Red Hat Fuse 7.13

部署到 JBoss EAP

将应用程序软件包部署到 JBoss Enterprise Application Platform (EAP)容器中

Last Updated: 2024-06-12

Red Hat Fuse 7.13 部署到 JBoss EAP

将应用程序软件包部署到 JBoss Enterprise Application Platform (EAP)容器中

法律通告

摘要

本指南描述了将应用程序部署到 JBoss EAP 容器中的选项。

目录

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。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看我们的 CTO Chris Wright 信息。

第 1 章 JBOSS FUSE ON JBOSS EAP 部署概述

在将 Fuse on EAP 软件包安装到 JBoss EAP 容器后，您可以在 Red Hat JBoss Enterprise Application Platform (JBoss EAP)上部署 Fuse 应用程序。

本节介绍使用 EAP 子系统上的 Camel 的部署模型。Fuse 中的 Apache Camel 允许您选择容器来运行集成应用程序。

注意

红帽 JBoss EAP 具有一系列应用程序部署和配置选项，可同时满足管理员和开发人员。有关 JBOSS EAP 配置和部署过程的更多信息，请参阅红帽 JBoss EAP 配置指南。

1.1. 支持的产品版本

要查看支持 Fuse 7.13 的 JBoss EAP 的最新版本，请参阅 支持的配置页面。

1.2. CAMEL ON EAP SUBSYSTEM

EAP 子系统上的 Camel 将 Apache Camel 直接集成到 JBoss EAP 容器中。将 Fuse on EAP 软件包安装到 JBoss EAP 容器后，可以使用此子系统。它为 Camel 部署提供了许多优势，包括简化 Camel 组件的部署，并与底层 JBoss EAP 容器紧密集成。

红帽建议您使用 Camel on EAP Subsystem 部署模型在 JBoss EAP 上部署 Apache Camel 应用程序。

第 2 章 在 JBOSS EAP 上构建应用程序

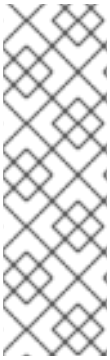
2.1. 概述

以下示例演示了将 **camel-cdi** 组件与 EAP 上的红帽 Fuse 搭配使用，以将 CDI Bean 与 Camel 路由集成。

在本例中，Camel 路由从 servlet **HTTP GET** 请求获取消息有效负载，并将其传递给直接端点。然后，它会将有效负载传递给 Camel CDI bean 调用，以生成消息响应并在 Web 浏览器页面中显示输出。

2.2. 运行项目

在运行项目之前，请确保您的设置包括 Maven 和红帽 Fuse 的应用服务器。



注意

如果使用 Java 17，则必须在启动应用程序前启用 JBoss EAP Elytron 子系统，方法是使用 JBoss EAP Elytron subsystem。

- 对于 Linux: **${JBOSS_HOME}/bin/jboss-cli.sh --file=docs/examples/enable-elytron-se17.cli -Dconfig=standalone-full.xml**
- 对于 Windows: **%JBOSS_HOME%\bin\jboss-cli.bat --file=docss\enable-elytron-se17.cli -Dconfig=standalone-full.xml**

执行以下步骤来运行项目：

1. 以独立模式启动应用服务器：
   - 对于 Linux ：**${JBOSS_HOME}/bin/standalone.sh -c standalone-full.xml**
   - 对于 Windows： **%JBOSS_HOME%\bin\standalone.bat -c standalone-full.xml**
2. 构建和部署项目： **mvn install -Pdeploy**
3. 现在，浏览 **http://localhost:8080/example-camel-cdi/?name=World** 位置。在 web 页面中，以下消息 **Hello World from 127.0.0.1** 显示为一个输出。另外，您可以在 **MyRouteBuilder.java** 类下查看 Camel Route，如下所示：

```
from("direct:start").bean("helloBean");
```

bean DSL 使 Camel 在 bean 注册表中查找名为 **helloBean** 的 bean。另外，bean 也可用于 Camel，因为 some **Bean** 类。通过使用 **@Named** 注释，**camel-cdi** 将 bean 添加到 Camel bean registry。

```
@Named("helloBean")

public class SomeBean {

    public String someMethod(String name) throws Exception {

        return String.format("Hello %s from %s", name, InetAddress.getLocalHost().getHostAddress());
```

```
    }

}
```

如需更多信息，请参阅 **$ EAP_HOME/quickstarts/camel/camel-cdi** 目录。

2.3. JBOSS EAP 的 BOM 文件

Maven Bill of Materials (BOM) 文件的目的是提供一组策展的 Maven 依赖项版本，这些版本可以很好地协同工作，从而为您为每个 Maven 工件单独定义版本。

JBoss EAP 的 Fuse BOM 提供以下优点：

- 定义 Maven 依赖项的版本，以便在将依赖项添加到 POM 时不需要指定版本。
- 定义一组对特定版本的 Fuse 经过全面测试并支持的策展依赖关系。
- 简化 Fuse 的升级。

重要

红帽仅支持由 Fuse BOM 定义的一组依赖项。

要将 BOM 文件合并到 Maven 项目中，请在项目的 **pom.xml** 文件中指定 dependencies **Management** 元素（或者，在父 POM 文件中），如下例所示：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- configure the versions you want to use here -->
    <fuse.version>7.13.0.fuse-7_13_0-00012-redhat-00001</fuse.version>

  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-eap-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

使用依赖项管理机制指定 BOM 后，可以在不指定工件版本 *的情况下* 向 POM 添加 Maven 依赖项。例如，要为 **camel-velocity** 组件添加依赖项，您可以在 POM 中的 **dependencies** 元素中添加以下 XML 片段：

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
  <scope>provided</scope>
</dependency>
```

注意如何在这个依赖项定义中省略 **version** 元素。

第 3 章 功能

本章提供有关 Camel on EAP 功能的必要信息。

Camel 上下文定义

Camel Contexts 可以在 standalone-camel.xml 和 domain.xml 中配置，作为子系统定义的一部分，如下所示

```
<subsystem xmlns="urn:jboss:domain:camel:1.0">
  <camelContext id="system-context-1">
    <![CDATA[
    <route>
      <from uri="direct:start"/>
      <transform>
        <simple>Hello #{body}</simple>
      </transform>
    </route>
    ]]>
  </camelContext>
</subsystem>
```

Camel Context Deployments

您可以使用 **-camel-context.xml** 后缀将 camel 上下文部署到 JBoss EAP：

- 独立 XML 文件
- 另一个支持的部署的一部分

部署可以包含多个 **-camel-context.xml** 文件。

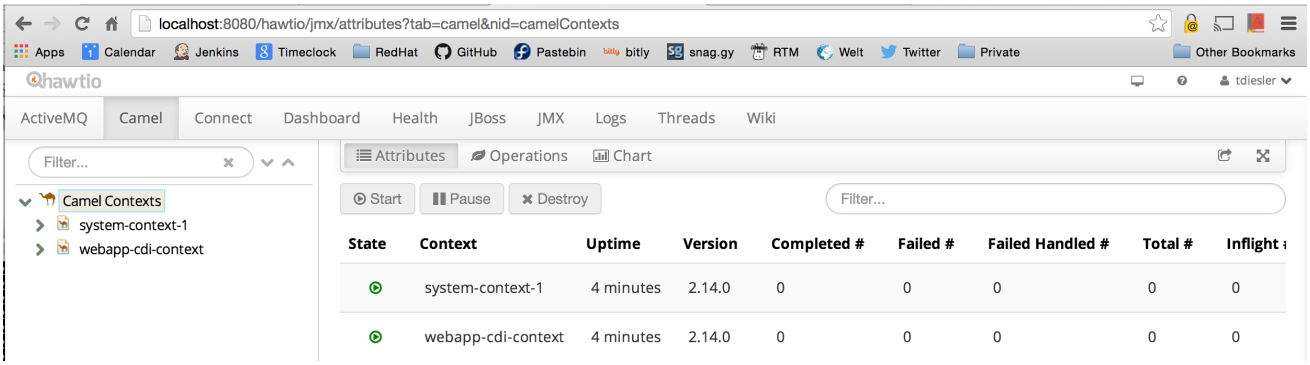
部署的 Camel 上下文是 CDI 注入项，如下所示

```
@Resource(lookup = "java:jboss/camel/context/mycontext")
CamelContext camelContext;
[discrete]
### Management Console
```

默认情况下，对管理控制台的访问是安全的。因此，您需要首先设置管理用户。

```
$ bin/add-user.sh

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
```

Hawt.io 控制台应该显示来自子系统配置的 camel 上下文。



JAAS 测试支持

Camel on EAP 测试套件使用 WildFly Arquillian 管理容器。这可以连接到已在运行的 JBoss EAP 实例，或者根据需要启动单机服务器实例。

实施了很多测试增强程序，允许您将这些 Camel on EAP 特定类型注入您的 JAAS 测试案例中。

```
@ArquillianResource
CamelContextFactory contextFactory;

@ArquillianResource
CamelContextRegistry contextRegistry;
```

第 4 章 配置

本章提供有关 Camel 子系统和部署配置的必要信息。

Camel 子系统配置

Camel 子系统配置可能包含静态路由路由。但是，系统会自动启动路由。

