Red Hat Fuse 7.13

部署到 Spring Boot

在独立模式中构建并运行 Spring Boot 应用程序

Last Updated: 2024-06-12

Red Hat Fuse 7.13 部署到 Spring Boot

在独立模式中构建并运行 Spring Boot 应用程序

法律通告

摘要

本指南介绍了如何构建作为 Jar 文件打包的 Spring Boot 应用程序，并直接以 JVM （单机模式）运行。

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 SPRING BOOT 独立入门

1.1. 关于 SPRING BOOT 独立部署模式

在单机部署模式中，Spring Boot 应用被打包为 Jar 文件，直接在 Java 虚拟机(JVM)中运行。此方法打包和运行应用与微服务理念一致，其中服务打包了最低要求。Spring Boot 应用可以使用 **java** 命令和 **-jar** 选项直接运行。例如：

```
java -jar SpringBootApplication.jar
```

其中 Spring Boot 为可执行 Jar 提供主类。在 Fuse 中构建 Spring Boot 独立应用程序需要以下元素：

- *Fuse Bill of Materials (BOM)* mvapich-进行了从 Red Hat Maven 存储库完全策展的依赖关系集合。BOM 利用 Maven *的依赖项管理机制* 来定义适当的 Maven 依赖项版本。
  注：红帽只支持 Fuse BOM 中定义的依赖项。
- *Spring Boot Maven 插件* iwl-criu 实现 Maven 中独立 Spring Boot 应用程序的构建过程。此插件负责将 Spring Boot 应用程序打包为可执行 Jar 文件。

1.2. 部署到 SPRING BOOT 2

在独立部署模式中，您可以选择部署到 Spring Boot 2 中。

注意

如需有关 OpenShift 部署模式的详细信息，请参阅 OpenShift 上的 Fuse 指南。

1.3. SPRING BOOT 2 的新 CAMEL 组件

Spring Boot 2 支持 Camel 版本 **2.23**，并支持以下一些新的 camel 组件：

Spring Boot 2 的新 Camel 组件

- as2-component
- aws-iam-component
- fhir-component
- google-calendar-stream-component
- google-mail-stream-component
- google-sheets-component
- google-sheets-stream-component
- ipfs-component
- kubernetes-hpa-component
- kubernetes-job-component
- micrometer-component

- mybatis-bean-component
- nsq-component
- rxjava2
- service-component
- spring-cloud-consul
- spring-cloud-zookeeper
- testcontainers-spring
- testcontainers
- web3j-component

第 2 章 使用 FUSE BOOSTERS

Red Hat Fuse 提供以下 boosters，以帮助您开始使用 Fuse 应用程序并演示有用的组件：

- 第 2.2 节 “构建并运行 Circuit Breaker booster” - 一个启用分布式应用程序的示例，可以应对对后端服务的网络连接和临时不可用中断。
- 第 2.3 节 “构建并运行 Externalized Configuration booster” - 有关如何为 Apache Camel 路由外部配置的示例。
- 第 2.4 节 “构建并运行 REST API booster” - 一个例子，它引入了使用 HTTP 协议与远程（由 Apache Camel 公开）服务交互的机制。

构建并运行 booster 演示的先决条件，安装以下先决条件：

- 支持的 Java Developer Kit (JDK)版本。详情请查看 支持的配置页面。
- Apache Maven 3.3.x 或更高版本.请参阅 Maven Download 页面。

2.1. 生成您的 BOOSTER 项目

Fuse booster 项目存在，可帮助开发人员开始使用运行独立应用程序。此处提供的说明指导您生成其中一个增强项目，即 Circuit Breaker booster。此练习演示了 Spring Boot 上 Fuse 的有用组件。

Netflix/Hystrix 断路器使分布式应用能够处理对后端服务的网络连接和临时不可用的中断。断路器模式的基本理念是，自动检测到依赖服务的丢失，如果后端服务暂时不可用，可以编程替代行为。

