy.doc.ref1.txt",
    "http://test.com/policy.doc.ref2.txt" }));

// production place
props.setSignatureProductionPlaceCity("Munich");
props.setSignatureProductionPlaceCountryName("Germany");
props.setSignatureProductionPlacePostalCode("80331");
props.setSignatureProductionPlaceStateOrProvince("Bavaria");

//role
// you can add claimed roles either by specifying text or an XML fragment with the root
element "ClaimedRole"
props.setSignerClaimedRoles(Arrays.asList(new String[] {"test",
    "<a:ClaimedRole xmlns:a=\"http://uri.etsi.org/01903/v1.3.2#\">
<TestRole>TestRole</TestRole></a:ClaimedRole>" }));
props.setSignerCertifiedRoles(Collections.singletonList(new
XAdESEncapsulatedPKIData("AhixI6upD6av8N7pEvDABhEL6hM=",
    "http://uri.etsi.org/01903/v1.2.2#DER", "IdCertifiedRole")));

// data object format
props.setDataObjectFormatDescription("invoice");
props.setDataObjectFormatMimeType("text/xml");
props.setDataObjectFormatIdentifier("urn:oid:1.2.840.113549.1.9.16.6.2");
props.setDataObjectFormatIdentifierQualifier("OIDAsURN"); //allowed values are empty string,
"OIDAsURI", "OIDAsURN"; default value is empty string
props.setDataObjectFormatIdentifierDescription("identifier desc");
props.setDataObjectFormatIdentifierDocumentationReferences(Arrays.asList(new String[] {
    "http://test.com/dataobject.format.doc.ref1.txt",
"http://test.com/dataobject.format.doc.ref2.txt" }));

//commitment
props.setCommitmentTypeId("urn:oid:1.2.840.113549.1.9.16.6.4");
```

```
props.setCommitmentTypeIdQualifier("OIDAsURN"); //allowed values are empty string,
"OIDAsURI", "OIDAsURN"; default value is empty string
props.setCommitmentTypeIdDescription("description for commitment type ID");
props.setCommitmentTypeIdDocumentationReferences(Arrays.asList(new String[]
{"http://test.com/commitment.ref1.txt",
    "http://test.com/commitment.ref2.txt" }));
// you can specify a commitment type qualifier either by simple text or an XML fragment with
root element "CommitmentTypeQualifier"
props.setCommitmentTypeQualifiers(Arrays.asList(new String[] {"commitment qualifier",
    "<c:CommitmentTypeQualifier xmlns:c=\"http://uri.etsi.org/01903/v1.3.2#\"><C>c</C>
</c:CommitmentTypeQualifier>" }));

beanRegistry.bind("xmlSignatureProperties",props);
beanRegistry.bind("keyAccessorDefault",keyAccessor);

// you must reference the properties bean in the "xmlsecurity" URI
from("direct:xades").to("xmlsecurity:sign://xades?
keyAccessor=#keyAccessorDefault&properties=#xmlSignatureProperties")
          .to("mock:result");
```