```
<subsystem xmlns="urn:jboss:domain:camel:1.0">
  <camelContext id="system-context-1">
    <![CDATA[
    <route>
      <from uri="direct:start"/>
      <transform>
        <simple>Hello #{body}</simple>
      </transform>
    </route>
    ]]>
  </camelContext>
</subsystem>
```

Camel 部署配置

如果要修改 Camel 部署的默认配置，您可以编辑部署中的 **WEB-INF/jboss-all.xml** 或 **META-INF/jboss-all.xml** 配置文件。

使用 **jboss-all.xml 文件中的 <jboss-camel** > XML 元素来控制 camel 配置。

禁用 Camel 子系统

如果您不想将 camel 子系统添加到部署中，请在 **jboss-camel** XML 元素上设置 **enabled="false"** 属性。

jboss-all.xml 文件示例：

```
<jboss umlns="urn:jboss:1.0">
  <jboss-camel xmlns="urn:jboss:jboss-camel:1.0" enabled="false"/>
</jboss>
```

选择组件

如果您添加嵌套的 & **lt;component** & **gt; 或 <component-module** > XML 元素，则不会将默认 Camel 组件列表添加到部署中，只有指定的组件会添加到您的部署中。

jboss-all.xml 文件示例：

```
<jboss umlns="urn:jboss:1.0">
  <jboss-camel xmlns="urn:jboss:jboss-camel:1.0">
    <component name="camel-ftp"/>
    <component-module name="org.apache.camel.component.rss"/>
  </jboss-camel>
</jboss>
```

第 5 章 JARKARTA EE 集成

本章提供有关使用 Jarkarta EE 集成点的必要信息。

5.1. CDI

Camel CDI 组件为 Apache Camel 提供自动配置，使用 CDI 作为依赖项注入框架。但是，它基于惯例的配置。它实现了标准的 camel bean 集成，以便您可以在 CDI Bean 中轻松使用 Camel 注解。

有关 CDI 的更多信息，请参阅 cdi 文档。

以下示例演示了如何通过路由来消耗 Camel 上下文并达到sosci。

```
@Startup
@ApplicationScoped
@ContextName("cdi-context")
public class MyRouteBuilder extends RouteBuilder {

    @Override
    public void configure() throws Exception {
     from("direct:start").transform(body().prepend("Hi"));
    }
}
```

```
@Inject
@ContextName("cdi-context")
private CamelContext camelctx;
```

5.1.1. 导入 XML DSL 配置

Camel CDI 集成允许您通过 **@ImportResource** 注释导入现有的 XML DSL 文件：

```
@ImportResource("camel-context.xml")
class MyBean {
}
```

注意

导入文件的位置必须存在于部署类路径上。将文件放置到 **WEB-INF** 等位置。但是，**WEB-INF/classes** 将可以正常工作。

5.2. EJB

管理支持通过 ejb 组件提供，该组件与 EJB3 子系统集成。

```
CamelContext camelctx = new DefaultCamelContext();
    camelctx.addRoutes(new RouteBuilder() {
        @Override
        public void configure() throws Exception {
            from("direct:start").to("ejb:java:module/HelloBean");
        }
    });
```

5.3. JAXB

JAXB 支持通过 Camel JAXB 数据格式 提供。

Camel 支持 unmarshalling XML 数据到 JAXB 注解的类，以及从类到 XML 的 marshalling。以下演示了一个简单的 Camel 路由，用于处理 Camel JAXB 数据格式类的 marshalling 和 unmarshalling。

5.3.1. JAXB Annotated 类

```
@XmlRootElement(name = "customer")
@XmlAccessorType(XmlAccessType.FIELD)
public class Customer implements Serializable {

    private String firstName;
    private String lastName;

    public Customer() {
    }

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

5.3.2. JAXB 类 XML 表示

```
<customer xmlns="http://org/wildfly/test/jaxb/model/Customer">
    <firstName>John</firstName>
    <lastName>Doe</lastName>
</customer>
```

5.3.3. Camel JAXB Unmarshalling

```
WildFlyCamelContext camelctx = contextFactory.createCamelContext(getClass().getClassLoader());

final JaxbDataFormat jaxb = new JaxbDataFormat();
```

```
jaxb.setContextPath("org.wildfly.camel.test.jaxb.model");

camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start")
        .unmarshal(jaxb);
    }
});
camelctx.start();

ProducerTemplate producer = camelctx.createProducerTemplate();

// Send an XML representation of the customer to the direct:start endpoint
Customer customer = producer.requestBody("direct:start", readCustomerXml(), Customer.class);
Assert.assertEquals("John", customer.getFirstName());
Assert.assertEquals("Doe", customer.getLastName());
```

5.3.4. Camel JAXB Marshalling

```
WildFlyCamelContext camelctx = contextFactory.createCamelContext();

final JaxbDataFormat jaxb = new JaxbDataFormat();
jaxb.setContextPath("org.wildfly.camel.test.jaxb.model");

camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start")
        .marshal(jaxb);
    }
});
camelctx.start();

ProducerTemplate producer = camelctx.createProducerTemplate();
Customer customer = new Customer("John", "Doe");
String customerXML = producer.requestBody("direct:start", customer, String.class);
Assert.assertEquals(readCustomerXml(), customerXML);
```

5.4. JAX-RS

JAX-RS 支持由 Camel CXF-RS 提供。

5.4.1. cxf-RS Producer

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

```
spring.xsd">

    <cxf:rsClient id="cxfProducer"
                  address="http://localhost:8080/rest"
                  serviceClass="org.wildfly.camel.examples.cxf.jaxrs.GreetingService" />

    <camelContext id="cxfrs-camel-context" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="direct:start" />
            <setHeader headerName="operationName">
                <simple>greet</simple>
            </setHeader>
            <setHeader headerName="CamelCxfRsUsingHttpAPI">
                <constant>false</constant>
            </setHeader>
            <to uri="cxfrs:bean:cxfProducer" />
        </route>
    </camelContext>
</beans>
```

5.4.2. cxf-RS Consumer

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

    <cxf:rsServer id="cxfConsumer"
                  address="http://localhost:8080/rest"
                  serviceClass="org.wildfly.camel.examples.cxf.jaxrs.GreetingService" />

    <camelContext id="cxfrs-camel-context" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="cxfrs:bean:cxfConsumer" />
            <setBody>
                <constant>Hello world</constant>
            </setBody>
        </route>
    </camelContext>
</beans>
```

5.4.3. 使用 Camel REST DSL 的 JAX-RS Consumer

Camel REST DSL 提供编写充当 JAX-RS 用户的 Camel 路由的能力。以下 RouteBuilder 类演示了这一点。

```
@Startup
@ApplicationScoped
@ContextName("rest-camel-context")
```

```
public class RestConsumerRouteBuilder extends RouteBuilder {
  @Override
  public void configure() throws Exception {

    // Use the camel-undertow component to provide REST integration
    restConfiguration().component("undertow")
      .contextPath("/rest").port(8080).bindingMode(RestBindingMode.json);

    rest("/customer")
      // GET /rest/customer
      .get()
        .produces(MediaType.APPLICATION_JSON)
        .to("direct:getCustomers")
      // GET /rest/customer/1
      .get("/{id}")
        .produces(MediaType.APPLICATION_JSON)
        .to("direct:getCustomer")
      // POST /rest/customer
      .post()
        .type(Customer.class)
        .to("direct:createCustomer");
      // PUT /rest/customer
      .put()
        .type(Customer.class)
        .to("direct:updateCustomer");
      // DELETE /rest/customer/1
      .delete("/{id}")
        .to("direct:deleteCustomer");
  }
}
```

通过设置绑定模式，Camel 可以通过指定 'produces （）' 或 'type （）' 配置步骤来对 JSON 数据进行 marshal 和 unmarshal JSON 数据。

注意

- REST DSL 配置以 **restConfiguration （）.component ("undertow")** 开始。
- EAP 子系统上的 Camel 只支持 camel-servlet 和 camel-undertow 组件以用于 REST DSL。但是，如果您配置其他组件，则它无法正常工作。

5.4.4. 安全性

请参阅 JAX-RS 安全性部分。

5.4.5. EAP 上的 Fuse 中的快速入门示例

在 EAP 安装的 Fuse 中提供了快速入门示例，请访问 **quickstarts/camel/camel-cxf-jaxrs** 目录。

5.5. JAX-WS

WebService 支持通过 CXF 组件提供，该组件与也使用 Apache CXF 的 JBoss EAP WebServices 子系统集成。

5.5.1. JAX-WS CXF Producer

以下代码示例使用 CXF 来使用由 WildFly Web 服务子系统部署的 Web 服务。

5.5.1.1. JAX-WS web 服务

以下简单 Web 服务具有一个简单的"greet"方法，它将把两个字符串参数串联在一起并返回它们。

当 JBoss EAP Web 服务子系统检测到包含 JAX-WS 注释的类时，它会引导 CXF 端点。在本例中，服务端点将位于 http://hostname:port/context-root/greeting。

```
// Service interface
@WebService(name = "greeting")
public interface GreetingService {
    @WebMethod(operationName = "greet", action = "urn:greet")
    String greet(@WebParam(name = "message") String message, @WebParam(name = "name")
String name);
}

// Service implementation
public class GreetingServiceImpl implements GreetingService{
    public String greet(String message, String name) {
        return message + " " + name ;
    }
}
```

5.5.1.2. Camel 路由配置

此 RouteBuilder 配置 CXF producer 端点，该端点将使用上面定义的"greeting"Web 服务。CDI 与 camel-cdi 组件结合使用，用于引导 RouteBuilder 和 CamelContext。

