Red Hat build of Apache Camel for Spring Boot 3.20

Camel Spring Boot 入门

Camel Spring Boot 入门

Last Updated: 2024-06-14

Red Hat build of Apache Camel for Spring Boot 3.20 Camel Spring Boot 入门

Camel Spring Boot 入门

法律通告

摘要

本指南介绍了红帽构建的 Spring Boot 的 Apache Camel，并解释了使用红帽构建的 Apache Camel for Spring Boot 创建和部署应用程序的各种方法。

目录

前言

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 CAMEL SPRING BOOT 入门

本指南介绍了 Camel Spring Boot，并演示了如何开始使用 Camel Spring Boot 构建应用程序：

- 第 1.1 节 “Camel Spring Boot 启动器”
- 第 1.2 节 “Spring Boot”
- 第 1.3 节 “组件启动器”
- 第 1.4 节 “Starter 配置”
- 第 1.5 节 “使用 Maven 为 Spring Boot 应用程序生成 Camel”
- 第 1.7 节 “将补丁应用到 Camel Spring Boot”
- 第 1.8 节 “Camel REST DSL OpenApi Maven 插件”
- 第 1.9 节 “支持 FIPS 合规性”

1.1. CAMEL SPRING BOOT 启动器

Camel 支持 Spring Boot 为许多 Camel 组件提供 Camel 和启动程序的 自动配置。Camel 上下文会自动探测到 Spring 上下文中的 Camel 路由的不透明自动配置，并将密钥 Camel 实用程序（如制作者模板、消费者模板和类型转换器）注册为 Bean。

注意

有关使用 Maven archtype 为 Spring Boot 应用程序生成 Camel 的详情，请参考使用 Maven 为 Spring Boot 应用程序生成 Camel。

要开始，您必须将 Camel Spring Boot BOM 添加到 Maven **pom.xml** 文件中。

```
<dependencyManagement>

  <dependencies>
    <!-- Camel BOM -->
    <dependency>
      <groupId>com.redhat.camel.springboot.platform</groupId>
      <artifactId>camel-spring-boot-bom</artifactId>
      <version>3.20.1.redhat-00104</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <!-- ... other BOMs or dependencies ... -->
  </dependencies>

</dependencyManagement>
```

camel-spring-boot-bom 是一个基本 BOM，其中包含 Camel Spring Boot starter JARs 列表。

接下来，添加 Camel Spring Boot Start 来启动 Camel 上下文。

```
  <dependencies>
```

```
    <!-- Camel Starter -->
    <dependency>
      <groupId>com.redhat.camel.springboot.platform</groupId>
      <artifactId>camel-spring-boot-starter</artifactId>
    </dependency>
    <!-- ... other dependencies ... -->
  </dependencies>
```

您还必须添加 Spring Boot 应用程序所需的任何组件启动程序。以下示例演示了如何将自动配置 入门 程序添加到 MQTT5 组件

```
<dependencies>
  <!-- ... other dependencies ... -->
  <dependency>
    <groupId>org.apache.camel.springboot</groupId>
    <artifactId>camel-paho-mqtt5</artifactId>
  </dependency>
</dependencies>
```

1.1.1. Camel Spring Boot BOM 与 Camel Spring Boot Dependencies BOM

策展的 **camel-spring-boot-dependencies** BOM （生成）包含 Spring Boot 和 Apache Camel 使用的调整 JAR，以避免任何冲突。此 BOM 用于测试 camel-spring-boot 本身。

Spring Boot 用户可以使用 **camel-spring-boot-bom** 选择使用 *纯* Camel 依赖项，该依赖项只有 Camel starter JARs 作为受管依赖项。但是，如果 Spring Boot 的第三方 JAR 与特定的 Camel 组件不兼容，这可能会导致类路径冲突。

1.1.2. Spring Boot 配置支持

每个 入门 程序列出了您可以在标准 **application.properties** 或 **application.yml** 文件中配置的配置参数。这些参数的格式为 **camel.component.[component-name].[parameter]**。例如，要配置 mq5 代理的 URL，您可以设置：

```
camel.component.paho-mqtt5.broker-url=tcp://localhost:61616
```

1.1.3. 添加 Camel 路由

Camel 路由 在 Spring 应用程序上下文中检测到，例如使用 **org.springframework.stereotype.Component** 注解的路由会被加载，添加到 Camel 上下文并运行。

```
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyRoute extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("...")
      .to("...");
```

```
    }

}
```

1.2. SPRING BOOT

Spring Boot 可为您自动配置 Camel。Camel 上下文会自动探测到 Spring 上下文中的 Camel 路由的不透明自动配置，并将密钥 Camel 实用程序（如制作者模板、消费者模板和类型转换器）注册为 Bean。

Maven 用户需要将以下依赖项添加到其 **pom.xml** 中，才能使用此组件：

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-spring-boot</artifactId>
  <version>3.20.1.redhat-00104</version> <!-- use the same version as your Camel core version -->
</dependency>
```

camel-spring-boot jar 附带 **spring.factories** 文件，因此当您将该依赖项添加到类路径后，Spring Boot 将自动为您自动配置 Camel。

1.2.1. Camel Spring Boot Starter

Apache Camel 提供了一个 Spring Boot Starter 模块，它允许您使用启动程序开发 Spring Boot 应用程序。源代码中也有一个 示例应用程序。

要使用启动程序，将以下内容添加到 spring boot pom.xml 文件中：

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-spring-boot-bom</artifactId>
  <version>3.20.1.redhat-00104</version> <!-- use the same version as your Camel core version -->
</dependency>
```

然后，您只能在 Camel 路由中添加类，例如：

```
package com.example;

import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyRoute extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("timer:foo").to("log:bar");
  }
}
```

然后，这些路由将自动启动。

您可以在 **application.properties** 或 **application.yml** 文件中自定义 Camel 应用程序。

1.2.2. Spring Boot 自动配置

当在 Spring Boot 中使用 spring-boot 时，请确保使用以下 Maven 依赖项来支持自动配置：

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-spring-boot-starter</artifactId>
  <version>3.20.1.redhat-00104</version> <!-- use the same version as your Camel core version -->
</dependency>
```

1.2.3. 自动配置的 Camel 上下文

Camel 自动配置提供的最重要功能是 **CamelContext** 实例。Camel 自动配置为您创建一个 **Spring CamelContext**，并负责该上下文的正确初始化和关闭。创建的 Camel 上下文也在 Spring 应用程序上下文中注册（在 **camelContext** bean 名称下），以便您可以像任何其他 Spring bean 一样访问它。

```
@Configuration
public class MyAppConfig {

  @Autowired
  CamelContext camelContext;

  @Bean
  MyService myService() {
    return new DefaultMyService(camelContext);
  }

}
```

1.2.4. 自动检测 Camel 路由

Camel 自动配置从 Spring 上下文收集所有 **RouteBuilder** 实例，并自动将它们注入提供的 **CamelContext**。这意味着，使用 Spring Boot starter 创建新的 Camel 路由非常简单，就像将 **@Component** 注解类添加到您的 classpath 中一样简单：

```
@Component
public class MyRouter extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("jms:invoices").to("file:/invoices");
  }

}
```

或者在 **@Configuration** 类中创建新路由 **RouteBuilder** bean：

```
@Configuration
public class MyRouterConfiguration {

  @Bean
  RoutesBuilder myRouter() {
    return new RouteBuilder() {
```

```
    @Override
    public void configure() throws Exception {
      from("jms:invoices").to("file:/invoices");
    }

  };
 }

}
```

1.2.5. Camel 属性

Spring Boot 自动配置自动连接到 Spring Boot 外部配置 （可能包含带有 Camel 属性支持的属性占位符、OS 环境变量或系统属性）。它基本上意味着 **application.properties** 文件中定义的任何属性：

```
route.from = jms:invoices
```

或者通过系统属性设置：

```
java -Droute.to=jms:processed.invoices -jar mySpringApp.jar
```

可用作 Camel 路由中的占位符：

```
@Component
public class MyRouter extends RouteBuilder {

 @Override
 public void configure() throws Exception {
  from("{{route.from}}").to("{{route.to}}");
 }

}
```

1.2.6. 自定义 Camel 上下文配置

如果要对 Camel 自动配置创建的 **CamelContext** bean 执行一些操作，请在 Spring 上下文中注册 **CamelContextConfiguration** 实例：