```
...
<from uri="direct:xades" />
   <to
      uri="xmlsecurity:sign://xades?keyAccessor=#accessorRsa&amp;properties=#xadesProperties"
/>
   <to uri="mock:result" />
...
<bean id="xadesProperties"
   class="org.apache.camel.component.xmlsecurity.api.XAdESSignatureProperties">
   <!-- For more properties see the the previous Java DSL example.
       If you want to have a signing certificate then use the bean class
DefaultXAdESSignatureProperties (see the previous Java DSL example). -->
   <property name="signaturePolicy" value="ExplicitId" />
   <property name="sigPolicyId" value="http://www.test.com/policy.pdf" />
   <property name="sigPolicyIdDescription" value="factura" />
   <property name="signaturePolicyDigestAlgorithm"
value="http://www.w3.org/2000/09/xmldsig#sha1" />
   <property name="signaturePolicyDigestValue" value="Ohixl6upD6av8N7pEvDABhEL1hM=" />
   <property name="signerClaimedRoles" ref="signerClaimedRoles_XMLSigner" />
   <property name="dataObjectFormatDescription" value="Factura electrónica" />
   <property name="dataObjectFormatMimeType" value="text/xml" />
</bean>
<bean class="java.util.ArrayList" id="signerClaimedRoles_XMLSigner">
   <constructor-arg>
      <list>
         <value>Emisor</value>
         <value>&lt;ClaimedRole
            xmlns=&quot;http://uri.etsi.org/01903/v1.3.2#&quot;&gt;&lt;test
            xmlns=&quot;http://test.com/&quot;&gt;test&lt;/test&gt;&lt;/ClaimedRole&gt;</value>
      </list>
   </constructor-arg>
</bean>
```

–

344.7.1.

344.7.2.

-
-
-
-
-
-
-
-
-

-

344.8.

-

第345章

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xmpp</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

345.1.

```
xmpp:///[login@]hostname[:port][/participant][?Options]
```

345.2.

```
xmpp:host:port/participant
```

345.2.1.

345.2.2.

345.3.

345.4.

```
xmpp://superman@jabber.org/?room=krypton@conference.jabber.org&password=secret
```

```
xmpp://superman@jabber.org/joker@jabber.org?password=secret
```

```
from("timer://kickoff?period=10000").
setBody(constant("I will win!\n Your Superman.")).
to("xmpp://superman@jabber.org/joker@jabber.org?password=secret");
```

```
from("xmpp://superman@jabber.org/joker@jabber.org?password=secret").
to("activemq:evil.talk");
```

```
from("xmpp://superman@jabber.org/?
password=secret&room=krypton@conference.jabber.org").
to("activemq:krypton.talk");
```

```
from("xmpp://superman@jabber.org/?password=secret&room=krypton").
to("activemq:krypton.talk");
```

```
from("direct:start").
  to("xmpp://talk.google.com:5222/touser@gmail.com?
serviceName=gmail.com&user=fromuser&password=secret").
  to("mock:result");
```

345.5.

-
-
-
-

第346章

```
from("queue:foo").
  filter().xpath("//foo")).
  to("queue:bar")
```

```
from("queue:foo").
  choice().xpath("//foo")).to("queue:bar").
  otherwise().to("queue:others");
```

346.1.

346.2.

346.3.

-
-

346.3.1.

346.3.2.

-
-

-

346.4.

注意

346.5.

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

  <camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring"
xmlns:foo="http://example.com/person">
    <route>
      <from uri="activemq:MyQueue"/>
```

```
      <filter>
        <xpath>/foo:person[@name='James']</xpath>
        <to uri="mqseries:SomeOtherQueue"/>
      </filter>
    </route>
  </camelContext>
</beans>
```

346.6.

```
xpath("/foo:person/@id", String.class)
```

```
<xpath resultType="java.lang.String">/foo:person/@id</xpath>
```

```
@XPath(value = "concat('foo-',//order/name/)", resultType = String.class) String name)
```

346.7.

```
xpath("/invoice/@orderType = 'premium'", "invoiceDetails")
```

346.8.

346.9.

```
public class Foo {

    @MessageDriven(uri = "activemq:my.queue")
    public void doSomething(@MyXPath("/ns1:foo/ns2:bar/text()") String correlationID, @Body
String body) {
        // process the inbound message here
    }
}
```

346.10.

```
boolean matches = XPathBuilder.xpath("/foo/bar/@xyz").matches(context, "<foo><bar
xyz='cheese'/></foo>"));
```

```
String name = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>cheese</bar>
</foo>", String.class);
Integer number = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>123</bar>
</foo>", Integer.class);
Boolean bool = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>true</bar></foo>",
Boolean.class);
```

```
String name = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>cheese</bar>
</foo>");
```

346.11.