```
@Startup
@ApplicationScoped
@ContextName("cxf-camel-context")
public class CxfRouteBuilder extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from("direct:start")
        .to("cxf://http://localhost:8080/example-camel-cxf/greeting?serviceClass=" +
GreetingService.class.getName());
    }
}
```

greeting web service 'greet' 需要两个参数。它们可以通过 **ProducerTemplate** 提供给上述路由。Web 服务方法参数值通过构造作为交换正文传递的对象数组来配置。

```
String message = "Hello"
String name = "Kermit"

ProducerTemplate producer = camelContext.createProducerTemplate();
Object[] serviceParams = new Object[] {message, name};
String result = producer.requestBody("direct:start", serviceParams, String.class);
```

5.5.2. Camel CXF JAX-WS Consumer

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

    <cxf:cxfEndpoint id="cxfConsumer"
                     address="http://localhost:8080/webservices/greeting"
                     serviceClass="org.wildfly.camel.examples.cxf.jaxws.GreetingService" />

    <camelContext id="cxfws-camel-context" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="cxf:bean:cxfConsumer" />
            <to uri="log:ws" />
        </route>
    </camelContext>

</beans>
```

5.5.3. 安全性

请参阅 JAX-WS 安全性部分。

5.5.4. EAP 上的 Fuse 中的快速入门示例

在 EAP 安装的 Fuse 中提供了快速入门示例，请访问 **quickstarts/camel/camel-cxf-jaxws** 目录。

5.6. JMS

消息传递支持通过 JMS 组件提供，该组件与 JBoss EAP Messaging (ActiveMQ Artemis)子系统集成。

通过配置供应商特定资源适配器或不可用，可以使用 JBoss Generic JMS 资源适配器与其他 JMS 实现集成。

5.6.1. JBoss EAP JMS 配置

您可以通过标准 JBoss EAP XML 配置文件配置 JBoss EAP 消息传递子系统。https://docs.jboss.org/author/display/WFLY8/Messaging+configuration例如： standalone.xml 或 domain.xml。

例如，如下是您在内存实例中使用嵌入的 ActiveMQ Artemis。您首先在 messaging 子系统中配置一个新的 JMS 队列，方法是将以下 XML 配置添加到 jms-destinations 部分：

```
<jms-queue name="WildFlyCamelQueue">
  <entry name="java:/jms/queue/WildFlyCamelQueue"/>
</jms-queue>
```

或者，也可以使用 CLI 脚本添加队列。

```
$ jms-queue add --queue-address=WildFlyCamelQueue --
entries=queue/WildFlyCamelQueue,java:/jms/queue/WildFlyCamelQueue
```

另外，您可以在自定义 jms.xml 部署描述符中创建 **messaging-deployment** 配置。如需更多信息，请参阅 JBoss EAP messaging 子系统文档中的"部署 -jms.xml 文件"部分。

5.6.2. Camel 路由配置

以下 JMS 生成者和消费者示例使用 JBoss EAP 的嵌入式 ActiveMQ Artemis 服务器发布和使用消息到目的地或从目的地使用。

这个示例还将 CDI 与 camel-cdi 组件一起使用。JMS ConnectionFactory 实例通过 JNDI 查找注入到 Camel RouteBuilder 中。

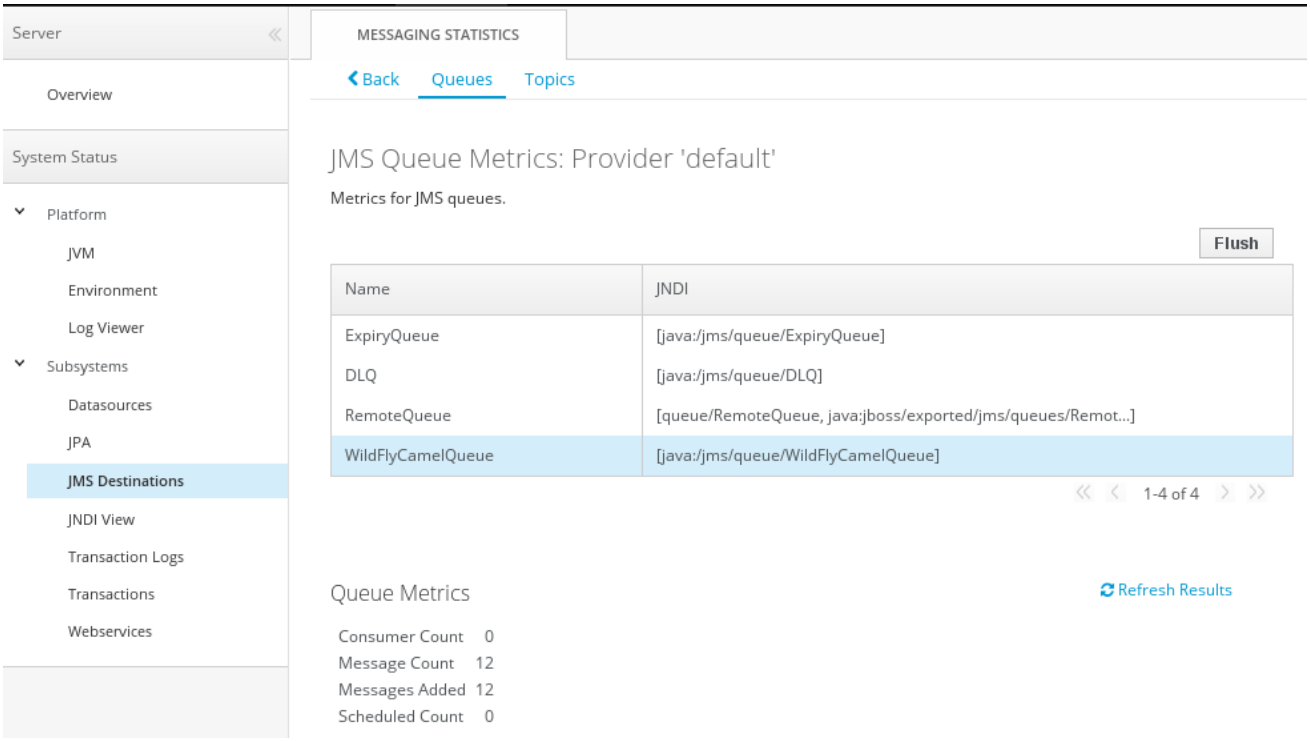
5.6.2.1. JMS Producer

DefaultJMSConnectionFactory 连接工厂从 JNDI 注入到 RouteBuilder 中。在 JBoss EAP XML 配置下，您可以在 messaging 子系统中找到连接工厂。

接下来，计时器端点每 10 秒运行一次，将 XML 有效负载发送到之前配置的 WildFlyCamelQueue 目的地。

```
@Startup
@ApplicationScoped
@ContextName("jms-camel-context")
public class JmsRouteBuilder extends RouteBuilder {

  @Resource(mappedName = "java:jboss/DefaultJMSConnectionFactory")
  private ConnectionFactory connectionFactory;

  @Override
  public void configure() throws Exception {
    JmsComponent component = new JmsComponent();
    component.setConnectionFactory(connectionFactory);

    getContext().addComponent("jms", component);

    from("timer://sendJMSMessage?fixedRate=true&period=10000")
    .transform(constant("<?xml version='1.0><message><greeting>hello world</greeting>
</message>"))
    .to("jms:queue:WildFlyCamelQueue")
    .log("JMS Message sent");
  }
}
```

每次将一个 JMS 消息添加到 WildFlyCamelQueue 目的地时，都会向控制台输出一条日志消息。要验证消息是否真正放置在队列中，您可以使用 JBoss EAP 管理控制台。



5.6.2.2. JMS Consumer

要使用 JMS 消息，Camel RouteBuilder 实施与制作者示例类似。

如前文所述，连接工厂从 JNDI 发现，注入并在 JMSComponent 实例上设置。

当 JMS 端点消耗来自 WildFlyCamelQueue 目的地的消息时，内容会记录到控制台。

```
@Override
public void configure() throws Exception {
  JmsComponent component = new JmsComponent();
  component.setConnectionFactory(connectionFactory);

  getContext().addComponent("jms", component);

  from("jms:queue:WildFlyCamelQueue")
  .to("log:jms?showAll=true");
}
```

5.6.2.3. JMS Transactions

要让 Camel JMS 路由能够参与 JMS 事务，需要进行一些额外的配置。由于 camel-jms 围绕 spring-jms 构建，您需要配置一些 Spring 类，以便它们能够处理 JBoss EAP 的事务管理器和连接工厂。以下代码示例演示了如何使用 CDI 配置事务 JMS Camel 路由。

camel-jms 组件需要一个类型为 **org.springframework.transaction.PlatformTransactionManager** 的事务管理器。因此，您首先创建一个 bean 扩展 **JtaTransactionManager**。请注意，bean 被标注为 **@Named**，以允许在 Camel bean registry 中注册 bean。另请注意，JBoss EAP 事务管理器和用户事务实例使用 CDI 注入。

```
@Named("transactionManager")
public class CdiTransactionManager extends JtaTransactionManager {
```

```
  @Resource(mappedName = "java:/TransactionManager")
  private TransactionManager transactionManager;

  @Resource
  private UserTransaction userTransaction;

  @PostConstruct
  public void initTransactionManager() {
    setTransactionManager(transactionManager);
    setUserTransaction(userTransaction);
  }
}
```