```
@Configuration
public class MyAppConfig {

 @Bean
 CamelContextConfiguration contextConfiguration() {
  return new CamelContextConfiguration() {
   @Override
   void beforeApplicationStart(CamelContext context) {
    // your custom configuration goes here
   }
  };
 }

}
```

在 Spring 上下文启动前才会调用 **ApplicationStart** 的方法，因此传递到此回调的 **CamelContext** 实例完全自动配置。如果您将多个 **CamelContextConfiguration** 实例添加到 Spring 上下文中，则会执行每个实例。

1.2.7. 自动配置的消费者和制作者模板

Camel auto-configuration 提供预配置的 **ConsumerTemplate** 和 **ProducerTemplate** 实例。只需将它们注入到 Spring 管理的 Bean 中：

```
@Component
public class InvoiceProcessor {

  @Autowired
  private ProducerTemplate producerTemplate;

  @Autowired
  private ConsumerTemplate consumerTemplate;

  public void processNextInvoice() {
    Invoice invoice = consumerTemplate.receiveBody("jms:invoices", Invoice.class);
    ...
    producerTemplate.sendBody("netty-http:http://invoicing.com/received/" + invoice.id());
  }

}
```

默认情况下，消费者模板和制作者模板将端点缓存大小设置为 1000。您可以通过修改以下 Spring 属性来更改这些值：

```
camel.springboot.consumer-template-cache-size = 100
camel.springboot.producer-template-cache-size = 200
```

1.2.8. 自动配置的 TypeConverter

Camel auto-configuration 在 Spring 上下文中注册名为 **typeConverter** 的 **TypeConverter** 实例。

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

  public long parseInvoiceValue(Invoice invoice) {
    String invoiceValue = invoice.grossValue();
    return typeConverter.convertTo(Long.class, invoiceValue);
  }

}
```

1.2.8.1. Spring 类型转换 API 网桥

Spring 附带强大的 类型转换 API。Spring API 与 Camel 类型转换器 API 类似。就像两个 API 都是相似的，Camel Spring Boot 会自动注册到 Spring 转换 API 的桥接转换器(**SpringTypeConverter**)。这意味着

开箱即用的 Camel 将像 Camel 一样对待 Spring Converters。使用这个方法，您可以使用通过 Camel **TypeConverter** API 访问的 Camel 和 Spring 转换器：

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

  public UUID parseInvoiceId(Invoice invoice) {
    // Using Spring's StringToUUIDConverter
    UUID id = invoice.typeConverter.convertTo(UUID.class, invoice.getId());
  }

}
```

在 hood Camel Spring Boot 下，将转换委派给应用程序上下文中可用的 Spring **ConversionService** 实例。如果没有 **ConversionService** 实例，Camel Spring Boot 自动配置将为您创建一个。

1.2.9. 使应用程序保持处于活动状态

在启动时启用此功能的 Camel 应用程序，以便防止 JVM 终止使应用程序保持处于活动状态的唯一目的。这意味着，在使用 Spring Boot 启动 Camel 应用程序后，您的应用程序会等待 **Ctrl+C** 信号，且不会立即退出。

可以使用 **camel.springboot.main-run-controller** 激活控制器线程为 **true**。

```
camel.springboot.main-run-controller = true
```

使用 Web 模块的应用程序（例如，导入 **org.springframework.boot:spring-boot-web-starter** 模块的应用程序）通常不需要使用此功能，因为应用程序会存在其他非守护进程线程。

1.2.10. 添加 XML 路由

默认情况下，您可以将 Camel XML 路由放在目录 camel 下的 classpath 中，它 camel-spring-boot 将自动探测并包含它。您可以配置目录名称，或使用配置选项关闭这个目录：

```
# turn off
camel.springboot.routes-include-pattern = false
```

```
# scan only in the com/foo/routes classpath
camel.springboot.routes-include-pattern = classpath:com/foo/routes/*.xml
```

XML 文件应该是 Camel XML 路由(**而非 < CamelContext&gt**;)，例如：

```
<routes xmlns="http://camel.apache.org/schema/spring">
   <route id="test">
      <from uri="timer://trigger"/>
      <transform>
         <simple>ref:myBean</simple>
      </transform>
```

```
        <to uri="log:out"/>
    </route>
</routes>
```

1.2.11. 测试 JUnit 5 方法

为了进行测试，Maven 用户需要将以下依赖项添加到其 **pom.xml** 中：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <version>2.7.18</version> <!-- Use the same version as your Spring Boot version -->
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-test-spring-junit5</artifactId>
    <version>3.20.1.redhat-00050</version> <!-- use the same version as your Camel core version -->
    <scope>test</scope>
</dependency>
```

要测试 Camel Spring Boot 应用程序，请使用 **@CamelSpringBootTest** 注解您的测试类。这为应用程序提供了 Camel 的 Spring Test 支持，以便您可以使用 Spring Boot 测试惯例编写测试。

若要获取 **CamelContext** 或 **ProducerTemplate**，您可以使用 **@Autowired** 以普通 Spring 方式将它们注入类。

您还可以使用 camel-test-spring-junit5 来配置测试。本例使用 **@MockEndpoints** 注释自动-mock a 端点：

```
@CamelSpringBootTest
@SpringBootApplication
@MockEndpoints("direct:end")
public class MyApplicationTest {

    @Autowired
    private ProducerTemplate template;

    @EndpointInject("mock:direct:end")
    private MockEndpoint mock;

    @Test
    public void testReceive() throws Exception {
        mock.expectedBodiesReceived("Hello");
        template.sendBody("direct:start", "Hello");
        mock.assertIsSatisfied();
    }

}
```

1.3. 组件启动器

Camel Spring Boot 支持以下 Camel 工件作为 Spring Boot Starters：

- 表 1.1 “Camel 组件”
- 表 1.2 “Camel 数据格式”
- 表 1.3 “Camel Languages”
- 表 1.4 “其它扩展”

注意

以下列出的一些工件还不提供参考文档。当本文档可用后立即发布。

表 1.1. Camel 组件

组件	工件	描述
AMQP	camel-amqp-starter	使用 Apache QPid 客户端使用 AMQP 协议进行消息传递.
AWS Cloudwatch	camel-aws2-cw-starter	使用 AWS SDK 版本 2.x 将指标发送到 AWS CloudWatch。
AWS DynamoDB	camel-aws2-ddb-starter	使用 AWS SDK 版本 2.x 从 AWS DynamoDB 服务存储和检索数据。
AWS Kinesis	camel-aws2-kinesis-starter	使用 AWS SDK 版本 2.x 从 AWS Kinesis Streams 使用和生成记录。
AWS Lambda	camel-aws2-lambda-starter	使用 AWS SDK 版本 2.x 管理并调用 AWS Lambda 功能。
AWS S3 Storage Service	camel-aws2-s3-starter	使用 AWS SDK 版本 2.x 从 AWS S3 Storage Service 存储和检索对象。
AWS Simple Notification System (SNS)	camel-aws2-sns-starter	使用 AWS SDK 版本 2.x 将信息发送到 AWS Simple Notification 主题。
AWS Simple Queue Service (SQS)	camel-aws2-sqs-starter	使用 AWS SDK 版本 2.x 向 AWS SQS 服务发送和接收信息。
Azure ServiceBus	camel-azure-servicebus-starter	向 Azure 事件总线发送和接收信息。
Azure Storage Blob Service	camel-azure-storage-blob-starter	使用 SDK v12 从 Azure Storage Blob Service 存储和检索 Blob。