Fuse 断路器提升器由两个相关服务组成：

- 名称服务，将 **名称** 返回到 greet 的后端服务。
- 一个 **问候** 服务，即调用 **name** 服务的 frontend 服务以获取名称，然后返回字符串 **Hello, NAME**。

在此提升器演示中，Hystrix circuit breaker 在 **问候器** 服务和 **名称服务** 之间插入。如果 **后端名称** 服务不可用，则 **问候** 服务可能会回退到替代的行为并立即响应客户端，而不是在等待名称服务重启时被阻断。

先决条件

- **您必须有权访问 Red Hat Developer Platform。**
- **您必须有受支持的 Java Developer Kit (JDK)版本。详情请查看 支持的配置页面。**
- **您必须已安装并配置了 Apache Maven 3.3.x 或更高版本，如 在 本地设置 Maven所述。**

流程

1. 进入 https://developers.redhat.com/launch。

2. 单击 START。

   启动程序向导会提示您登录到您的红帽帐户。

3. 单击 登录或注册 按钮，然后登录。

4. 在 Launcher 页面上，单击 Deploy an Example Application 按钮。

5. 在 Create Example Application 页面上，在 Create Example Application 字段中输入 name fuse-circuit-breaker。

6. 点 Select an Example。

7. 在 Example 对话框中，选择 Circuit Breaker 选项。此时会出现 选择运行时 下拉菜单。

   a. 从 Select a Runtime 下拉菜单中选择 Fuse。

   b. 从版本下拉菜单中，选择 7.13 （红帽 Fuse） （不要选择 2.21.2 (Community) 版本）。

   c. 点击 Save。

8. 在 Create Example Application 页面中，点 Download。

9. 当您看到您的应用程序 是 Ready 对话框时，点 Download.zip。您的浏览器下载生成的 booster 项目（打包为 ZIP 文件）。

10. 使用存档实用程序将生成的项目提取到本地文件系统中的方便位置。

2.2. 构建并运行 CIRCUIT BREAKER BOOSTER

Netflix/Hystrix 断路器组件使分布式应用能够应对对后端服务的网络连接和临时不可用的中断。断路器模式的基本理念是，自动检测到依赖服务的丢失，如果后端服务暂时不可用，可以编程替代行为。

Fuse 断路器提升器由两个相关服务组成：

- *名称服务*，它将返回一个名称到 greet
- 一个 *问候服务*，它调用 name 服务以获取名称，然后返回字符串 Hello, NAME。

在演示中，Hystrix circuit breaker 在问候服务和名称服务之间插入。如果名称服务不可用，则问候服务可能会回退到替代的行为并立即响应客户端，而不是在等待名称服务重启时阻止或超时。

先决条件

- 您已完成了 第 2.1 节 “生成您的 booster 项目” 部分所述的步骤。

流程

按照以下步骤构建并运行 Circuit breaker booster 项目：

1. 打开 shell 提示符并使用 Maven 从命令行构建项目：

```
cd PROJECT_DIR
mvn clean package
```

2. 打开一个新的 shell 提示符并启动名称服务，如下所示：

```
cd name-service
mvn spring-boot:run -DskipTests -Dserver.port=8081
```

当 Spring Boot 启动时，您应该看到类似如下的输出：

```
...
```

```
2017-12-08 15:44:24.223  INFO 22758 --- [           main]
o.a.camel.spring.SpringCamelContext      : Total 1 routes, of which 1 are started
2017-12-08 15:44:24.227  INFO 22758 --- [           main]
o.a.camel.spring.SpringCamelContext      : Apache Camel 2.20.0 (CamelContext: camel-1)
started in 0.776 seconds
2017-12-08 15:44:24.234  INFO 22758 --- [           main]
org.jboss.fuse.boosters.cb.Application   : Started Application in 4.137 seconds (JVM running
for 4.744)
```

3. **打开一个新的 shell 提示符并启动问候服务，如下所示：**

```
cd greetings-service
mvn spring-boot:run -DskipTests
```