接下来，您需要声明要使用的事务策略。同样，使用 **@Named** 注释使 bean 可用于 Camel。事务管理器也会注入，以便使用所需的事务策略创建 TransactionTemplate。此实例中 PROPAGATION_REQUIRED。

```
@Named("PROPAGATION_REQUIRED")
public class CdiRequiredPolicy extends SpringTransactionPolicy {
  @Inject
  public CdiRequiredPolicy(CdiTransactionManager cdiTransactionManager) {
    super(new TransactionTemplate(cdiTransactionManager,
      new DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRED)));
  }
}
```

现在，您可以配置 Camel RouteBuilder 类，并注入 Camel JMS 组件所需的依赖项。JBoss EAP XA 连接工厂与之前配置的事务管理器一起注入。

在本例中，当任何消息都从 queue1 使用时，它们都会路由到另一个名为 queue2 的 JMS 队列。从 queue2 使用的消息会导致 JMS 事务通过 rollback （）DSL 方法回滚。这将使原始消息放置在死信队列 (DLQ)上。

```
@Startup
@ApplicationScoped
@ContextName("jms-camel-context")
public class JMSRouteBuilder extends RouteBuilder {

  @Resource(mappedName = "java:/JmsXA")
  private ConnectionFactory connectionFactory;

  @Inject
  CdiTransactionManager transactionManager;

  @Override
  public void configure() throws Exception {
    // Creates a JMS component which supports transactions
    JmsComponent jmsComponent = JmsComponent.jmsComponentTransacted(connectionFactory,
transactionManager);
    getContext().addComponent("jms", jmsComponent);

    from("jms:queue:queue1")
      .transacted("PROPAGATION_REQUIRED")
      .to("jms:queue:queue2");
```

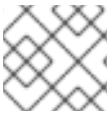
```
    // Force the transaction to roll back. The message will end up on the Wildfly 'DLQ' message queue
    from("jms:queue:queue2")
      .to("log:end")
      .rollback();
  }
```

5.6.2.4. 远程 JMS 目的地

一个 JBoss EAP 实例可以通过 远程 JNDI 将消息发送到另一个 JBoss EAP 实例上配置的 ActiveMQ Artemis 目的地。

需要一些额外的 JBoss EAP 配置才能实现此目的。首先配置一个导出的 JMS 队列。

只有 **java:jboss/exported** 命名空间中绑定的 JNDI 名称才被视为远程客户端的候选者，因此队列被正确命名。



注意

您必须在 JBoss EAP 客户端 *应用服务器和*JBoss EAP 远程服务器上配置队列。

```
<jms-queue name="RemoteQueue">
  <entry name="java:jboss/exported/jms/queues/RemoteQueue"/>
</jms-queue>
```

在客户端可以连接到远程服务器之前，需要配置用户访问凭据。在远程服务器上，运行 add user 实用程序，以在 'guest' 组中创建新应用程序用户。本例具有名为 'admin' 的用户，密码是 'secret'。

RouteBuilder 实施与前面的示例不同。您需要配置 InitialContext 并从 JNDI weselves 检索连接工厂，而不是注入连接工厂。

configureInitialContext 方法会创建这个 InitialContext。请注意，您需要设置一个提供程序 URL，它应引用您的远程 JBoss EAP 实例主机名和端口号。本例使用 JBoss EAP JMS http-connector，但 此处 提供了备选方案。

最后，路由配置为每 10 秒发送 XML 有效负载到之前配置的远程目的地 - 'RemoteQueue'。

```
@Override
public void configure() throws Exception {
  Context initialContext = configureInitialContext();
  ConnectionFactory connectionFactory = (ConnectionFactory)
initialContext.lookup("java:jms/RemoteConnectionFactory");

  JmsComponent component = new JmsComponent();
  component.setConnectionFactory(connectionFactory);

  getContext().addComponent("jms", component);

  from("timer://foo?fixedRate=true&period=10000")
  .transform(constant("<?xml version='1.0><message><greeting>hello world</greeting>
</message>"))
  .to("jms:queue:RemoteQueue?username=admin&password=secret")
  .to("log:jms?showAll=true");
}
```

```
private Context configureInitialContext() throws NamingException {
  final Properties env = new Properties();
  env.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory");
  env.put(Context.PROVIDER_URL, System.getProperty(Context.PROVIDER_URL, "http-
remoting://my-remote-host:8080"));
  env.put(Context.SECURITY_PRINCIPAL, System.getProperty("username", "admin"));
  env.put(Context.SECURITY_CREDENTIALS, System.getProperty("password", "secret"));
  return new InitialContext(env);
}
```

5.6.3. 安全性

请参阅 JMS 安全性部分。

5.6.4. EAP 上的 Fuse 中的快速入门示例

快速入门示例包括在 EAP 安装的 **quickstarts/camel/camel-jms** 目录的 Fuse 中。

5.7. JMX

您可以通过与 JBoss EAP JMX 子系统集成的 JMX 组件提供管理支持。

```
CamelContext camelctx = contextFactory.createWildflyCamelContext(getClass().getClassLoader());
camelctx.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      String host = InetAddress.getLocalHost().getHostName();
      from("jmx:platform?format=raw&objectDomain=org.apache.camel&key.context=" + host +
"/system-context-1&key.type=routes&key.name=\"route1\"" +
      "&monitorType=counter&observedAttribute=ExchangesTotal&granularityPeriod=500").
      to("direct:end");
   }
});
camelctx.start();

ConsumerTemplate consumer = camelctx.createConsumerTemplate();
MonitorNotification notifcation = consumer.receiveBody("direct:end", MonitorNotification.class);
Assert.assertEquals("ExchangesTotal", notifcation.getObservedAttribute());
```

5.8. JNDI

JNDI 集成由 JBoss EAP 特定的 CamelContext 提供，如下所示：

```
InitialContext inictx = new InitialContext();
CamelContextFactory factory = inictx.lookup("java:jboss/camel/CamelContextFactory");
WildFlyCamelContext camelctx = factory.createCamelContext();
```

从 **WildFlyCamelContext**，您可以获得预配置的命名上下文

```
Context context = camelctx.getNamingContext();
context.bind("helloBean", new HelloBean());
```

然后可以从 Camel 路由引用。

```
camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start").beanRef("helloBean");
    }
});
camelctx.start();
```

5.9. JPA

JPA 集成由 Camel JPA 组件提供。您可以通过提供 persistence.xml 配置文件以及一些 JPA 注解的类来开发 Camel JPA 应用程序。

5.9.1. persistence.xml 示例

在以下示例中，您可以使用 JBoss EAP in-memory ExampleDS 数据源，该数据源在 JBoss EAP standalone.xml 配置文件中配置。

```
<persistence version="2.0"
    xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

    <persistence-unit name="camel">
        <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
        <class>org.wildfly.camel.test.jpa.model.Customer</class>
        <properties>
            <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>

</persistence>
```

5.9.2. JPA ENTitiy 示例

```
@Entity
@Table(name = "customer")
public class Customer implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    private String firstName;
    private String lastName;

    public Customer() {
    }

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
```

```
        this.lastName = lastName;
    }

    public Long getId() {
        return id;
    }

    public void setId(final Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

5.9.3. Camel JPA endpoint / route configuration

配置 JPA 后，您可以使用 CDI 将 EntityManager 和 UserTransaction 实例注入 RouteBuilder 类或测试案例中：

```
@PersistenceContext
EntityManager em;

@Inject
UserTransaction userTransaction;
```

现在，配置 Camel 路由和 JPA 端点：

```
WildFlyCamelContext camelctx = contextFactory.createCamelContext(getClass().getClassLoader());

EntityManagerFactory entityManagerFactory = em.getEntityManagerFactory();

// Configure a transaction manager
JtaTransactionManager transactionManager = new JtaTransactionManager();
transactionManager.setUserTransaction(userTransaction);
transactionManager.afterPropertiesSet();

// Configure the JPA endpoint to use the correct EntityManagerFactory and JtaTransactionManager
final JpaEndpoint jpaEndpoint = new JpaEndpoint();
jpaEndpoint.setCamelContext(camelctx);
jpaEndpoint.setEntityType(Customer.class);
jpaEndpoint.setEntityManagerFactory(entityManagerFactory);
```

```
jpaEndpoint.setTransactionManager(transactionManager);

camelctx.addRoutes(new RouteBuilder() {
@Override
public void configure() throws Exception {
    from("direct:start")
    .to(jpaEndpoint);
}
});

camelctx.start();
```

最后，您可以将 'Customer' 实体发送到 'direct:start' 端点，然后查询 ExampleDS 数据源来验证已保存记录。

```
Customer customer = new Customer("John", "Doe");
ProducerTemplate producer = camelctx.createProducerTemplate();
producer.sendBody("direct:start", customer);

// Query the in memory database customer table to verify that a record was saved
CriteriaBuilder criteriaBuilder = em.getCriteriaBuilder();
CriteriaQuery<Long> query = criteriaBuilder.createQuery(Long.class);
query.select(criteriaBuilder.count(query.from(Customer.class)));

long recordCount = em.createQuery(query).getSingleResult();