组件	工件	描述
Azure Storage Queue Service	camel-azure-storage-queue-starter	azure-storage-queue 组件用于使用 Azure SDK v12 存储和检索信息到 Azure Storage Queue。
bean	camel-bean-starter	调用存储在 Camel registry 中的 Java Bean 的方法。
Bean Validator	camel-bean-validator-starter	使用 Java Bean Validation API 验证消息正文。
浏览	camel-browse-starter	检查在支持 BrowsableEndpoint 的端点上收到的消息。
Cassandra CQL	camel-cassandraql-starter	使用 CQL3 API （而不是 Thrift API）与 Cassandra 2.0 集成。基于 DataStax 提供的 Cassandra Java 驱动程序。
控制总线	camel-controlbus-starter	管理和监控 Camel 路由。
Cron	camel-cron-starter	通过 Unix cron 语法指定的时间触发事件的通用接口。
CXF	camel-cxf-soap-starter	使用 Apache CXF 公开 SOAP WebServices，或使用 CXF WS 客户端连接到外部 WebServices。
数据格式	camel-dataformat-starter	使用 Camel 数据格式作为常规 Camel 组件。
dataset	camel-dataset-starter	提供 Camel 应用程序的负载和 soak 测试数据。
direct	camel-direct-starter	同步调用来自同一 Camel 上下文的另一个端点。
Elastic Search	camel-elasticsearch-starter	通过 Java 客户端 API 将请求发送到 ElasticSearch。
FHIR	camel-fhir-starter	使用 FHIR (Fast Healthcare Interoperability Resources)标准在电信域中交换信息。
File	camel-file-starter	读取和写入文件。
FTP	camel-ftp-starter	将文件上传到 FTP 服务器。

组件	工件	描述
Google BigQuery	camel-google-bigquery-starter	Google BigQuery 数据存储用于分析。
Google Pubsub	camel-google-pubsub-starter	向 Google Cloud Platform PubSub Service 发送和接收消息。
HTTP	camel-http-starter	使用 Apache HTTP 客户端 4.x 向外部 HTTP 服务器发送请求。
Infinispan	camel-infinispan-starter	从/写入 Infinispan 分布式键/值存储和数据网格。
JIRA	camel-jira-starter	与 JIRA 问题跟踪器交互。
JMS	camel-jms-starter	发送和接收来自 JMS Queue 或 Topic 的信息。
JPA	camel-jpa-starter	使用 Java Persistence API (DSL) 从数据库存储和检索 Java 对象。
JSLT	camel-jslt-starter	使用 JSLT 查询或转换 JSON 有效负载。
Kafka	camel-kafka-starter	发送和接收来自 Apache Kafka 代理的信息。
kamelet	camel-kamelet-starter	调用 Kamelets
语言	camel-language-starter	使用 Camel 支持的任何语言执行脚本。
Log	camel-log-starter	将消息记录到底层日志记录机制。
Mail	camel-mail-starter	使用 imap、pop3 和 smtp 协议发送和接收电子邮件。
邮件 Microsoft OAuth	camel-mail-microsoft-oauth-starter	用于 Microsoft Exchange Online 的 Camel 邮件 OAuth2 验证器
MapStruct	camel-mapstruct-starter	使用 Mapstruct 类型 Conversion
Master	camel-master-starter	集群中只有一个消费者从给定端点消耗；如果 JVM 结束，则自动故障转移。

组件	工件	描述
Minio	camel-minio-starter	使用 Minio SDK 从 Minio Storage Service 存储和检索对象。
MLLP	camel-mllp-starter	使用 MLLP 协议与外部系统通信。
Mock	camel-mock-starter	使用模拟测试路由和介质规则。
MongoDB	camel-mongodb-starter	对 MongoDB 文档和集合执行操作。
Netty	camel-netty-starter	使用带有 Netty 4.x 的 TCP 或 UDP 的套接字级别网络。
paho	camel-paho-starter	使用 Eclipse Paho MQTT 客户端与 mq 消息代理通信。
paho mq 5	camel-paho-mqtt5-starter	使用 Eclipse Paho MQTT v5 客户端与 MQTT 消息代理进行通信。
quartz	camel-quartz-starter	使用 Quartz 2.x 调度程序调度消息发送。
Ref	camel-ref-starter	根据 Camel Registry 中的名称，将消息路由到端点动态查找。
REST	camel-rest-starter	公开 REST 服务或调用外部 REST 服务。
Salesforce	camel-salesforce-starter	使用 Java DTO 与 Salesforce 通讯。
SAP	camel-sap-starter	使用 SAP Java Connector (SAP JCo)库与 SAP 和 SAP IDoc 库进行双向通信，以中间文档(IDoc)格式传输文档。
scheduler	camel-scheduler-starter	使用 java.util.concurrent.ScheduledExecutorService 以指定间隔生成消息。
SEDA	camel-seda-starter	在同一 JVM 中异步调用任何 Camel 上下文的另一个端点。
Servlet	camel-servlet-starter	通过 Servlet 提供 HTTP 请求。
Slack	camel-slack-starter	向/从 Slack 发送和接收消息。

组件	工件	描述
Spring Batch	camel-spring-batch	将消息发送到 Spring Batch 以进一步处理。
Spring JDBC	camel-spring-jdbc	使用 Spring Transaction 支持通过 SQL 和 JDBC 访问数据库。
Spring LDAP	camel-spring-ldap	将过滤器用作消息有效负载，在 LDAP 服务器中执行搜索。
Spring RabbitMQ	camel-spring-rabbitmq	使用 Spring RabbitMQ 客户端从 RabbitMQ 发送和接收消息。
Spring Redis	camel-spring-redis	从 Redis 发送和接收信息。
Spring Webservice	camel-spring-ws	您可以使用此组件与 Spring Web Services 集成。它为访问 Web 服务和服务器端支持提供客户端支持，以创建您的合同优先 Web 服务。
SQL	camel-sql-starter	使用 Spring JDBC 执行 SQL 查询。
stub	camel-stub-starter	在开发或测试过程中了解任何物理端点。
电话报	camel-telegram-starter	发送并接收作为站报 Bot 报 Bot API 的消息。
timer	camel-timer-starter	使用 java.util.Timer 以指定间隔生成消息。
验证器	camel-validator-starter	使用 XML 架构和 JAXP 验证来验证有效负载。
Webhook	camel-webhook-starter	公开 Webhook 端点以接收用于其他 Camel 组件的推送通知。
XSLT	camel-xslt-starter	使用 XSLT 模板转换 XML 有效负载。

表 1.2. Camel 数据格式

组件	工件	描述
avro	camel-avro-starter	使用 Apache Avro 二进制数据格式序列化和反序列化消息。
Avro Jackson	camel-jackson-avro-starter	marshal POJOs 到 Avro，并使用 Jackson 回来。
bindy	camel-bindy-starter	使用 Camel Bindy 在 POJO 和键值对(KVP)格式之间进行 marshal 和 unmarshal
HL7	camel-hl7-starter	使用 HL7 MLLP codec marshal 和 unmarshal HL7 (Health Liberty)模型对象。
JacksonXML	camel-jacksonxml-starter	将 XML 有效负载解压缩为 POJO，并使用 Jackson 的 XMLMapper 扩展返回。
JAXB	camel-jaxb-starter	将 XML 有效负载解压缩为 POJO，并使用 JAXB2 XML marshalling 标准。
JSON Gson	camel-gson-starter	marshal POJOs 到 JSON，并使用 Gson 重新
JSON Jackson	camel-jackson-starter	marshal POJOs 到 JSON，并使用 Jackson 回来
protobuf Jackson	camel-jackson-protobuf-starter	marshal POJOs 到 Protobuf，并使用 Jackson 回来。
SOAP	camel-soap-starter	marshal Java 对象到 SOAP 消息，并返回。
zip 文件	camel-zipfile-starter	使用 java.util.zip.ZipStream 压缩和解压缩流。

表 1.3. Camel Languages

语言	工件	描述
常数	camel-core-starter	固定值只在路由启动过程中设置一次。
CSimple	camel-core-starter	评估编译的简单表达式。

语言	工件	描述
ExchangeProperty	camel-core-starter	从 Exchange 获取属性。
File	camel-core-starter	简单语言的文件相关功能。
标头	camel-core-starter	从 Exchange 获取标头。
jsonPath	camel-jsonpath-starter	根据 JSON 消息正文评估 JSONPath 表达式。
Ref	camel-core-starter	使用 registry 中的现有表达式。
Simple（简单）	camel-core-starter	评估 Camel 简单表达式。
令牌化	camel-core-starter	使用分隔符模式的令牌化文本有效负载。
XML 令牌化	camel-xml-jaxp-starter	令牌化 XML 有效负载。
XPath	camel-xpath-starter	根据 XML 有效负载评估 XPath 表达式。
XQuery	camel-saxon-starter	使用 XQuery 和 Saxon 查询和/或转换 XML 有效负载。