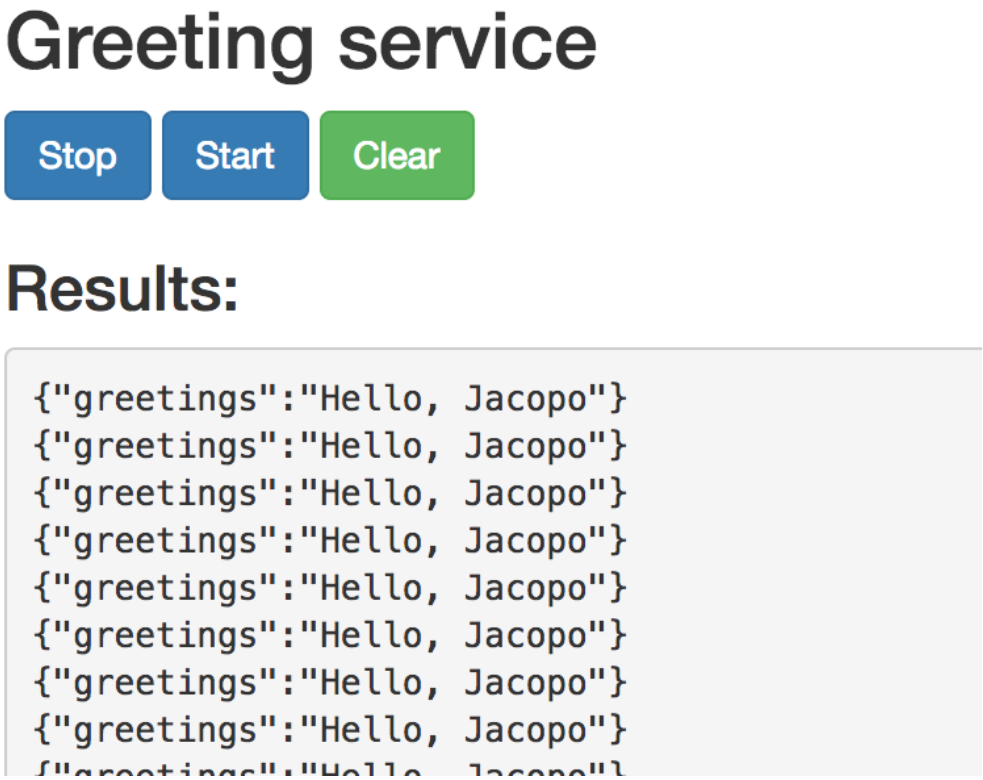
当 Spring Boot 启动时，您应该看到类似如下的输出：

```
...
2017-12-08 15:46:58.521  INFO 22887 --- [           main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2017-12-08 15:46:58.524  INFO 22887 --- [           main]
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2017-12-08 15:46:58.536  INFO 22887 --- [           main]
org.jboss.fuse.boosters.cb.Application   : Started Application in 6.263 seconds (JVM running
for 6.819)
```