-
-

346.12.

```
XPathBuilder  INFO  Using system property
javax.xml.xpath.XPathFactory:http://saxon.sf.net/jaxp/xpath/om with value:
                net.sf.saxon.xpath.XPathFactoryImpl when creating XPathFactory
```

```
-Djavax.xml.xpath.XPathFactory=org.apache.xpath.jaxp.XPathFactoryImpl
```

346.13.

```
<xpath factoryRef="saxonFactory" resultType="java.lang.String">current-dateTime()</xpath>
```

```
<xpath objectModel="http://saxon.sf.net/jaxp/xpath/om" resultType="java.lang.String">current-dateTime()</xpath>
```

```
<xpath saxon="true" resultType="java.lang.String">current-dateTime()</xpath>
```

346.14.

```
[me: {prefix -> namespace}, {prefix -> namespace}], [parent: [me: {prefix -> namespace}, {prefix -> namespace}], [parent: [me: {prefix -> namespace}]]]
```

1.

2.

346.15.

-
-
-

```
XPathBuilder.xpath("/foo:person/@id", String.class).logNamespaces()
```

```
<xpath logNamespaces="true" resultType="String">/foo:person/@id</xpath>
```

```
2012-01-16 13:23:45,878 [stSaxonWithFlag] INFO  XPathBuilder  - Namespaces discovered in
message:
{xmlns:a=[http://apache.org/camel], DEFAULT=[http://apache.org/default],
xmlns:b=[http://apache.org/camelA, http://apache.org/camelB]}
```

346.16.

```
.setHeader("myHeader").xpath("resource:classpath:myxpath.txt", String.class)
```

346.17.

第347章

347.1.


```
xquery:resourceUri
```

347.1.1.

347.1.2.

347.2.

```
from("queue:foo").filter().
  xquery("//foo").
  to("queue:bar")
```

```
from("direct:start").
  recipientList().xquery("concat('mock:foo.', /person/@city)", String.class);
```

347.3.

347.4.

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:foo="http://example.com/person"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

  <camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring">
    <route>
```

```
      <from uri="activemq:MyQueue"/>
      <filter>
        <xquery>/foo:person[@name='James']</xquery>
        <to uri="mqseries:SomeOtherQueue"/>
      </filter>
    </route>
  </camelContext>
</beans>
```

```
    <xquery type="java.lang.String">concat('mock:foo.', /person/@city)</xquery>
```

347.5.

```
from("direct:start").
  transform().xquery("/people/person");
```

```
from("direct:start").
  transform().xquery("/people/person/text()", String.class);
```

347.6.

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="activemq:MyQueue"/>
    <to uri="xquery:com/acme/someTransform.xquery"/>
```

```
    <to uri="mqseries:SomeOtherQueue"/>
  </route>
</camelContext>
```

347.7.

347.8.

-
-

347.9.

```
.setHeader("myHeader").xquery("resource:classpath:myxquery.txt", String.class)
```

347.10.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-saxon</artifactId>
  <version>x.x.x</version>
</dependency>
```

第348章

348.1.

```
xslt:templateName[?options]
```

-
-

表348.1


```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring</artifactId>
```

```
    <version>x.x.x</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

348.2.


```
xslt:resourceUri
```

348.2.1.

348.2.2.

348.3.

```
from("activemq:My.Queue").
  to("xslt:com/acme/mytransform.xsl");
```

```
from("activemq:My.Queue").
  to("xslt:com/acme/mytransform.xsl").
  to("activemq:Another.Queue");
```

348.4.

```
<setHeader headerName="myParam"><constant>42</constant></setHeader>
<to uri="xslt:MyTransform.xsl"/>
```

```
<xsl: ..... >

  <xsl:param name="myParam"/>

   <xsl:template ...>
```

348.5.

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="activemq:My.Queue"/>
    <to uri="xslt:org/apache/camel/spring/processor/example.xsl"/>
    <to uri="activemq:Another.Queue"/>
  </route>
</camelContext>
```

348.6.