表 1.4. 其它扩展

扩展	工件	描述
kamelet Main	camel-kamelet-main-starter	主要运行 Kamelet 独立
OpenAPI Java	camel-openapi-java-starter	rest-dsl 支持使用 openapi doc
OpenTelemetry	camel-opentelemetry-starter	使用 OpenTelemetry 的分布式追踪
Spring Security	camel-spring-security	使用 Spring Security 的安全性
YAML DSL	camel-yaml-dsl-starter	使用 YAML 的 Camel DSL

1.4. STARTER 配置

清除并可访问的配置是任何应用程序的重要部分。Camel 启动器完全支持 Spring Boot 的外部配置机制。您还可以通过 Spring Beans 配置它们，以获取更复杂的用例。

1.4.1. 使用外部配置

在内部，每个 启动程序 都通过 Spring Boot 的 ConfigurationProperties 配置。每个配置参数都可以 以各种方式 设置（**application.[properties|json|yaml]** 文件、命令行参数、环境变量等。参数的格式为 **camel.[component|language|dataformat].[name].[parameter]**

例如，要配置 mq5 代理的 URL，您可以设置：

```
camel.component.paho-mqtt5.broker-url=tcp://localhost:61616
```

或者要配置 CSV 数据格式的 **delimeter** 为分号 (;)，您可以设置：

```
camel.dataformat.csv.delimiter=;
```

当将属性设置为所需类型时，Camel 将使用 Type Converter 机制。

您可以使用 sVirt **bean:name 在 Registry 中引用 Bean** ：

```
camel.component.jms.transactionManager=#bean:myjtaTransactionManager
```

Bean 通常以 Java 为单位创建：

```
@Bean("myjtaTransactionManager")
public JmsTransactionManager myjtaTransactionManager(PooledConnectionFactory pool) {
    JmsTransactionManager manager = new JmsTransactionManager(pool);
    manager.setDefaultTimeout(45);
    return manager;
}
```

Bean 也可以在 配置文件中 创建，但不建议用于复杂的用例。

1.4.2. 使用 Beans

也可以通过 Spring Beans 创建和配置启动程序。在创建入门程序之前，Camel 将首先在 Registry 中查找它（如果已存在）。例如，配置 Kafka 组件：

```
@Bean("kafka")
public KafkaComponent kafka(KafkaConfiguration kafkaconfiguration){
    return ComponentsBuilderFactory.kafka()
                    .brokers("{{kafka.host}}:{{kafka.port}}")
                    .build();
}
```

Bean 名称必须与您要配置的组件、数据格式或语言相同。如果没有在注解中指定 **Bean** 名称，它将设置为方法名称。

典型的 Camel Spring Boot 项目将组合使用外部配置和 Beans 来配置应用程序。有关如何配置 Camel Spring Boot 项目的更多信息，请参阅示例 存储库。

1.5. 使用 MAVEN 为 SPRING BOOT 应用程序生成 CAMEL

您可以使用 Maven archetype **org.apache.camel.archetypes:camel-archetype-spring-boot:3.20.1.redhat-00104** 来生成 Camel Spring Boot 应用程序。

流程

1. 运行以下命令：

```
mvn archetype:generate \
 -DarchetypeGroupId=org.apache.camel.archetypes \
 -DarchetypeArtifactId=camel-archetype-spring-boot \
 -DarchetypeVersion=3.20.1.redhat-00104 \
 -DgroupId=com.redhat \
 -DartifactId=csb-app \
 -Dversion=1.0-SNAPSHOT \
 -DinteractiveMode=false
```

2. 构建应用程序：

```
mvn package -f csb-app/pom.xml
```

3. 运行应用程序：

```
java -jar csb-app/target/csb-app-1.0-SNAPSHOT.jar
```

4. 通过检查由应用生成的 *Hello World* 输出的控制台日志来验证应用是否正在运行。

```
com.redhat.MySpringBootApplication      : Started MySpringBootApplication in 3.514
seconds (JVM running for 4.006)
Hello World
Hello World
```

1.6. 将 CAMEL SPRING BOOT 应用程序部署到 OPENSHIFT

本指南介绍了如何将 Camel Spring Boot 应用程序部署到 OpenShift。

先决条件

- 您可以访问 OpenShift 集群。
- 已安装 OpenShift **oc** CLI 客户端，或者您可以访问 OpenShift Container Platform Web 控制台。

注意

经认证的 OpenShift Container Platform 在 Camel for Spring Boot 支持的配置 中列出。以下示例中使用 Red Hat OpenJDK 11 (ubi8/openjdk-11)容器镜像。

流程

1. 按照本指南的 Maven 为 Spring Boot 应用程序第 1.5 节为 Spring Boot 应用程序生成 Camel for Spring Boot 应用程序 中的说明，生成 Camel for Spring Boot 应用程序。
2. 在存在修改的 pom.xml 目录下，执行以下命令。

```
mvn clean -DskipTests oc:deploy -Popenshift
```

3. 验证 CSB 应用程序是否在 pod 上运行。

```
oc logs -f dc/csb-app
```

1.7. 将补丁应用到 CAMEL SPRING BOOT

使用新的 **patch-maven-plugin** 机制，您可以对 Red Hat Camel Spring Boot 应用程序应用补丁。这种机制允许您更改由不同红帽应用程序 BOMS 提供的各个版本，例如 **camel-spring-boot-bom**。

patch-maven-plugin 的目的是将 Camel on Spring Boot BOM 中列出的依赖项版本更新为您要应用到应用程序的补丁元数据中指定的版本。

patch-maven-plugin 执行以下操作：

- 检索与当前红帽应用 BOM 相关的补丁元数据。
- 将版本更改应用到从 BOMs 导入的 <dependencyManagement>。

在 **patch-maven-plugin** 获取元数据后，它会迭代声明插件的项目的所有受管和直接依赖项，并使用 CVE/patch 元数据替换依赖项版本（如果匹配）。替换版本后，Maven 构建将继续并通过标准 Maven 项目阶段进行。

流程

以下流程解释了如何将补丁应用到您的应用程序。

1. 将 **patch-maven-plugin** 添加到项目的 **pom.xml** 文件中。**patch-maven-plugin** 的版本必须与 Spring Boot BOM 上的 Camel 版本相同。

```
<build>
  <plugins>
    <<plugin>
      <groupId>com.redhat.camel.springboot.platform</groupId>
      <artifactId>patch-maven-plugin</artifactId>
      <version>${camel-spring-boot-version}</version>
      <extensions>true</extensions>
    </plugin>
  </plugins>
</build>
```

2. 当您运行任何 **mvn clean deploy**,**mvn validate**, 或 **mvn dependencies:tree** 命令时，插件通过项目模块搜索，以检查模块是否使用 Red Hat Camel Spring Boot BOM。只有以下内容是支持的 BOM：

   - **com.redhat.camel.springboot.platform:camel-spring-boot-bom**: 用于 Camel Spring Boot BOM

3. 如果插件找不到上述 BOM，插件会显示以下信息：

```
$ mvn clean install

[INFO] Scanning for projects...
[INFO]

========== Red Hat Maven patching ==========

[INFO] [PATCH] No project in the reactor uses Camel on Spring Boot product BOM. Skipping
```

```
patch processing.
[INFO] [PATCH] Done in 7ms

================================================
```

4. 如果使用了正确的 BOM，则会找到补丁元数据，但不找到任何补丁。

```
$ mvn clean install

[INFO] Scanning for projects...
[INFO]

========== Red Hat Maven patching ==========

[INFO] [PATCH] Reading patch metadata and artifacts from 2 project repositories
[INFO] [PATCH]  - redhat-ga-repository: http://maven.repository.redhat.com/ga/
[INFO] [PATCH]  - central: https://repo.maven.apache.org/maven2
Downloading from redhat-ga-repository:
http://maven.repository.redhat.com/ga/com/redhat/camel/springboot/platform/redhat-camel-spring-boot-patch-metadata/maven-metadata.xml
Downloading from central:
https://repo.maven.apache.org/maven2/com/redhat/camel/springboot/platform/redhat-camel-spring-boot-patch-metadata/maven-metadata.xml
[INFO] [PATCH] Resolved patch descriptor:
/path/to/.m2/repository/com/redhat/camel/springboot/platform/redhat-camel-spring-boot-patch-metadata/3.20.1.redhat-00043/redhat-camel-spring-boot-patch-metadata-3.20.1.redhat-00043.xml
[INFO] [PATCH] Patch metadata found for com.redhat.camel.springboot.platform/camel-spring-boot-bom/[3.20,3.21)
[INFO] [PATCH] Done in 938ms

================================================
```