Assert.assertEquals(1L, recordCount);
```

第 6 章 CAMEL 组件

本章有关支持的 camel 组件的详情

6.1. CAMEL-ACTIVEMQ

Camel ActiveMQ 集成由 activemq 组件提供。

组件可以配置为使用嵌入式或外部代理。对于 Wildfly / EAP 容器管理的连接池和 XA-Transaction 支持，可以将 ActiveMQ 资源适配器 配置到容器配置文件中。

6.1.1. JBoss EAP ActiveMQ 资源适配器配置

下载 ActiveMQ 资源适配器 rar 文件。下列步骤概述了配置 ActiveMQ 资源适配器。

1. 停止您的 JBoss EAP 实例。
2. 下载资源适配器并复制到相关的 JBoss EAP 部署目录。对于独立模式：

```
cp activemq-rar-5.11.1.rar ${JBOSS_HOME}/standalone/deployments/activemq-rar.rar
```

3. 为 ActiveMQ 适配器配置 JBoss EAP 资源适配器子系统。

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:2.0">
    <resource-adapters>
        <resource-adapter id="activemq-rar.rar">
            <archive>
                activemq-rar.rar
            </archive>
            <transaction-support>XATransaction</transaction-support>
            <config-property name="UseInboundSession">
                false
            </config-property>
            <config-property name="Password">
                defaultPassword
            </config-property>
            <config-property name="UserName">
                defaultUser
            </config-property>
            <config-property name="ServerUrl">
                tcp://localhost:61616?jms.rmIdFromConnectionId=true
            </config-property>
            <connection-definitions>
                <connection-definition class-
name="org.apache.activemq.ra.ActiveMQManagedConnectionFactory" jndi-
name="java:/ActiveMQConnectionFactory" enabled="true" pool-name="ConnectionFactory">
                    <xa-pool>
                        <min-pool-size>1</min-pool-size>
                        <max-pool-size>20</max-pool-size>
                        <prefill>false</prefill>
                        <is-same-rm-override>false</is-same-rm-override>
                    </xa-pool>
                </connection-definition>
            </connection-definitions>
```

```
            <admin-objects>
               <admin-object class-name="org.apache.activemq.command.ActiveMQQueue" jndi-
name="java:/queue/HELLOWORLDMDBQueue" use-java-context="true" pool-
name="HELLOWORLDMDBQueue">
                  <config-property name="PhysicalName">
                     HELLOWORLDMDBQueue
                  </config-property>
               </admin-object>
               <admin-object class-name="org.apache.activemq.command.ActiveMQTopic" jndi-
name="java:/topic/HELLOWORLDMDBTopic" use-java-context="true" pool-
name="HELLOWORLDMDBTopic">
                  <config-property name="PhysicalName">
                     HELLOWORLDMDBTopic
                  </config-property>
               </admin-object>
            </admin-objects>
         </resource-adapter>
      </resource-adapters>
   </subsystem>
```

如果您的资源适配器存档文件名与 activemq-rar.rar.rar 不同，您必须更改上述配置中的 archive 元素的内容，以匹配您的存档文件名称。

必须选择 UserName 和 Password 配置属性的值，以匹配外部代理中有效用户的凭证。

您可能需要更改 ServerUrl 配置 属性的值，以匹配外部代理公开的实际主机名和端口。

4） 启动 JBoss EAP。如果一切配置正确，您应该在 JBoss EAP server.log 中查看消息，如下所示：

```
13:16:08,412 INFO  [org.jboss.as.connector.deployment] (MSC service thread 1-5) JBAS010406:
Registered connection factory java:/AMQConnectionFactory`
```

6.1.2. Camel 路由配置

以下 ActiveMQ producer 和消费者示例使用 ActiveMQ 嵌入式代理和"vm"传输（避免对外部 ActiveMQ 代理的需求）。

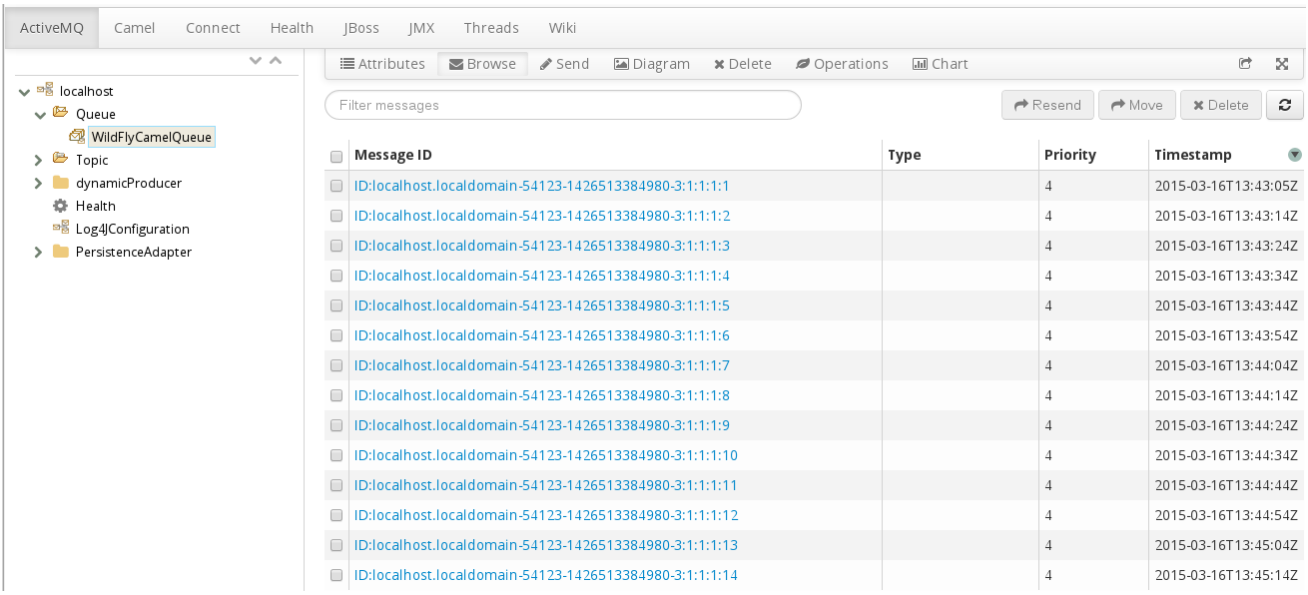
示例将 CDI 与 camel-cdi 组件一起使用。JMS ConnectionFactory 实例通过 JNDI 查找注入到 Camel RouteBuilder 中。

6.1.2.1. ActiveMQ Producer

```
@Startup
@ApplicationScoped
@ContextName("activemq-camel-context")
public class ActiveMQRouteBuilder extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("timer://sendJMSMessage?fixedRate=true&period=10000")
    .transform(constant("<?xml version='1.0><message><greeting>hello world</greeting>
</message>"))
    .to("activemq:queue:WildFlyCamelQueue?brokerURL=vm://localhost")
```

```
    .log("JMS Message sent");
  }
}
```

每次将消息添加到 WildFlyCamelQueue 目的地时，都会向控制台输出一条日志消息。要验证消息实际上被放入队列中，您可以使用 Camel on EAP 子系统提供的 ../features/hawtio.md[Hawtio 控制台,window=_blank]。



6.1.2.2. ActiveMQ Consumer

要使用 ActiveMQ 消息，Camel RouteBuilder 实施与制作者示例类似。

当 ActiveMQ 端点消耗来自 WildFlyCamelQueue 目的地的消息时，内容会记录到控制台。

```
@Override
public void configure() throws Exception {
  from("activemq:queue:WildFlyCamelQueue?brokerURL=vm://localhost")
  .to("log:jms?showAll=true");
}
```

6.1.2.3. ActiveMQ Transactions

6.1.2.3.1. ActiveMQ 资源适配器配置

需要 ActiveMQ 资源适配器来利用 XA 事务支持、连接池等。

以下 XML 片段显示了如何在 JBoss EAP 服务器 XML 配置中配置资源适配器。请注意，**ServerURL** 设置为使用嵌入式代理。连接工厂绑定到 JNDI 名称 **java:/ActiveMQConnectionFactory**。这将在下面的 RouteBuilder 示例中查找。

最后，两个队列配置为"queue1"和"queue2"。

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:2.0">
  <resource-adapters>
    <resource-adapter id="activemq-rar.rar">
      ...
      <admin-objects>
```

```
        <admin-object class-name="org.apache.activemq.command.ActiveMQQueue" jndi-
name="java:/queue/queue1" use-java-context="true" pool-name="queue1pool">
          <config-property name="PhysicalName">queue1</config-property>
        </admin-object>
        <admin-object class-name="org.apache.activemq.command.ActiveMQQueue" jndi-
name="java:/queue/queue2" use-java-context="true" pool-name="queue2pool">
          <config-property name="PhysicalName">queue2</config-property>
        </admin-object>
      </admin-objects>
    </resource-adapter>
  </resource-adapters>
</subsystem>
```

6.1.2.4. 事务管理器

camel-activemq 组件需要类型为 **org.springframework.transaction.PlatformTransactionManager** 的事务管理器。因此，您可以先创建一个可满足此要求的 bean 扩展 **JtaTransactionManager**。请注意，bean 被标注为 **@Named**，以允许在 Camel bean registry 中注册 bean。另请注意，JBoss EAP 事务管理器和用户事务实例使用 CDI 注入。

```
@Named("transactionManager")
public class CdiTransactionManager extends JtaTransactionManager {

  @Resource(mappedName = "java:/TransactionManager")
  private TransactionManager transactionManager;

  @Resource
  private UserTransaction userTransaction;