```
<xsl:include href="staff_template.xsl"/>
```

```
<xsl:include href="staff_template.xsl"/>
```

```
<xsl:include href="../staff_other_template.xsl"/>
```

348.7.

348.8.

```
SimpleRegistry registry = new SimpleRegistry();
registry.put("function1", new MyExtensionFunction1());
registry.put("function2", new MyExtensionFunction2());

CamelContext context = new DefaultCamelContext(registry);
context.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("direct:start")
         .to("xslt:org/apache/camel/component/xslt/extensions/extensions.xslt?
saxonExtensionFunctions=#function1,#function2");
   }
});
```

```
<bean id="function1" class="org.apache.camel.component.xslt.extensions.MyExtensionFunction1"/>
<bean id="function2" class="org.apache.camel.component.xslt.extensions.MyExtensionFunction2"/>

<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:extensions"/>
    <to uri="xslt:org/apache/camel/component/xslt/extensions/extensions.xslt?
saxonExtensionFunctions=#function1,#function2"/>
  </route>
</camelContext>
```

348.9.

348.10.

```
<xsl:template match="/">
  <html>
    <body>
      <xsl:for-each select="staff/programmer">
        <p>Name: <xsl:value-of select="name"/><br />
          <xsl:if test="dob=''">
            <xsl:message terminate="yes">Error: DOB is an empty string!</xsl:message>
          </xsl:if>
        </p>
      </xsl:for-each>
    </body>
  </html>
</xsl:template>
```

348.11.

1.

2.

348.12.

-

-

-

-

第349章

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xstream</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

349.1.

349.2.

```
// lets turn Object messages into XML then send to MQSeries
from("activemq:My.Queue").
  marshal().xstream().
  to("mqseries:Another.Queue");
```

```
XStream xStream = new XStream();
xStream.aliasField("money", PurchaseOrder.class, "cash");
// new Added setModel option since Camel 2.14
xStream.setModel("NO_REFERENCES");
...

from("direct:marshal").
  marshal(new XStreamDataFormat(xStream)).
  to("mock:marshaled");
```

349.3.

349.4.

```
from("activemq:My.Queue").
  marshal().xstream("UTF-8").
  to("mqseries:Another.Queue");
```

349.5.

```
<dataFormats>
   <xstream id="xstream-default"
         permissions="org.apache.camel.samples.xstream.*"/>
   ...
```

第350章

-

350.1.

警告

350.2.

-

```
from("activemq:My.Queue")
  .marshal().yaml()
  .to("mqseries:Another.Queue");
```

```
from("activemq:My.Queue")
  .marshal().yaml(YAMLLibrary.SnakeYAML)
  .to("mqseries:Another.Queue");
```

-

```
// Creat a SnakeYAMLDataFormat instance
SnakeYAMLDataFormat yaml = new SnakeYAMLDataFormat();

// Restrict classes to be loaded from YAML
yaml.addTypeFilters(TypeFilters.types(MyPojo.class, MyOtherPojo.class));

from("activemq:My.Queue")
  .unmarshal(yaml)
  .to("mqseries:Another.Queue");
```

350.3.

```
<dataFormats>
  <!--
    here we define a YAML data format with the id snake and that it should use
    the TestPojo as the class type when doing unmarshal. The unmarshalTypeName
    is optional
  -->
  <yaml
    id="snake"
    library="SnakeYAML"
```

```
    unmarshalTypeName="org.apache.camel.component.yaml.model.TestPojo"/>

  <!--
    here we define a YAML data format with the id snake-safe which restricts the
    classes to be loaded from YAML to TestPojo and those belonging to package
    com.mycompany
  -->
  <yaml id="snake-safe">
    <typeFilter value="org.apache.camel.component.yaml.model.TestPojo"/>
    <typeFilter value="com.mycompany\..*" type="regexp"/>
  </yaml>
</dataFormats>
```

```
  <route>
    <from uri="direct:unmarshal"/>
    <unmarshal>
      <custom ref="snake"/>
    </unmarshal>
    <to uri="mock:unmarshal"/>
  </route>
  <route>
    <from uri="direct:unmarshal-safe"/>
    <unmarshal>
      <custom ref="snake-safe"/>
    </unmarshal>
    <to uri="mock:unmarshal-safe"/>
  </route>
```

350.4.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-snakeyaml</artifactId>
  <version>${camel-version}</version>
</dependency>
```

第351章

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-yammer</artifactId>
  <version>${camel-version}</version>
</dependency>
```

351.1.