5. **patch-maven-plugin** 会尝试获取此 Maven 元数据。

- 对于带有 Camel Spring Boot BOM 的项目，**com.redhat.camel.springboot.platform:redhat-camel-spring-boot-patch-metadata/maven-metadata.xml** 已解决。此 XML 数据是带有 **com.redhat.camel.springboot.platform:redhat-camel-spring-boot-patch-metadata:RELEASE** 协调的工件的元数据。

  Maven 生成的元数据示例

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
  <groupId>com.redhat.camel.springboot.platform</groupId>
  <artifactId>redhat-camel-spring-boot-patch-metadata</artifactId>
  <versioning>
    <release>3.20.1.redhat-00041</release>
    <versions>
      <version>3.20.1.redhat-00041</version>
    </versions>
    <lastUpdated>20230322103858</lastUpdated>
  </versioning>
</metadata>
```

6. **patch-maven-plugin** 解析元数据，以选择应用到当前项目的版本。此操作只能针对使用带有特定版本的 Spring Boot BOM 上的 Camel 进行 Maven 项目。只有与版本范围或之后匹配的元数据才适用，它只获取元数据的最新版本。

7. **patch-maven-plugin** 收集远程 Maven 存储库列表，以下载由 **groupId**、**artifactId 和版本** 标识的补丁元数据。这些 Maven 存储库列在活跃配置集的项目 & **lt;repositories** > 元素中，以及 **settings.xml** 文件中的存储库。

```
$ mvn clean install
[INFO] Scanning for projects...
[INFO]

========== Red Hat Maven patching ==========

[INFO] [PATCH] Reading patch metadata and artifacts from 2 project repositories
[INFO] [PATCH]  - MRRC-GA: https://maven.repository.redhat.com/ga
[INFO] [PATCH]  - central: https://repo.maven.apache.org/maven2
```

8. 元数据来自远程存储库、本地存储库还是 ZIP 文件，它由 **patch-maven-plugin** 分析。获取的元数据包含 CVE 列表以及每个 CVE，我们有一个受影响的 Maven 工件列表（由 glob 模式和版本范围指定）以及包含给定 CVE 修复的版本。例如，

```
<?xml version="1.0" encoding="UTF-8" ?>

<<metadata xmlns="urn:redhat:patch-metadata:1">
   <product-bom groupId="com.redhat.camel.springboot.platform" artifactId="camel-spring-
boot-bom" versions="[3.20,3.21)" />
   <cves>
   </cves>
   <fixes>
      <fix id="HF0-1" description="logback-classic (Example) - Version Bump">
         <affects groupId="ch.qos.logback" artifactId="logback-classic" versions="[1.0,1.3.0)"
fix="1.3.0" />
      </fix>
   </fixes>
</metadata>
```

9. 最后，当迭代当前项目中所有受管依赖项时，会参考补丁元数据中指定的修复列表。匹配的这些依赖项（和受管依赖项）被改为固定的版本。例如：

```
$ mvn dependency:tree

[INFO] Scanning for projects...
[INFO]

========== Red Hat Maven patching ==========

[INFO] [PATCH] Reading patch metadata and artifacts from 3 project repositories
[INFO] [PATCH]  - redhat-ga-repository: http://maven.repository.redhat.com/ga/
[INFO] [PATCH]  - local: file:///path/to/.m2/repository
[INFO] [PATCH]  - central: https://repo.maven.apache.org/maven2
[INFO] [PATCH] Resolved patch
descriptor:/path/to/.m2/repository/com/redhat/camel/springboot/platform/redhat-camel-spring-
boot-patch-metadata/3.20.1.redhat-00043/redhat-camel-spring-boot-patch-metadata-
3.20.1.redhat-00043.xml
```

```
[INFO] [PATCH] Patch metadata found for com.redhat.camel.springboot.platform/camel-spring-boot-bom/[3.20,3.21)
[INFO] [PATCH]  - patch contains 1 patch fix
[INFO] [PATCH] Processing managed dependencies to apply patch fixes...
[INFO] [PATCH] - HF0-1: logback-classic (Example) - Version Bump
[INFO] [PATCH]   Applying change ch.qos.logback/logback-classic/[1.0,1.3.0) -> 1.3.0
[INFO] [PATCH]   Project com.test:yaml-routes
[INFO] [PATCH]    - managed dependency: ch.qos.logback/logback-classic/1.2.11 -> 1.3.0
[INFO] [PATCH] Done in 39ms

================================================
```

跳过补丁

如果您不想将特定的补丁应用到项目，**patch-maven-plugin** 会提供 **skip** 选项。假设已将 **patch-maven-plugin** 添加到项目的 **pom.xml** 文件中，并且您不想更改版本，您可以使用以下方法之一跳过补丁。

- 将 skip 选项添加到项目的 **pom.xml** 文件中，如下所示：

```
<build>
  <plugins>
    <plugin>
      <groupId>com.redhat.camel.springboot.platform</groupId>
      <artifactId>patch-maven-plugin</artifactId>
      <version>${camel-spring-boot-version}</version>
      <extensions>true</extensions>
      <configuration>
        <skip>true</skip>
      </configuration>
    </plugin>
  </plugins>
</build>
```

- 或者，在运行 **mvn** 命令时使用 **-DskipPatch** 选项，如下所示：

```
$ mvn clean install -DskipPatch
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< com.example:test-csb >-------------------------
[INFO] Building A Camel Spring Boot Route 1.0-SNAPSHOT
...
```

如以上输出中显示，**patch-maven-plugin** 没有被调用，这会导致补丁没有被应用到应用程序。

1.8. CAMEL REST DSL OPENAPI MAVEN 插件

Camel REST DSL OpenApi Maven 插件支持以下目标：

- camel-restdsl-openapi:generate - 从 OpenApi 规范生成消费者 REST DSL RouteBuilder 源代码
- camel-restdsl-openapi:generate-with-dto - 要从 OpenApi 规范生成消费者 REST DSL RouteBuilder 源代码，并通过 swagger-codegen-maven-plugin 生成 DTO 模型类。
- camel-restdsl-openapi:generate-xml - 要从 OpenApi 规范生成消费者 REST DSL XML 源代码

- camel-restdsl-openapi:generate-xml-with-dto - 要从 OpenApi 规范生成消费者 REST DSL XML 源代码，并使用通过 swagger-codegen-maven-plugin 生成的 DTO 模型类。
- camel-restdsl-openapi:generate-yaml - 要从 OpenApi 规范生成使用者 REST DSL YAML 源代码
- camel-restdsl-openapi:generate-yaml-with-dto - 要从 OpenApi 规范生成消费者 REST DSL YAML 源代码，并使用通过 swagger-codegen-maven-plugin 生成的 DTO 模型类。

1.8.1. 将插件添加到 Maven pom.xml

此插件可以通过将其添加到 **plugins** 部分来添加到 Maven **pom.xml** 文件中，例如在 Spring Boot 应用程序中：

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>

    <plugin>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-restdsl-openapi-plugin</artifactId>
      <version>{CamelCommunityVersion}</version>
    </plugin>

  </plugins>
</build>
```

然后，可以使用其前缀 **camel-restdsl-openapi** 执行插件，如下所示。

```
$mvn camel-restdsl-openapi:generate
```

1.8.2. camel-restdsl-openapi:generate

Camel REST DSL OpenApi Maven 插件的目标是从 Maven 生成 REST DSL RouteBuilder 实施源代码。

1.8.3. 选项

该插件支持可从命令行配置的以下选项（使用 **-D** 语法），或者在 **配置** 标签的 **pom.xml** 文件中定义。

参数	默认值	描述
skip	**false**	设置为 **true** 以跳过代码生成
filterOperation		仅用于包括指定的操作 ID。可以使用逗号分隔多个 id。可以使用通配符，例如 **find,** 使其包含以 **find** 开头的所有操作。