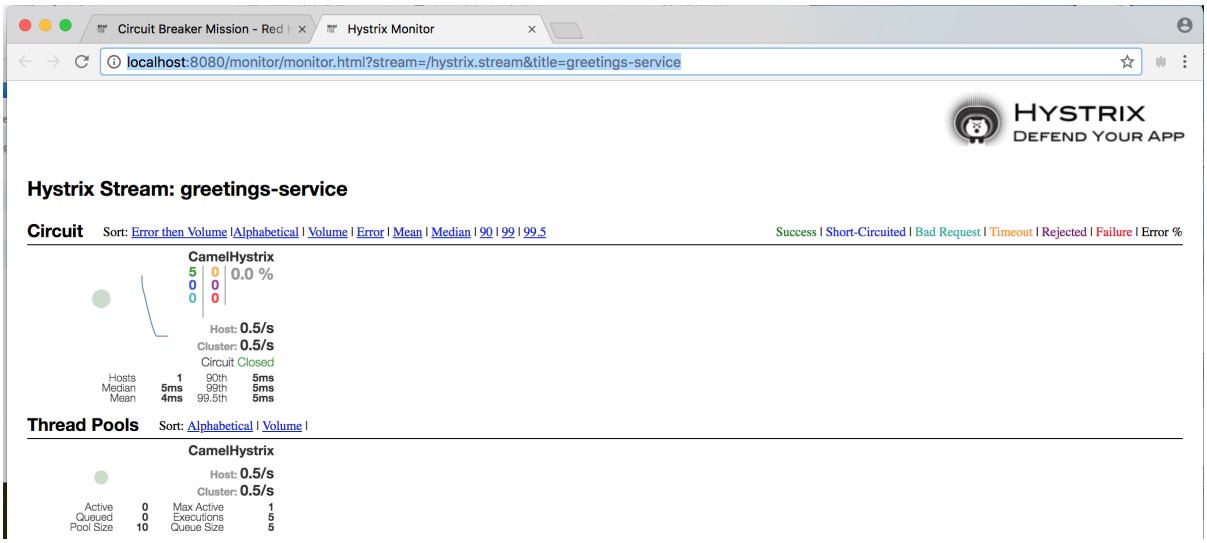
greetings 服务通过 URL 公开 REST 端点 http://localhost:8080/camel/greetings。