```
yammer:[function]?[options]
```

351.2.

351.3.

yammer:function

351.3.1.

351.3.2.

351.4.

yammer:messages?[options]	
yammer:my_feed?[options]	
yammer:algo?[options]	
yammer:following?[options]	

yammer:sent?[options]	
yammer:private?[options]	
yammer:received?[options]	

351.4.1.

```
from("yammer:messages?
consumerKey=aConsumerKey&consumerSecret=aConsumerSecretKey&accessToken=aAcce
ssToken")
    .to("mock:result");
```

```
{
   "messages":[
      {
         "replied_to_id":null,
         "network_id":7654,
         "url":"https://www.yammer.com/api/v1/messages/305298242",
         "thread_id":305298242,
         "id":305298242,
         "message_type":"update",
         "chat_client_sequence":null,
         "body":{
            "parsed":"Testing yammer API...",
            "plain":"Testing yammer API...",
            "rich":"Testing yammer API..."
         },
         "client_url":"https://www.yammer.com/",
         "content_excerpt":"Testing yammer API...",
         "created_at":"2013/06/25 18:14:45 +0000",
         "client_type":"Web",
         "privacy":"public",
         "sender_type":"user",
         "liked_by":{
            "count":1,
            "names":[
               {
```

```
                "permalink":"janstey",
                "full_name":"Jonathan Anstey",
                "user_id":1499642294
            }

        ]

    },
    "sender_id":1499642294,
    "language":null,
    "system_message":false,
    "attachments":[

    ],
    "direct_message":false,
    "web_url":"https://www.yammer.com/redhat.com/messages/305298242"
},
{
    "replied_to_id":null,
    "network_id":7654,
    "url":"https://www.yammer.com/api/v1/messages/294326302",
    "thread_id":294326302,
    "id":294326302,
    "message_type":"system",
    "chat_client_sequence":null,
    "body":{
        "parsed":"(Principal Software Engineer) has [[tag:14658]] the redhat.com network.
Take a moment to welcome Jonathan.",
        "plain":"(Principal Software Engineer) has #joined the redhat.com network. Take a
moment to welcome Jonathan.",
        "rich":"(Principal Software Engineer) has #joined the redhat.com network. Take a
moment to welcome Jonathan."
    },
    "client_url":"https://www.yammer.com/",
    "content_excerpt":"(Principal Software Engineer) has #joined the redhat.com network.
Take a moment to welcome Jonathan.",
    "created_at":"2013/05/10 19:08:29 +0000",
    "client_type":"Web",
    "sender_type":"user",
    "privacy":"public",
    "liked_by":{
        "count":0,
        "names":[

        ]

    }
}
]

}
```

```
Exchange exchange = mock.getExchanges().get(0);
Messages messages = exchange.getIn().getBody(Messages.class);

assertEquals(2, messages.getMessages().size());
assertEquals("Testing yammer API...", messages.getMessages().get(0).getBody().getPlain());
assertEquals("(Principal Software Engineer) has #joined the redhat.com network. Take a
moment to welcome Jonathan.", messages.getMessages().get(1).getBody().getPlain());
```

351.5.