参数	默认值	描述
specificationUri	**src/spec/openapi.json**	OpenApi 规范的 URI，支持文件系统路径、HTTP 和 classpath 资源，默认为项目目录中的 **src/spec/openapi.json**。支持 JSON 和 YAML。
auth		在远程获取 OpenApi 规格定义时添加授权标头。使用逗号分隔多个值传递 name:header 的 URL 编码字符串。
className	from **title** 或 **RestDslRoute**	生成的类的名称，从 OpenApi 规格标题获取，或默认设置为 **RestDslRoute**
packageName	来自 **主机或 rest.dsl.generated**	生成的类的软件包名称，从 OpenApi 规格主机值或 **rest.dsl.generated** 获取
indent	**" "**	哪个缩进要使用的字符（默认为四个空格），您可以使用 **\t** 表示标签字符
outputDirectory	**generated-sources/restdsl-openapi**	在项目目录中放置生成的源文件的位置，默认为 **generate-sources/restdsl-openapi**
destinationGenerator		实现 **org.apache.camel.generator.openapi.DestinationGenerator** 接口的类的完全限定类名称，以自定义目的地端点
destinationToSyntax	**direct:${operationId}**	到 uri 的默认语法，即使用直接组件。
restConfiguration	**true**	是否包含生成其余配置，并检测要使用的其余组件。
apiContextPath		如果 **restConfiguration** 设为 true，则定义 openapi 端点路径。
clientRequestValidation	**false**	是否启用请求验证。
basePath		覆盖 OpenAPI 规格中定义的 api 基础路径。

参数	默认值	描述
requestMappingValues	/**	允许生成自定义 RequestMapping 映射值。多个映射值可传递为： **<requestMappingValues> <param>/my-api-path/ </param> <param>/my-other-path/ </param> </requestMappingValues>**

1.8.4. 带有 Servlet 组件的 Spring Boot 项目

如果 Maven 项目是 Spring Boot 项目，并且启用了 **restConfiguration**，并且 servlet 组件用作 REST 组件，则此插件将自动检测软件包名称（如果尚未明确配置 packageName），其中 **@SpringBootApplication** 主类被启用，并使用同样的软件包名称和所需的 **CamelRestController** 支持类。

1.8.5. camel-restdsl-openapi:generate-with-dto

作为 **生成** 目标，还可以通过自动执行 swagger-codegen-maven-plugin 来生成 DTO 模型类的 java 源代码，从 OpenApi 规范生成 DTO 模型类的 java 源代码。

此插件的范围有限，仅限于只支持使用 swagger-codegen-maven-plugin 生成模型 DTO 的良好工作集。如果您需要更多电源和灵活性，则直接使用 Swagger Codegen Maven 插件 来生成 DTO 而不是此插件。

DTO 类可能需要额外的依赖项，例如：

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10.1</version>
</dependency>
<dependency>
  <groupId>io.swagger.core.v3</groupId>
  <artifactId>swagger-core</artifactId>
  <version>2.2.8</version>
</dependency>
<dependency>
  <groupId>org.threeten</groupId>
  <artifactId>threetenbp</artifactId>
  <version>1.6.8</version>
</dependency>
```

1.8.6. 选项

插件支持以下 **附加选项**

参数	默认值	描述
swaggerCodegenMavenPluginVersion	3.0.36	要使用的 **io.swagger.codegen.v3:swagger-codegen-maven-plugin** maven 插件的版本。
modelOutput		目标输出路径（默认为 ${project.build.directory}/generated-sources/openapi）
modelPackage	**io.swagger.client.model**	用于生成的模型对象/类的软件包
modelNamePrefix		为模型类和枚举设置前或后缀
modelNameSuffix		为模型类和枚举设置前或后缀
modelWithXml	false	在生成的模型中启用 XML 注解（仅适用于提供对 JSON 和 XML 支持的库）
configOptions		将特定于语言的参数映射传递给 **swagger-codegen-maven-plugin**

1.8.7. camel-restdsl-openapi:generate-xml

Camel REST DSL OpenApi Maven 插件的 **camel-restdsl-openapi:generate-xml** 目标用于从 Maven 生成 REST DSL XML 实施源代码。

1.8.8. 选项

该插件支持可从命令行配置的以下选项（使用 **-D** 语法），或者在 < **configuration** > 标签中的 **pom.xml** 文件中定义。

参数	默认值	描述
skip	**false**	设置为 **true** 以跳过代码生成。
filterOperation		仅用于包括指定的操作 ID。可以使用逗号分隔多个 id。可以使用通配符，例如 **find,** 使其包含以 **find** 开头的所有操作。
specificationUri	**src/spec/openapi.json**	OpenApi 规范的 URI，支持文件系统路径、HTTP 和 classpath 资源，默认为项目目录中的 **src/spec/openapi.json**。支持 JSON 和 YAML。

参数	默认值	描述
auth		在远程获取 OpenApi 规格定义时添加授权标头。使用逗号分隔多个值传递 name:header 的 URL 编码字符串。
outputDirectory	**generated-sources/restdsl-openapi**	在项目目录中放置生成的源文件的位置，默认为 **generate-sources/restdsl-openapi**
fileName	**camel-rest.xml**	XML 文件的名称作为输出。
蓝图	**false**	如果启用，生成 合并蓝图 XML 而不是 Spring XML。
destinationGenerator		实现 **org.apache.camel.generator.openapi.DestinationGenerator** 接口的类的完全限定类名称，以自定义目的地端点
destinationToSyntax	**direct:${operationId}**	到 uri 的默认语法，即使用直接组件。
	restConfiguration	**true**
是否包含生成其余配置，并检测要使用的其余组件。	**apiContextPath**	
如果 **restConfiguration** 设为 **true**，则定义 openapi 端点路径。	**clientRequestValidation**	**false**
是否启用请求验证。	**basePath**	
覆盖 OpenAPI 规格中定义的 api 基础路径。	**requestMappingValues**	/**

1.8.9. camel-restdsl-openapi:generate-xml-with-dto

作为 **generate-xml** 目标，还可以通过自动执行 swagger-codegen-maven-plugin 来生成 DTO 模型类的 java 源代码，从 OpenApi 规范生成 DTO 模型类的 java 源代码。

此插件的范围有限，仅限于只支持使用 swagger-codegen-maven-plugin 生成模型 DTO 的良好工作集。如果您需要更多电源和灵活性，则直接使用 Swagger Codegen Maven 插件 来生成 DTO 而不是此插件。

DTO 类可能需要额外的依赖项，例如：