  @PostConstruct
  public void initTransactionManager() {
    setTransactionManager(transactionManager);
    setUserTransaction(userTransaction);
  }
}
```

6.1.2.5. 事务策略

接下来，您需要声明要使用的事务策略。同样，使用 **@Named** 注释使 bean 可用于 Camel。事务管理器也会注入，以便使用所需的事务策略创建 **TransactionTemplate**。此实例中 **PROPAGATION_REQUIRED**。

```
@Named("PROPAGATION_REQUIRED")
public class CdiRequiredPolicy extends SpringTransactionPolicy {
  @Inject
  public CdiRequiredPolicy(CdiTransactionManager cdiTransactionManager) {
    super(new TransactionTemplate(cdiTransactionManager,
      new DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRED)));
  }
}
```

6.1.2.6. 路由构建器

现在，您可以配置 Camel RouteBuilder 类，并注入 Camel ActiveMQ 组件所需的依赖项。在资源适配器配置上配置的 ActiveMQ 连接工厂会与您之前配置的事务管理器一起注入。

在本例中，当任何消息都从 queue1 使用时，它们都会路由到另一个名为 queue2 的 JMS 队列。从 queue2 使用的消息会导致 JMS 事务通过 rollback （）DSL 方法回滚。这将使原始消息放置在死信队列 (DLQ)上。

```
@Startup
@ApplicationScoped
@ContextName("activemq-camel-context")
public class ActiveMQRouteBuilder extends RouteBuilder {

  @Resource(mappedName = "java:/ActiveMQConnectionFactory")
  private ConnectionFactory connectionFactory;

  @Inject
  private CdiTransactionManager transactionManager;

  @Override
  public void configure() throws Exception {
    ActiveMQComponent activeMQComponent = ActiveMQComponent.activeMQComponent();
    activeMQComponent.setTransacted(false);
    activeMQComponent.setConnectionFactory(connectionFactory);
    activeMQComponent.setTransactionManager(transactionManager);

    getContext().addComponent("activemq", activeMQComponent);

      errorHandler(deadLetterChannel("activemq:queue:ActiveMQ.DLQ")
      .useOriginalMessage()
      .maximumRedeliveries(0)
      .redeliveryDelay(1000));

    from("activemq:queue:queue1F")
      .transacted("PROPAGATION_REQUIRED")
      .to("activemq:queue:queue2");

    from("activemq:queue:queue2")
      .to("log:end")
      .rollback();
  }
}
```

6.1.3. 安全性

请参阅 JMS 安全性部分。

6.1.4. GitHub 上的代码示例

GitHub 上提供了一个 camel-activemq 应用 示例。

6.2. CAMEL-JMS

将 camel-jms、camel-sjms 和 camel-sjms2 端点连接到远程 AMQ 7 代理的方法有两种。

1. 使用 pooled-connection-factory 配置 remote-connector，如 JBoss EAP 配置消息传递指南中的 配置 Artemis 资源适配器 以连接到 Red Hat JBoss AMQ 7 部分。
2. 使用 connection-factory 配置 remote-connector，如下所述 使用 connection-factory 配置一个 remote-connector

第一个选项是首选的方法，因为它提供连接池和 XA 事务支持。

对于使用持久订阅者的消息传递场景，JBoss EAP 上的 Fuse 7.13 不支持 pooled-connection-factory 的消息传递场景，因为 deploymentutils 7 规格进行了限制。在这些情况下，首选配置一个标准的未池 connection-factory。

使用 connection-factory 配置一个 remote-connector

1. 创建一个指向远程消息传递服务器的 outbound-socket-binding：

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=messaging-remote-throughput:add(host=localhost, port=61616)
```

2. 创建一个 remote-connector，引用在第 1 步中创建的 outbound-socket-binding。

```
/subsystem=messaging-activemq/server=default/remote-connector=netty-remote-throughput:add(socket-binding=messaging-remote-throughput)
```

3. 创建一个 connection-factory 引用在第 2 步中创建的 remote-connector。

```
/subsystem=messaging-activemq/server=default/connection-factory=simple-remote-artemis-connection-factory:add(entries=[java:/jms/RemoteJms],connectors=[netty-remote-throughput])
```

6.2.1. 消息传递代理和客户端

摘要

Fuse 7.13 不附带默认的内部消息传递代理，但设计为与外部 JMS 代理接口。

JBoss EAP 上的 Fuse 7.13 使用 JBoss EAP 上配置消息传递 中详述的资源适配器，以访问外部消息传递代理。

有关 JBoss EAP 上 Fuse 7.13 上可用于消息传递的外部代理、JCA 适配器和 Camel 组件组合的更多信息，请参阅 支持的配置。

有关使用 JMS 在 JBoss EAP 上使用 Fuse 连接到外部代理的更多信息，请参阅 第 6.2 节 “camel-jms”。

camel-jms quickstart

提供了一个快速入门，用于演示将 camel-jms 组件与 JBoss EAP 上的 Fuse 搭配使用，以生成和使用 JMS 消息。

在本快速入门中，Camel 路由使用来自 **EAP_HOME/standalone/data/orders** 的文件，并将其内容放在名为 **OrdersQueue** 的内存中 ActiveMQ Artemis 队列中。然后，另一个 Camel 路由会消耗 **OrdersQueue** 的内容，并将顺序排序到 **EAP_HOME/standalone/data/orders/processed** 中的独立国家目录中。

CLI 命令创建和删除 **OrdersQueue** CLI 脚本时，负责在应用部署和取消部署时为您创建和删除 JMS **OrdersQueue**。这些脚本位于 **EAP_HOME/quickstarts/camel-jms/src/main/resources/cli** 目录中。

先决条件

要运行此快速入门，您必须有 Fuse 7.13 的工作版本

您还必须按照使用 JBoss AMQ 进行远程 JMS 通信 中的说明来连接到外部 AMQ 7 代理。然后，您可以使用默认连接工厂注入连接工厂。

```
@Resource(mappedName = "java:jboss/RemoteJmsXA")
ConnectionFactory connectionFactory;
```

设**置快速入**门

1. 以单机模式启动 JBOSS EAP。
2. 导航到 **EAP_HOME/quickstarts/camel/camel-jms**
3. 输入 **mvn clean install -Pdeploy'** 以构建和部署快速入门。
4. 浏览 http://localhost:8080/example-camel-jms

您应该会看到标题为 'Orders Received' 的页面。当我们向示例应用程序发送订单时，本页中将列出每个国家的订购列表。

运行快速启动

EAP_HOME/quickstarts/camel/camel-jms/src/main/resources 目录中有一些示例顺序 XML 文件。Camel 将每 5 秒随机选择一个文件，并将其复制到 **EAP_HOME/standalone/data/orders** 中以进行处理。

控制台将输出详细说明每个顺序所发生的消息。输出将如下所示：

```
JmsConsumer[OrdersQueue]) Sending order to the UK
JmsConsumer[OrdersQueue]) Sending order to another country
JmsConsumer[OrdersQueue]) Sending order to the US
```

使用这些文件后，您可以返回到 http://localhost:8080/example-camel-jms/orders。每个国家/地区接收的订购数应该已增加 1。

所有已处理的顺序将分成以下目的地：

```
EAP_HOME/standalone/data/orders/processed/uk
EAP_HOME/standalone/data/orders/processed/us
EAP_HOME/standalone/data/orders/processed/other
```

取消部署

要取消部署示例，请导航到 **EAP_HOME/quickstarts/camel/camel-jms** 运行 **mvn clean -Pdeploy**。

6.3. CAMEL-MAIL

与电子邮件交互由 邮件 组件提供。

默认情况下，Camel 将创建自己的邮件会话，并使用它来与您的邮件服务器交互。由于 JBoss EAP 已经提供了一个 mail 子系统，其中包含对安全连接、用户名和密码加密等所有相关支持，因此建议在 JBoss EAP 配置中配置您的邮件会话，并使用 JNDI 将它们连接到您的 Camel 端点。

6.3.1. JBoss EAP 配置

首先，您要为邮件服务器配置 JBoss EAP 邮件子系统。以下示例添加了 Google Mail IMAP 和 SMTP 的配置。

在 'default' 会话后配置额外的 mail-session。

```
<subsystem xmlns="urn:jboss:domain:mail:2.0">
    <mail-session name="default" jndi-name="java:jboss/mail/Default">
      <smtp-server outbound-socket-binding-ref="mail-smtp"/>
    </mail-session>
```

```
    <mail-session debug="true" name="gmail" jndi-name="java:jboss/mail/gmail">
      <smtp-server outbound-socket-binding-ref="mail-gmail-smtp" ssl="true" username="your-username-here" password="your-password-here"/>
      <imap-server outbound-socket-binding-ref="mail-gmail-imap" ssl="true" username="your-username-here" password="your-password-here"/>
    </mail-session>
</subsystem>
```

您可以配置 'mail-gmail-smtp' 和 'mail-gmail-imap' 的 **outbound-socket-binding-ref** 值。

下一步是配置这些套接字绑定。您可以根据以下内容向 **socket-binding-group** 配置添加额外的绑定：

```
<outbound-socket-binding name="mail-gmail-smtp">
  <remote-destination host="smtp.gmail.com" port="465"/>
</outbound-socket-binding>

<outbound-socket-binding name="mail-gmail-imap">
  <remote-destination host="imap.gmail.com" port="993"/>
</outbound-socket-binding>
```