```
yammer:messages?[options]
```

```
from("direct:start")
   .to("yammer:messages?
consumerKey=aConsumerKey&consumerSecret=aConsumerSecretKey&accessToken=aAcce
ssToken")
   .to("mock:result");
```

```
template.sendBody("direct:start", "Hi from Camel!");
```

```
Exchange exchange = mock.getExchanges().get(0);
Messages messages = exchange.getIn().getBody(Messages.class);

assertEquals(1, messages.getMessages().size());
assertEquals("Hi from Camel!", messages.getMessages().get(0).getBody().getPlain());
```

351.6.

yammer:relationships?[options]

351.7.

yammer:users?[options]	
yammer:current?[options]	

351.8.

```
from("direct:start")
    .pollEnrich("yammer:current?
consumerKey=aConsumerKey&consumerSecret=aConsumerSecretKey&accessToken=aAcce
ssToken")
    .to("mock:result");
```

351.9.

-
-

-
-

第352章

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-yql</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

352.1.

```
yql://query[?options]
```

352.2.


```
yql:query
```

352.2.1.

352.2.2.

352.3.

352.4.

352.5.

352.5.1.

```
from("direct:start")
  .to("yql://select wind, atmosphere from weather.forecast where woeid in (select woeid from
geo.places(1) where text='chicago, il'");
```

```
{
  "query":{
    "count":1,
    "created":"2017-11-01T19:37:26Z",
    "lang":"en-US",
    "results":{
      "channel":{
        "wind":{
          "chill":"32",
          "direction":"165",
          "speed":"22"
        },
        "atmosphere":{
          "humidity":"71",
          "pressure":"994.0",
          "rising":"0",
          "visibility":"16.1"
        }
      }
    }
  }
}
```


352.5.2.

```
from("direct:start")
  .to("yql://select symbol, Ask, Bid, AverageDailyVolume from yahoo.finance.quotes where
symbol in ('GOOG')?
```

```
env=store://datatables.org/alltableswithkeys&https=false&callback=yqlCallback");
```

```
/**/yqlCallback({
   "query":{
      "count":1,
      "created":"2017-11-01T19:48:17Z",
      "lang":"en-US",
      "results":{
         "quote":{
            "symbol":"GOOG",
            "Bid":"1025.57",
            "Ask":"1025.92",
            "AverageDailyVolume":"1350640"AverageDailyVolume
         }
      }
   }
});
```


352.5.3.

```
from("direct:start")
  .to("yql://select * from google.books where q='barack obama' and maxResults=1?
format=xml&crossProduct=optimized&env=store://datatables.org/alltableswithkeys");
```

```
<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:yahoo="http://www.yahooapis.com/v1/base.rng" yahoo:count="1"
yahoo:created="2017-11-01T20:32:22Z" yahoo:lang="en-US">
   <results>
      <json>
         <kind>books#volumes</kind>
         <totalItems>1993</totalItems>
         <items>
```

```
        <kind>books#volume</kind>
        <id>HRCHJp-V0QUC</id>
        <etag>SeTJeSgFDzo</etag>
        <selfLink>https://www.googleapis.com/books/v1/volumes/HRCHJp-V0QUC</selfLink>
        <volumeInfo>
          <title>Dreams from My Father</title>
          <subtitle>A Story of Race and Inheritance</subtitle>
          <authors>Barack Obama</authors>
          <publisher>Broadway Books</publisher>
          <publishedDate>2007-01-09</publishedDate>
          ...
        </volumeInfo>
      </items>
    </json>
  </results>
</query>
<!-- total: 646 -->
```


352.6.

-

第353章

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-zendesk</artifactId>
  <version>${camel-version}</version>
</dependency>
```

353.1.


```
zendesk:methodName
```

353.1.1.

353.1.2.

353.2.

```
zendesk://endpoint?[options]
```

353.3.

353.4.

353.5.

353.6.

第354章

354.1.

354.2.

```
from("direct:start").marshal().zip(Deflater.BEST_COMPRESSION).to("activemq:queue:MY_QUEUE");
```

```
from("direct:start").marshal().zip().to("activemq:queue:MY_QUEUE");
```

354.3.

```
from("activemq:queue:MY_QUEUE").unmarshal().zip().process(new UnZippedMessageProcessor());
```

354.4.