```
<dependency>
  <groupId>com.google.code.gson</groupId>
```

```
    <artifactId>gson</artifactId>
    <version>2.10.1</version>
  </dependency>
  <dependency>
    <groupId>io.swagger.core.v3</groupId>
    <artifactId>swagger-core</artifactId>
    <version>2.2.8</version>
  </dependency>
  <dependency>
    <groupId>org.threeten</groupId>
    <artifactId>threetenbp</artifactId>
    <version>1.6.8</version>
  </dependency>
```

1.8.10. 选项

插件支持以下 **附加选项**

参数	默认值	描述
swaggerCodegenMavenPluginVersion	3.0.36	要使用的 **io.swagger.codegen.v3:swagger-codegen-maven-plugin** maven 插件的版本。
modelOutput		目标输出路径（默认为 ${project.build.directory}/generated-sources/openapi）
modelPackage	**io.swagger.client.model**	用于生成的模型对象/类的软件包
modelNamePrefix		为模型类和枚举设置前或后缀
modelNameSuffix		为模型类和枚举设置前或后缀
modelWithXml	false	在生成的模型中启用 XML 注解（仅适用于提供对 JSON 和 XML 支持的库）
configOptions		将特定于语言的参数映射传递给 **swagger-codegen-maven-plugin**

1.8.11. camel-restdsl-openapi:generate-yaml

camel-restdsl-openapi:generate-yaml 目标用于从 Maven 生成 REST DSL YAML 实现源代码。

1.8.12. 选项

该插件支持可从命令行配置的以下选项（使用 **-D** 语法），或者在 < **configuration** > 标签中的 **pom.xml** 文件中定义。

参数	默认值	描述
skip	**false**	设置为 **true** 以跳过代码生成。
filterOperation		仅用于包括指定的操作 ID。可以使用逗号分隔多个 id。可以使用通配符，例如 **find,** 使其包含以 **find** 开头的所有操作。
specificationUri	**src/spec/openapi.json**	OpenApi 规范的 URI，支持文件系统路径、HTTP 和 classpath 资源，默认为项目目录中的 **src/spec/openapi.json**。支持 JSON 和 YAML。
auth		在远程获取 OpenApi 规格定义时添加授权标头。使用逗号分隔多个值传递 name:header 的 URL 编码字符串。
outputDirectory	**generated-sources/restdsl-openapi**	在项目目录中放置生成的源文件的位置，默认为 **generate-sources/restdsl-openapi**
fileName	**camel-rest.xml**	XML 文件的名称作为输出。
destinationGenerator		实现 **org.apache.camel.generator.openapi.DestinationGenerator** 接口的类的完全限定类名称，以自定义目的地端点
destinationToSyntax	**direct:${operationId}**	到 uri 的默认语法，即使用直接组件。
	restConfiguration	**true**
是否包含生成其余配置，并检测要使用的其余组件。	**apiContextPath**	
如果 **restConfiguration** 设为 **true**，则定义 openapi 端点路径。	**clientRequestValidation**	**false**
是否启用请求验证。	**basePath**	
覆盖 OpenAPI 规格中定义的 api 基础路径。	**requestMappingValues**	/**

1.8.13. camel-restdsl-openapi:generate-yaml-with-dto

作为 **generate-yaml** 目标，还可以通过自动执行 swagger-codegen-maven-plugin 来生成 DTO 模型类的 java 源代码，从 OpenApi 规范生成 DTO 模型类的 java 源代码。

此插件的范围有限，仅限于只支持使用 **swagger-codegen-maven-plugin** 生成模型 DTO 的良好工作集。如果您需要更多电源和灵活性，则直接使用 Swagger Codegen Maven 插件 来生成 DTO 而不是此插件。

DTO 类可能需要额外的依赖项，例如：

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10.1</version>
</dependency>
<dependency>
  <groupId>io.swagger.core.v3</groupId>
  <artifactId>swagger-core</artifactId>
  <version>2.2.8</version>
</dependency>
<dependency>
  <groupId>org.threeten</groupId>
  <artifactId>threetenbp</artifactId>
  <version>1.6.8</version>
</dependency>
```

1.8.14. 选项

插件支持以下 **附加选项**

参数	默认值	描述
swaggerCodegenMavenPluginVersion	3.0.36	要使用的 **io.swagger.codegen.v3:swagger-codegen-maven-plugin** maven 插件的版本。
modelOutput		目标输出路径（默认为 ${project.build.directory}/generated-sources/openapi）
modelPackage	**io.swagger.client.model**	用于生成的模型对象/类的软件包
modelNamePrefix		为模型类和枚举设置前或后缀
modelNameSuffix		为模型类和枚举设置前或后缀
modelWithXml	false	在生成的模型中启用 XML 注解（仅适用于提供对 JSON 和 XML 支持的库）

参数	默认值	描述
configOptions		将特定于语言的参数映射传递给 **swagger-codegen-maven-plugin**

1.9. 支持 FIPS 合规性

您可以在 x86_64 架构上安装使用 FIPS 验证的/Modules in Process 加密库的 OpenShift Container Platform 集群。

对于集群中的 Red Hat Enterprise Linux CoreOS (RHCOS)机器，当机器根据 install-config.yaml 文件中的选项的状态进行部署时，会应用此更改，该文件管理用户在集群部署期间更改的集群选项。在 Red Hat Enterprise Linux (RHEL)机器中，您必须在计划用作 worker 机器的机器上安装操作系统时启用 FIPS 模式。这些配置方法可确保集群满足 FIPS 合规审核的要求。在初始系统引导前，只启用 FIPS 验证的/Modules in Process 加密软件包。

因为您必须在集群首次引导前启用 FIPS，所以无法在部署集群后启用 FIPS。

1.9.1. OpenShift Container Platform 中的 FIPS 验证

OpenShift Container Platform 在 RHEL 和 RHCOS 中使用特定的 FIPS 验证的/Modules in Process 模块用于其操作系统组件。例如，当用户 SSH 到 OpenShift Container Platform 集群和容器时，这些连接会被正确加密。

OpenShift Container Platform 组件使用 Go 编写，并使用红帽的 Golang 编译器构建。当您为集群启用 FIPS 模式时，需要加密签名的所有 OpenShift Container Platform 组件都会调用 RHEL 和 RHCOS 加密库。

有关 FIPS 的详情，请参阅 FIPS 模式属性和限制

有关在 OpenShift 上部署 Camel Spring Boot 的详情，请参阅 如何将 Camel Spring Boot 应用程序部署到 OpenShift？

有关支持的配置的详情，请参考 Camel for Spring Boot 支持的配置

第 2 章 迁移到 CAMEL SPRING BOOT

本指南提供有关在 Spring Boot 上从 Red Hat Fuse 7 迁移到 Camel 3 的信息。

2.1. JAVA 版本

Camel 3 支持 Java 17 和 Java 11，但不支持 Java 8。

在 Java 11 中，JAXB 模块已从 JDK **中删除**，因此您需要将它们添加为 Maven 依赖项（如果您使用 JAXB，如在使用 XML DSL 或 camel-jaxb 组件时）：

```
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.1</version>
</dependency>

<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-core</artifactId>
  <version>2.3.0.1</version>
</dependency>

<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-impl</artifactId>
  <version>2.3.2</version>
</dependency>
```

注意 ：Java Platform, Standard Edition 11 Development Kit (JDK 11)在 Camel Spring Boot 3.x 版本中弃用，且不支持进一步的 4.x 版本。

2.2. CAMEL-CORE 的修改

在 Camel 3.x 中，**camel-core** 被分成多个 JAR，如下所示：

- camel-api
- camel-base
- camel-caffeine-lrucache
- camel-cloud
- camel-core
- camel-jaxp
- camel-main
- camel-management-api
- camel-management
- camel-support

- camel-util
- camel-util-json

Apache Camel 的 Maven 用户可以继续使用依赖 **camel-core**，其对所有模块具有传输的依赖关系，但 **camel-main** 除外，因此不需要迁移。

2.3. 对组件的修改

在 Camel 3.x 中，一些 camel-core 组件被移到独立的组件中。

- camel-attachments
- camel-bean
- camel-browse
- camel-controlbus
- camel-dataformat
- camel-dataset
- camel-direct
- camel-directvm
- camel-file
- camel-language
- camel-log
- camel-mock
- camel-ref
- camel-rest
- camel-saga
- camel-scheduler
- camel-seda
- camel-stub
- camel-timer
- camel-validator
- camel-vm
- camel-xpath
- camel-xslt

- camel-xslt-saxon
- camel-zip-deflater

2.4. SPRING BOOT 启动程序的更改

Spring Boot 启动器的 Maven **groupId** 从 **org.apache.camel** 更改为 **org.apache.camel.springboot**。

示例

使用：

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-component-starter</artifactId>
</dependency>
```

而不是

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-component-starter</artifactId>
</dependency>
```

2.5. 不支持每个应用程序有多个 CAMELCONTEXTS

对多个 CamelContexts 的支持已被删除，推荐支持每个部署的一个 CamelContext。因此，各种 Camel 注解上的 **context** 属性（如 **@EndpointInject**、**@Produce**、**@Consume** 等）已被删除。

2.6. 弃用的 API 和组件

Camel 2.x 中的所有已弃用的 API 和组件都已在 Camel 3 中删除。

2.6.1. 删除的组件

Camel 2.x 中的所有已弃用的组件都在 Camel 3.x 中删除，包括旧的 **camel-http**, **camel-hdfs**, **camel-mina**, **camel-mongodb**, **camel-netty**, **camel-netty-http**, **camel-quartz**, **camel-restlet** 和 **camel-rx** 组件。

- 删除了 **camel-jibx** 组件。
- 删除了 **camel-boon** 数据格式。
- 删除了 **camel-linkedin** 组件，因为 Linkedin API 1.0 不再被支持。CAMEL-13813 跟踪了对新 2.0 API 的支持。
- **camel-zookeeper** 删除了其路由策略功能，而是使用 **ZooKeeperClusterService** 或 **camel-zookeeper-master** 组件。
- **camel-jetty** 组件不再支持制作者（已删除），改为使用 **camel-http** 组件。
- 删除了 **twitter-streaming** 组件，因为它依赖于已弃用的 Tailoring Streaming API，且无法正常工作。