4. 进入 **http://localhost:8080**

打开此页面时，它会调用 Greeting Service：



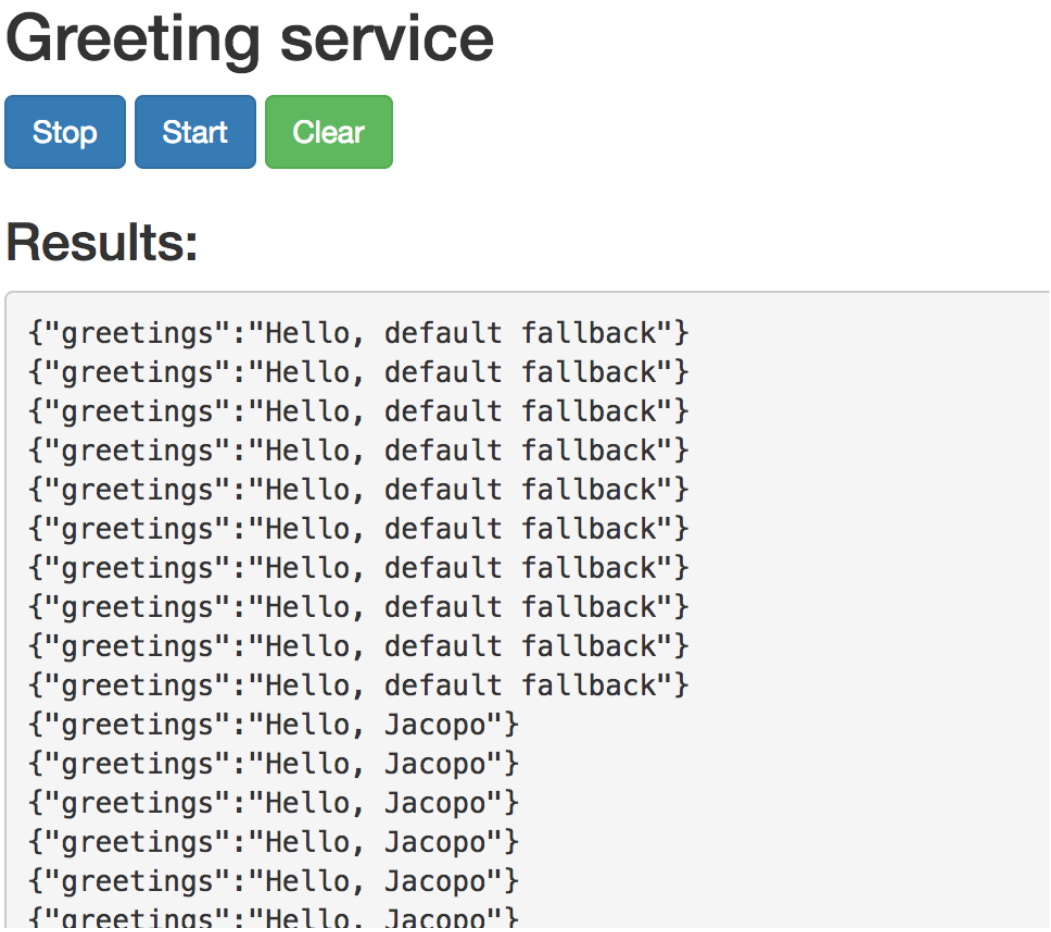
此页面也提供到 Hystrix 仪表板的链接，它监控断路器的状态。



5. 要演示 Camel Hystrix 提供的断路器功能，请在名称服务的 shell 提示符窗口中按 Ctrl+C 来终止后端名称服务。

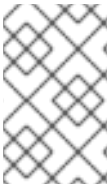
现在，名称服务不可用，断路器在 中启动，以防止在调用时间候服务挂起。

6. 观察 Hystrix Monitor 仪表板和 Greeting Service 输出中的更改：



2.3. 构建并运行 EXTERNALIZED CONFIGURATION BOOSTER

Externalized Configuration booster 提供了如何为 Apache Camel 路由外部配置的示例。对于 Spring Boot 独立部署，配置数据存储在 application.properties 文件中。



注意

对于 OpenShift 部署的 Fuse，配置数据存储在 ConfigMap 对象中。

先决条件

- 您已完成了 第 2.1 节 “生成您的 booster 项目” 部分所述的步骤。

流程

按照 外部配置 任务的 第 2.1 节 “生成您的 booster 项目” 步骤后，按照以下步骤构建并运行外部配置提升为本地机器上的独立项目：

1. 下载项目并在本地文件系统中提取存档。

2. 构建项目：

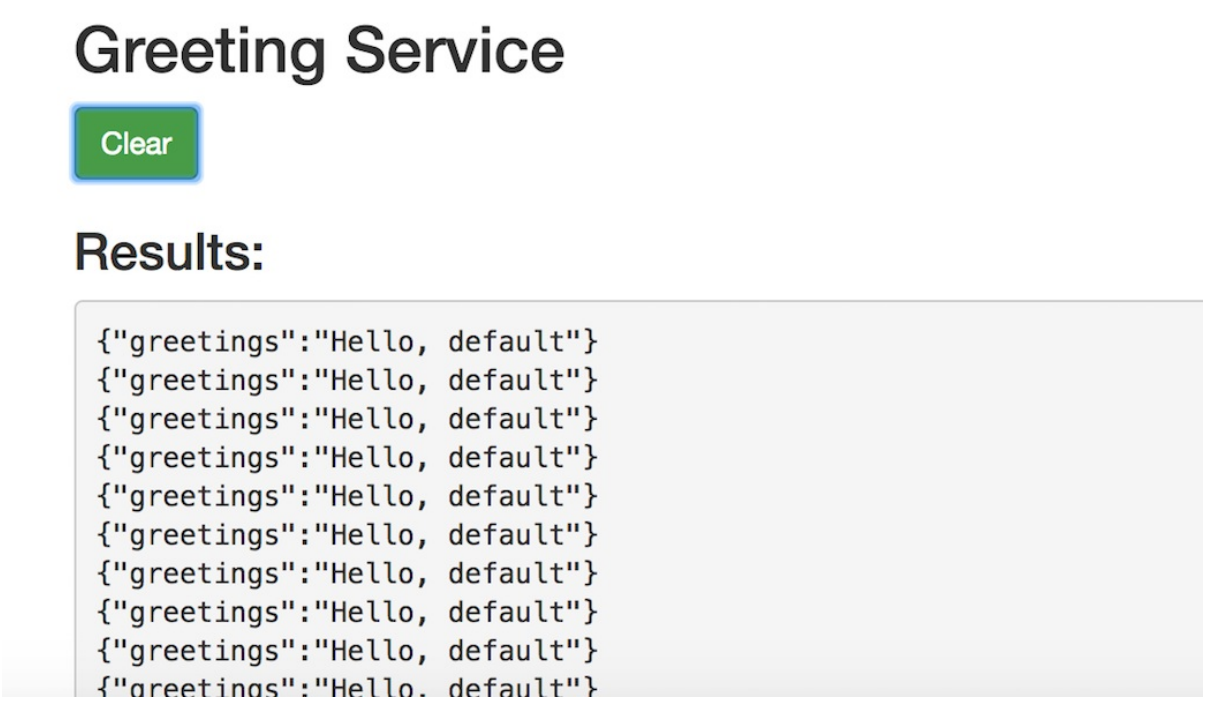
```
cd PROJECT_DIR
mvn clean package
```