第355章

355.1.

355.2.

```
from("direct:start")
    .marshal().zipFile()
    .to("activemq:queue:MY_QUEUE");
```

```
from("file:input/directory?antInclude=*/.txt")
    .marshal().zipFile()
    .to("file:output/directory");
```

```
from("direct:start")
    .setHeader(Exchange.FILE_NAME, constant("report.txt"))
    .marshal().zipFile()
    .to("file:output/directory");
```

355.3.

```
from("activemq:queue:MY_QUEUE")
   .unmarshal().zipFile()
   .process(new UnZippedMessageProcessor());
```

```
ZipFileDataFormat zipFile = new ZipFileDataFormat();
zipFile.setUsingIterator(true);

from("file:src/test/resources/org/apache/camel/dataformat/zipfile/?
consumer.delay=1000&noop=true")
   .unmarshal(zipFile)
   .split(body(Iterator.class)).streaming()
      .process(new UnZippedMessageProcessor())
   .end();
```

```
from("file:src/test/resources/org/apache/camel/dataformat/zipfile?
consumer.delay=1000&noop=true")
   .split(new ZipSplitter()).streaming()
      .process(new UnZippedMessageProcessor())
   .end();
```

355.4.

注記

```
from("file:input/directory?antInclude=*/.txt")
   .aggregate(constant(true), new ZipAggregationStrategy())
```

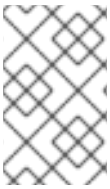
```
        .completionFromBatchConsumer().eagerCheckCompletion()
        .to("file:output/directory");
```

```
from("file:input/directory?antInclude=*/.txt")
    .aggregate(constant(true), new ZipAggregationStrategy())
        .completionFromBatchConsumer().eagerCheckCompletion()
        .setHeader(Exchange.FILE_NAME, constant("reports.zip"))
        .to("file:output/directory");
```

355.5.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-zipfile</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

第356章



注記

-
-
-

-
-
-

356.1.

356.2.

```
ZipkinTracer zipkin = new ZipkinTracer();
// Configure a reporter, which controls how often spans are sent
//   (the dependency is io.zipkin.reporter2:zipkin-sender-okhttp3)
sender = OkHttpSender.create("http://127.0.0.1:9411/api/v2/spans");
```

```
zipkin.setSpanReporter(AsyncReporter.create(sender));
// and then add zipkin to the CamelContext
zipkin.init(camelContext);
```

```
<!-- configure how to reporter spans to a Zipkin collector
       (the dependency is io.zipkin.reporter2:zipkin-reporter-spring-beans) -->
<bean id="http" class="zipkin2.reporter.beans.AsyncReporterFactoryBean">
  <property name="sender">
    <bean id="sender" class="zipkin2.reporter.beans.OkHttpSenderFactoryBean">
      <property name="endpoint" value="http://localhost:9411/api/v2/spans"/>
    </bean>
  </property>
  <!-- wait up to half a second for any in-flight spans on close -->
  <property name="closeTimeout" value="500"/>
</bean>

<!-- setup zipkin tracer -->
<bean id="zipkinTracer" class="org.apache.camel.zipkin.ZipkinTracer">
  <property name="serviceName" value="dude"/>
  <property name="spanReporter" ref="http"/>
</bean>
```

356.2.1.

-

```
zipkin.setServiceName("invoices");
```

356.2.2.

-
-

```
from("activemq:queue:inbox")
  .to("http:someserver/somepath");
```

```
zipkin.addServerServiceMapping("activemq:queue:inbox", "orders");
```

```
zipkin.addClientServiceMapping("http:someserver/somepath", "audit");
```

```
zipkin.addClientServiceMapping("http:someserver*", "audit");
```

356.3.

1.
2.

3.

4.

5.

1.

2.

3.

4.

5.