2.6.2. 重命名的组件

以下组件在 Camel 3.x 中重命名。

- camel **-microprofile-metrics** 已重命名为 **camel-micrometer**
- **测试** 组件已重命名为 **dataset-test**，并将 **camel-core** 移到 **camel-dataset** JAR 中。
- **http4** 组件已重命名为 **http**，它对应的组件软件包从 **org.apache.camel.component.http4** 重命名为 **org.apache.camel.component.http**。现在，支持的方案只是 **http** 和 **https**。
- **hdfs2** 组件已重命名为 **hdfs**，它对应于从 **org.apache.camel.component.hdfs2** 到 **org.apache.camel.component.hdfs** 的组件软件包。现在，支持的方案是 **hdfs**。
- **mina2** 组件已重命名为 **mina**，它对应于来自从 **org.apache.camel.component.mina2** 到 **org.apache.camel.component.mina** 2 的软件包。现在，支持的方案为 **mina**。
- **mongodb3** 组件已重命名为 **mongodb**，它是从 **org.apache.camel.component.mongodb3** 到 **org.apache.camel.component.mongodb** 的对应组件软件包。现在，支持的方案是 **mongodb**。
- **netty4-http** 组件已重命名为 **netty-http**，它将 **org.apache.camel.component.netty4.http** 中的相应组件软件包重命名为 **org.apache.camel.component.netty.http**。现在，支持的方案为 **netty-http**。
- **netty4** 组件已重命名为 **netty**，它是从 **org.apache.camel.component.netty4** 到 **org.apache.camel.component.netty** 的对应组件软件包。现在，支持的方案为 **netty**。
- **quartz2** 组件已重命名为 **quartz**，它对应于从 **org.apache.camel.component.quartz2** 到 **org.apache.camel.component.quartz** 的组件软件包。现在，支持的方案是 **quartz**。
- **rxjava2** 组件已重命名为 **rxjava**，它对应于从 **org.apache.camel.component.rxjava2** 到 **org.apache.camel.component.rxjava** 的组件软件包。
- 将 **camel-jetty9** 重命名为 **camel-jetty**。现在，支持的方案为 **jetty**。

2.7. CAMEL 组件的更改

2.7.1. 模拟组件

模拟 组件已从 **camel-core** 移出。由于这种方法在其 *断言条款构建器上* 被移除。

2.7.2. ActiveMQ

如果使用 **activemq-camel** 组件，您应该迁移到使用 **camel-activemq** 组件，其中组件名称已从 **org.apache.activemq.camel.component.ActiveMQComponent** 改为 **org.apache.camel.component.activemq.ActiveMQComponent**。

2.7.3. AWS

组件 **camel-aws** 已分成多个组件：

- camel-aws-cw
- camel-aws-ddb （包含 ddb 和 ddbstreams 组件）

- camel-aws-ec2
- camel-aws-iam
- camel-aws-kinesis （包含 kinesis 和 kinesis-firehose 组件）
- camel-aws-kms
- camel-aws-lambda
- camel-aws-mq
- camel-aws-s3
- camel-aws-sdb
- camel-aws-ses
- camel-aws-sns
- camel-aws-sqs
- camel-aws-swf

注意

建议为这些组件添加 specifc 依赖项。

2.7.4. Camel CXF

camel-cxf JAR 已分为 SOAP 与 REST 和 Spring JAR。当从 **came-cxf** 进行迁移时，建议从以下列表中选择特定的 JAR。

- **camel-cxf-soap**
- **camel-cxf-spring-soap**
- **camel-cxf-rest**
- **camel-cxf-spring-rest**
- **camel-cxf-transport**
- **camel-cxf-spring-transport**

例如，如果您使用 CXF 用于 SOAP 并使用 Spring XML，那么在从 **camel-cxf** 进行迁移时，请选择 **camel-cxf-spring-soap** 和 **camel-cxf-spring-transport**。

使用 Spring Boot 时，当您从 **camel-cxf-starter** 迁移到 SOAP 或 REST 时，从以下入门中选择：

- **camel-cxf-soap-starter**
- **camel-cxf-rest-starter**

camel-cxf XML XSD 模式也更改了命名空间。

表 2.1. 对命名空间的更改

旧命名空间	新命名空间
http://camel.apache.org/schema/cxf	http://camel.apache.org/schema/cxf/jaxws
http://camel.apache.org/schema/cxf/camel-cxf.xsd	http://camel.apache.org/schema/cxf/jaxws/camel-cxf.xsd
http://camel.apache.org/schema/cxf	http://camel.apache.org/schema/cxf/jaxrs
http://camel.apache.org/schema/cxf/camel-cxf.xsd	http://camel.apache.org/schema/cxf/jaxrs/camel-cxf.xsd

camel-cxf SOAP 组件被移到一个新的 **jaxws** 子软件包，即 **org.apache.camel.component.cxf** 现在是 **org.apache.camel.component.cxf.jaws**。例如，**CxfComponent** 类现在位于 **org.apache.camel.component.cxf.jaxws**。

2.7.5. FHIR

camel-fhir 组件已将其 hapi-fhir 依赖项升级到 4.1.0。默认 FHIR 版本已改为 R4。因此，如果需要 DSTU3，则必须明确设置它。

2.7.6. Kafka

camel-kafka 组件删除了选项 **bridgeEndpoint** 和 **circularTopicDetection**，因为该组件不再需要，因为组件在 Camel 2.x 上可以正常工作。换句话说，**camel-kafka** 将向来自端点 uri 的主题发送消息。要覆盖此功能，请使用带有新主题的 **KafkaConstants.OVERRIDE_TOPIC** 标头。请参阅 **camel-kafka** 组件文档以了解更多信息。

2.7.7. 电话报

camel-telegram 组件已将授权令牌从 uri-path 移到查询参数，例如 migrate

```
telegram:bots/myTokenHere
```

to

```
telegram:bots?authorizationToken=myTokenHere
```

2.7.8. JMX

如果您只使用 **camel-core** 作为依赖项运行 Camel 独立，并且您希望启用 JMX，那么您需要将 **camel-management** 添加为依赖项。

对于使用 **ManagedCamelContext**，现在需要从 **CamelContext** 获取此扩展，如下所示：

```
ManagedCamelContext managed = camelContext.getExtension(ManagedCamelContext.class);
```

2.7.9. XSLT

XSLT 组件已从 camel-core 移到 **camel-xslt** 和 **camel-xslt-saxon** 中。组件被分开，因此 **camel-xslt** 使

用 JDK XSTL 引擎(Xalan)和 **camel-xslt-saxon** 是使用 Saxon 时。这意味着您应该使用 **xslt** 和 **xslt-saxon** 作为 Camel 端点 URI 中的组件名称。如果您使用 XSLT 聚合策略，则使用 **org.apache.camel.component.xslt.saxon.XsltSaxonAggregationStrategy** 用于 Saxon 支持。在使用 xslt 构建器时，将 **org.apache.camel.component.xslt.saxon.XsltSaxonBuilder** 用于 Saxon 支持。另请注意，只有 **camel-xslt-saxon** 也支持 **allowStax**，因为 JDK XSLT 不支持它。

2.7.10. XML DSL 迁移

XML DSL 稍有变化。

自定义负载均衡器 EIP 已从 < **custom> 改为 &lt**; **customLoadBalancer>**

在 **<secureXML>** tag 中，XMLSecurity 数据格式将属性 **keyOrTrustStoreParametersId** 重新命名为 **keyOrTrustStoreParametersRef**。

& **lt;zipFile&** gt; 数据格式已重命名为 < **zipfile>**。

2.8. 迁移 CAMEL MAVEN 插件

camel-maven-plugin 已分成两个 maven 插件：

camel-maven-plugin

camel-maven-plugin 具有 **运行** 目标，旨在快速运行 Camel 应用程序。如需更多信息，请参阅 https://camel.apache.org/manual/camel-maven-plugin.html。

camel-report-maven-plugin

camel-report-maven-plugin 具有 **validate** 和 **route-coverage** 目标，用于生成 Camel 项目报告，如验证 Camel 端点 URI 和路由覆盖报告等。如需更多信息，请参阅 https://camel.apache.org/manual/camel-report-maven-plugin.html。