这会将邮件会话配置为连接到端口 465 上的 smtp.gmail.com 和端口 993 上的 imap.gmail.com。如果您使用其他邮件主机，则此详细信息将有所不同。

6.3.2. POP3 配置

如果您需要配置 POP3 会话，原则与上例中定义的原则相同。

```
<!-- Server configuration -->
<pop3-server outbound-socket-binding-ref="mail-pop3" ssl="true" username="your-username-here" password="your-password-here"/>

<!-- Socket binding configuration -->
<outbound-socket-binding name="mail-gmail-imap">
  <remote-destination host="pop3.gmail.com" port="993"/>
</outbound-socket-binding>
```

6.3.3. Camel 路由配置

6.3.3.1. 邮件制作者

这个示例使用 SMTPS 协议，以及 CDI 与 camel-cdi 组件。您在 JBoss EAP 配置中配置的 Java 邮件会话通过 JNDI 注入到 Camel RouteBuilder 中。

6.3.3.1.1. 路由构建器 SMTPS 示例

GMail 邮件会话使用 **@Resource** 注释注入 Producer 类，并引用您之前配置的 **jndi-name** 属性。这可让您引用 camel-mail 端点配置上的邮件会话。

```
public class MailSessionProducer {
    @Resource(lookup = "java:jboss/mail/greenmail")
    private Session mailSession;

    @Produces
    @Named
    public Session getMailSession() {
        return mailSession;
    }
}
```

```
public class MailRouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("direct:start")
        .to("smtps://smtp.gmail.com?session=#mailSession");
    }
}
```

要发送电子邮件，您可以创建一个 ProducerTemplate，并将适当的正文与所需的电子邮件标头一起发送。

```
Map<String, Object> headers = new HashMap<String, Object>();
headers.put("To", "destination@test.com");
headers.put("From", "sender@example.com");
headers.put("Subject", "Camel on Wildfly rocks");

String body = "Hi,\n\nCamel on Wildfly rocks!.";

ProducerTemplate template = camelContext.createProducerTemplate();
template.sendBodyAndHeaders("direct:start", body, headers);
```

6.3.3.2. 邮件消费者

要接收电子邮件，您可以使用 IMAP MailEndpoint。Camel 路由配置类似如下：

```
public void configure() throws Exception {
   from("imaps://imap.gmail.com?session=#mailSession")
   .to("log:email");
}
```

6.3.4. 安全性

6.3.4.1. SSL 配置

JBoss EAP 可以配置为使用 SSL / TLS 管理 Java 邮件会话及其关联的传输。在配置邮件会话时，您可以在服务器类型上配置 SSL 或 TLS：

- smtp-server
- imap-server
- pop-server

通过设置属性 **ssl="true"** 或 **tls="true"**。

6.3.4.2. 保护密码

建议不要在配置文件中使用明文密码。您可以使用 WildFly Vault 屏蔽敏感数据。

6.3.4.3. Camel 安全

Camel 端点安全文档可在 邮件 组件指南中找到。Camel 还有一个 安全摘要页面。

6.3.5. GitHub 上的代码示例

GitHub 上提供了一个 camel-mail 应用程序示例，供您尝试发送 / 接收电子邮件。

6.4. CAMEL-REST

通过 其余 组件，您可以使用 Rest DSL 和插件到其他 Camel 组件作为 REST 传输来定义 REST 端点。

注意

EAP 子系统上的 Camel 只支持 camel-servlet 和 camel-undertow 组件以用于 REST DSL。但是，如果尝试配置其他组件，则子系统不起作用。

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        restConfiguration().component("servlet").contextPath("camel/rest").port(8080);
        rest("/hello").get("/{name}").to("direct:hello");
        from("direct:hello").transform(simple("Hello ${header.name}"));
    }
});
```

6.5. CAMEL-REST-SWAGGER

rest-swagger 组件可以从 Swagger 文档配置 REST 生成者，并委派给实施 **RestProducerFactory** 接口的组件，例如：

- camel-http4
- camel-undertow

6.6. CAMEL-SQL

SQL 组件允许您使用 JDBC 查询处理数据库。此组件和 JDBC 组件之间的区别在于，如果 SQL 查询是端点的属性，它将消息有效负载用作传递给查询的参数。

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("sql:select name from information_schema.users?
dataSource=java:jboss/datasources/ExampleDS")
        .to("direct:end");
    }
});
```

注意

上面显示的 JNDI 数据源查找仅在配置 **DefaultCamelContext** 时正常工作。请参阅以下适用于 **CdiCamelContext** 和 **SpringCamelContext** 示例。

与 camel-cdi 组件一起使用时，Jarkarta EE 注解可以为 Camel 提供数据源。本例使用 **@Named** 注释，以便 Camel 能够发现所需的数据源。

```
public class DatasourceProducer {
    @Resource(lookup = "java:jboss/datasources/ExampleDS")
    DataSource dataSource;

    @Produces
    @Named("wildFlyExampleDS")
    public DataSource getDataSource() {
        return dataSource;
    }
}
```

现在，可以通过 camel-sql 端点配置上的 **dataSource** 参数来引用数据源。

```
@ApplicationScoped
@ContextName("camel-sql-cdi-context")
@Startup
public class CdiRouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("sql:select name from information_schema.users?dataSource=wildFlyExampleDS")
        .to("direct:end");
    }
}
```

使用 camel-spring 时，路由配置类似如下：

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xsi:schemaLocation="
```

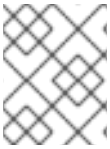
```
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
      http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd
      http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-
jee.xsd">

   <jee:jndi-lookup id="wildFlyExampleDS" jndi-name="java:jboss/datasources/ExampleDS"/>

   <camelContext id="sql-spring-context" xmlns="http://camel.apache.org/schema/spring">
      <route>
         <from uri="sql:select name from information_schema.users?dataSource=#wildFlyExampleDS"
/>
         <to uri="direct:end" />
      </route>
   </camelContext>

</beans>
```

6.6.1. Spring JDBC XML 命名空间支持

支持以下 Spring JDBC XML 配置

jdbc:embedded-database

```
<jdbc:embedded-database id="datasource" type="H2">
  <jdbc:script location="db-schema.sql"/>
</jdbc:embedded-database>
```



注意

默认情况下，仅支持 H2 数据库，因为 JBoss EAP 具有原生支持。如果要使用其他嵌入式数据库供应商，您需要安装适当的数据库驱动程序。

jdbc:initialize-database

```
<jdbc:initialize-database data-source="datasource">
  <jdbc:script location="classpath:db-init.sql"/>
</jdbc:initialize-database>
```

6.7. CAMEL-SOAP-REST-BRIDGE

简单的 Camel 路由可以将 REST 调用桥接到传统的 SOAP 服务。提供了快速入门示例，以演示将 **camel-soap-rest-bridge** 组件与 Camel 的 REST DSL 搭配使用，以公开后端 SOAP API 服务。

在这个快速入门中，安全性同时涉及 REST 端点和 SOAP 端点，它们都由 RH SSO 支持。frontend REST API 通过 OAuth 和 OpenID Connect 保护，客户端将使用"Resource Owner Password Credentials" OAuth2 模式从 RH SSO 获取 JWT (JSON Web Token)访问令牌。客户端将使用此令牌来访问 REST 端点。

在 bridge Camel 路由中，客户端身份从 SecurityContext 传播，当 **camel-cxf producer** 与后端 WS-SECURITY 保护的 SOAP 服务通信时，它将使用此客户端身份来获取 CXF STS 服务发布的 SAML2 令牌（由 RH SSO 作为身份提供程序支持）。SAML2 令牌已签名并添加到 WS-SECURITY 标头中，后端 WS-SECURITY 保护 SOAP 服务将验证此 SAML2 令牌。

SOAP 调用也包含 XSD 架构验证。如果令牌验证成功，后端 SOAP 服务会返回发起请求的 REST 客户端的响应。

先决条件

1. 已安装 JBoss EAP 7.3 或更高版本。
2. 已安装 Apache Maven 3.3.x 或更高版本。
3. 已安装并配置了 RH SSO 7.4 - 按照安装说明 https://access.redhat.com/documentation/zh-cn/red_hat_single_sign-on/7.4/html/getting_started_guide/installing-standalone_#installing-server-product
4. 已安装 RH SSO EAP 适配器 - 按照 https://access.redhat.com/documentation/zh-cn/red_hat_single_sign-on/7.4/html/getting_started_guide/securing-sample-app_#installing-client-adapter的安装说明操作

设**置快速**启动

1. 以单机模式启动 JBOSS EAP。
2. 导航到 **EAP_HOME/quickstarts/camel/camel-soap-rest-bridge**
3. 输入 **mvn clean install -Pdeploy** 以构建和部署快速入门。
4. 配置 RH SSO
   a. 从 http://localhost:8180/auth 登录 RH SSO 管理控制台，使用 admin/admin 作为用户名/密码
   b. 点 **Add realm**
   c. 点 **Select file**
   d. 在本示例文件夹中选择 ./src/main/resources/keycloak-config/realm-export-new.json，这会在此示例中导入预定义的必要的 realm/client/user/role
   e. 点 **Create**

EAP 上的 Fuse 中的快速入门示例

此快速入门示例包含关于在 **EAP_HOME/quickstarts/camel/camel-soap-rest-bridge** 目录的 Fuse 上安装中运行 Quickstart 和 test 案例结果的额外信息。

取消部署

要取消部署示例，请导航到 **EAP_HOME/quickstarts/camel/camel-soap-rest-bridge** 目录，并运行 **mvn clean -Pdeploy**。

6.8. 添加组件

可轻松添加对额外 Camel 组件的支持

添加 modules.xml 定义
modules.xml 描述符定义组件的类加载行为。它应当与组件的 jar 放在 **modules/system/layers/fuse/org/apache/camel/component** 中。模块依赖项应设置为直接编译时间依赖项。

以下是 camel-ftp 组件的示例