356.3.1.

```
zipkin.addServerServiceMapping("activemq:queue:inbox", "activemq:queue:inbox");
zipkin.addClientServiceMapping("http:someserver/somepath", "http:someserver/somepath");
```

356.4.

第357章

1.

2.

3.

4.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-zookeeper</artifactId>
  <version>x.x.x</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

357.1.

```
zookeeper://zookeeper-server[:port][/path][?options]
```

357.2.

zookeeper:serverUrls/path

357.2.1.

357.2.2.

357.3.

357.3.1.

```
from("zookeeper://localhost:39913/somepath/somenode").to("mock:result");
```

```
from("zookeeper://localhost:39913/somepath/somenode?
awaitCreation=true").to("mock:result");
```

357.3.2.

357.3.3.

```
from("direct:write-to-znode")
    .to("zookeeper://localhost:39913/somepath/somenode");
```

警告

```
Object testPayload = ...
template.sendBodyAndHeader("direct:write-to-znode", testPayload, "CamelZooKeeperNode",
"/somepath/someothernode");
```

```
from("direct:create-and-write-to-znode")
   .to("zookeeper://localhost:39913/somepath/somenode?create=true");
```

```
from("direct:delete-znode")
   .setHeader(ZooKeeperMessage.ZOOKEEPER_OPERATION, constant("DELETE"))
   .to("zookeeper://localhost:39913/somepath/somenode");
```

```
<route>
  <from uri="direct:delete-znode" />
  <setHeader headerName="CamelZookeeperOperation">
    <constant>DELETE</constant>
  </setHeader>
  <to uri="zookeeper://localhost:39913/somepath/somenode" />
</route>
```

-
-
-
-

```
from("direct:create-and-write-to-persistent-znode")
   .to("zookeeper://localhost:39913/somepath/somenode?
create=true&createMode=PERSISTENT");
```

警告

```
Object testPayload = ...
template.sendBodyAndHeader("direct:create-and-write-to-persistent-znode", testPayload,
"CamelZooKeeperCreateMode", "PERSISTENT");
```

357.4.

```
ZooKeeperRoutePolicy policy = new
ZooKeeperRoutePolicy("zookeeper:localhost:39913/someapp/somepolicy", 1);
from("direct:policy-controlled")
    .routePolicy(policy)
    .to("mock:controlled");
```

-
-
-

357.5.

-
-
-
-

第358章

358.1.

```
from("zookeeper-master:cheese:jms:foo").to("activemq:wine");
```

358.2.

```
zookeeper-master:name:endpoint[?options]
```

358.3.

zookeeper-master:groupName:consumerEndpointUri

358.3.1.

358.3.2.

358.4.

```
// the file endpoint we want to consume from
String url = "file:target/inbox?delete=true";

// use the zookeeper master component in the clustered group named myGroup
// to run a master/slave mode in the following Camel url
from("zookeeper-master:myGroup:" + url)
    .log(name + " - Received file: ${file:name}")
```

```
        .delay(delay)
        .log(name + " - Done file:     ${file:name}")
        .to("file:target/outbox");
```

```
    MasterComponent master = new MasterComponent();
    master.setZooKeeperUrl("myzookeeper:2181");
```

```
export ZOOKEEPER_URL = "myzookeeper:2181"
```

第359章

-
-
-

```
MasterRoutePolicy master = new MasterRoutePolicy();
master.setZooKeeperUrl("localhost:2181");
master.setGroupName("myGroup");

// its import to set the route to not auto startup
// as we let the route policy start/stop the routes when it becomes a master/slave etc
from("file:target/inbox?delete=true").noAutoStartup()
    // use the zookeeper master route policy in the clustered group
    // to run this route in master/slave mode
    .routePolicy(master)
    .log(name + " - Received file: ${file:name}")
    .delay(delay)
    .log(name + " - Done file:     ${file:name}")
    .to("file:target/outbox");
```

359.1.

-
-
-

-