```
<module xmlns="urn:jboss:module:1.1" name="org.apache.camel.component.ftp">
  <resources>
    <resource-root path="camel-ftp-2.14.0.jar" />
  </resources>
  <dependencies>
    <module name="com.jcraft.jsch" />
    <module name="javax.xml.bind.api" />
    <module name="org.apache.camel.core" />
    <module name="org.apache.commons.net" />
  </dependencies>
</module>
```

请确保您不会重复已在 WildFly 中提供的模块，并可重复使用。

添加对组件的引用
要使此模块默认可见到任意>=< 部署，请添加对
modules/system/layers/fuse/org/apache/camel/component/main/module.xml的引用

```
<module xmlns="urn:jboss:module:1.3" name="org.apache.camel.component">
  <dependencies>
    ...
    <module name="org.apache.camel.component.ftp" export="true" services="export"/>
  </dependencies>
</module>
```

第 7 章 安全性

JBoss EAP 中的安全性是一个巨大的主题。JBoss EAP 和 Camel 都经过充分记录，用于保护配置、端点和有效负载的标准化方法。

7.1. HAWTIO 安全

要保护 HawtIO 控制台，请执行以下操作：

1. 在 standalone.xml 中添加系统属性

```
<system-properties>
    <property name="hawtio.authenticationEnabled" value="true" />
    <property name="hawtio.realm" value="hawtio-domain" />
</system-properties>
```

2. 在 security 子系统中为 HawtIO 添加一个安全域

```
<security-domain name="hawtio-domain" cache-type="default">
    <authentication>
        <login-module code="RealmDirect" flag="required">
            <module-option name="realm" value="ManagementRealm"/>
        </login-module>
    </authentication>
</security-domain>
```

3. 配置管理用户

```
$JBOSS_HOME/bin/add-user.sh -u someuser -p s3cret
```

4. 浏览 http://localhost:8080/hawtio，并使用为管理用户配置的凭据进行身份验证。

注意

如果您在 EAP 上运行 Fuse，并且您使用的是 Keycloak，则不支持 Elytron 安全性。

提示

如果您在 JDK 17 上的 EAP 上运行 Fuse，则 Keycloak 保护并出现日志记录到 HawtIO 控制台的问题，降级到 JDK 8 或 11，并在 EAP 上禁用 Elytron。

7.2. JAX-RS 安全性

以下主题解释了如何保护 JAX-RS 端点。

- WildFly HTTP 基本身份验证
- Security Realms 和 SSL
- 保护 EJB

7.3. JAX-WS SECURITY

以下主题解释了如何保护 JAX-WS 端点。

- WildFly HTTP 基本身份验证
- WS-Security
- CXF 安全性
- Security Realms 和 SSL
- 保护 EJB

7.4. JMS SECURITY

以下主题解释了如何保护 JMS 端点。

- ActiveMQ Artemis 安全文档
- ActiveMQ Artemis 地址和 JMS 目的地的安全设置
- ActiveMQ Artemis 安全域配置
- ActiveMQ Security

此外，您可以使用 Camel 的 Route 策略与 JBoss EAP 安全系统集成。

7.5. 路由策略

Camel 支持 RoutePolicies 的概念，可用于与 JBoss EAP 安全系统集成。目前有两种安全集成的场景。

7.5.1. Camel 调用 Jarkarta EE

当 camel 路由调用到安全 Jarkarta EE 组件时，它充当客户端，并且必须提供与调用关联的适当凭证。

您可以使用 **ClientAuthorizationPolicy** 分离路由，如下所示：

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
   @Override
   public void configure() throws Exception {
      from("direct:start")
      .policy(new ClientAuthorizationPolicy())
      .to("ejb:java:module/AnnotatedSLSB?method=doSelected");
   }
});
```

这不作为 camel 消息处理的一部分，不执行任何身份验证和授权。相反，它会将与 Camel Exchange 附带的凭据与 EJB3 层的调用相关联。

调用消息消费者的客户端必须在 AUTHENTICATION 标头中提供适当的凭证，如下所示：

```
ProducerTemplate producer = camelctx.createProducerTemplate();
Subject subject = new Subject();
subject.getPrincipals().add(new DomainPrincipal(domain));
```

```
subject.getPrincipals().add(new EncodedUsernamePasswordPrincipal(username, password));
producer.requestBodyAndHeader("direct:start", "Kermit", Exchange.AUTHENTICATION, subject,
String.class);
```

身份验证和授权将在 Jarkarta EE 层中进行。

7.5.2. 保护 Camel 路由

为了保护 Camel 路由，您可以将 **DomainAuthorizationPolicy** 与路由关联。此策略需要针对给定安全域成功进行身份验证，并为 "Role2" 授权。

```
CamelContext camelctx = new DefaultCamelContext();
camelctx.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("direct:start")
        .policy(new DomainAuthorizationPolicy().roles("Role2"))
        .transform(body().prepend("Hello "));
    }
});
camelctx.start();
```

同样，调用消息消费者的客户端必须在 AUTHENTICATION 标头中提供适当的凭证，如下所示：

```
ProducerTemplate producer = camelctx.createProducerTemplate();
Subject subject = new Subject();
subject.getPrincipals().add(new DomainPrincipal(domain));
subject.getPrincipals().add(new EncodedUsernamePasswordPrincipal(username, password));
producer.requestBodyAndHeader("direct:start", "Kermit", Exchange.AUTHENTICATION, subject,
String.class);
```

7.6. 部署 CXF JAX-WS 快速入门

本例演示了在 EAP 上使用带有红帽 Fuse 的 camel-cxf 组件来生成和使用 Elytron 安全域保护的 JAX-WS Web 服务。从 EAP 7.1 开始，Elytron 是一个新的安全框架。在这个快速入门中，Camel 路由从直接端点获取消息有效负载，并将其传递给 CXF producer 端点。生产者使用有效负载将参数传递给通过 BASIC HTTP 身份验证保护的 CXF JAX-WS Web 服务。

先决条件

- 确保已安装并配置了 Maven。
- 确保安装和配置有红帽 Fuse 的应用服务器。

流程

1. 将 **JBOSS_HOME** 环境变量设置为指向应用服务器安装的根目录。
   - 对于 Linux

     ```
     export JBOSS_HOME=...
     ```

   - 对于 Windows:

```
set JBOSS_HOME=...
```

2. 使用 **add-user** 脚本创建新服务器应用用户和组。

   - 对于 Linux

     ```
     ${JBOSS_HOME}/bin/add-user.sh -a -u testUser -p testPassword1+ -g testRole
     ```

   - 对于 Windows:

     ```
     %JBOSS_HOME%\bin\add-user.bat -a -u testUser -p testPassword1+ -g testRole
     ```

3. 以单机模式启动应用服务器。

   - 对于 Linux

     ```
     ${JBOSS_HOME}/bin/standalone.sh -c standalone-full.xml
     ```

   - 对于 Windows:

     ```
     %JBOSS_HOME%\bin\standalone.bat -c standalone-full.xml
     ```

     此项目的 **webapp/WEB-INF** 目录中的 **jboss-web-xml** 和 **web.xml** 文件设置应用程序安全域、安全角色和限制。

4. 构建和部署项目。

   ```
   mvn install -Pdeploy
   ```

   此命令还会调用 CLI 脚本 **configure-basic-security.cli**，它将创建安全域和一些其他管理对象。

5. 浏览到 **http://localhost:8080/example-camel-cxf-jaxws-secure/**。
   此时会显示标题为 **Send A Greeting** 的页面。此 UI 可让您与测试 问候 Web 服务进行交互。服务 WSDL 位于 **http://localhost:8080/webservices/greeting-security-basic?wsdl**。

   有一个名为 **greet** 的单个服务操作，它取两个名为 **message** 和 **name** 的 String 参数。调用 Web 服务将返回这些值链接在一起的响应。

测试 **Camel Secure CXF JAX-WS 快速入门**

1. 浏览到 http://localhost:8080/example-camel-cxf-jaxws-secure/。

2. 在 **Send A Greeting** Web 表单上，在文本字段中输入 **消息和 名称**，然后按 发送 按钮。
   您输入的信息会显示在 UI 中以问候语显示。**CamelCxfWsServlet** 从 Web UI 处理 POST 请求。它从参数值检索消息和名称，并构造对象数组。此对象数组是发送到 **direct:start** 端点的消息有效负载。**ProducerTemplate** 将消息有效负载发送到 Camel。**direct:start** 端点将对象数组传递给 **cxf:bean** web service producer。 **CamelCxfWsServlet** 使用 Web 服务响应来显示 Web UI 上的问候语。您可以在 **src/main/webapp/WEB-INF/cxfws-security-camel-context.xml** 文件中看到完整的 Camel 路由。

取消部署快速入门

1. 运行以下命令来取消部署快速入门。

```
mvn clean -Pdeploy
```