3. 运行该服务：

```
mvn spring-boot:run
```

4. 打开 Web 浏览器，访问 http://localhost:8080。此页面每 5 秒调用一次 Greeting Service。Greetings Service 通过使用来自 target/classes/application.properties 文件中的 booster.nameToGreetvalue 来响应。

   Greeting Service 每 5 秒显示问候到"默认"：



5. 修改 booster.nameToGreet 值：

   a. 在编辑器中打开 target/classes/application.properties 文件。

b. 将 booster.nameToGreetvalue 的值从 默认值 改为另一个值，例如：

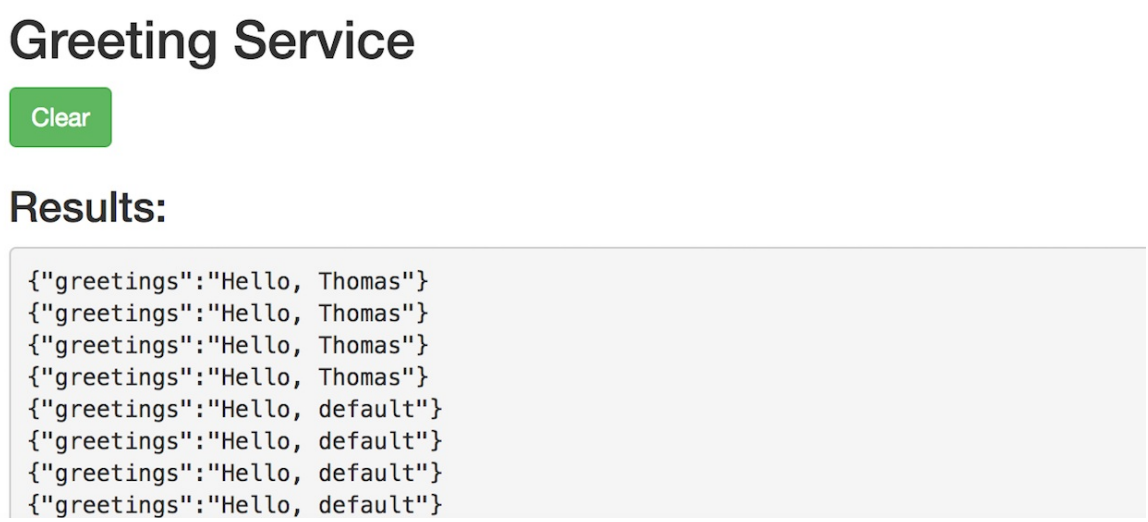
```
booster.nameToGreetvalue=Thomas
```

6. 在终端窗口中，按 CTRL+C 来停止服务。

7. 再次运行该服务：

```
mvn spring-boot:run
```

8. 在 Web 浏览器中，返回到 http://localhost:8080 页面，在 Greeting Service 结果窗口中查看更改的值。



2.4. 构建并运行 REST API BOOSTER

REST API 级别 0 任务演示了如何使用 REST 框架通过 HTTP 将业务运营映射到通过 HTTP 的远程过程调用端点。这个任务与用于 Matson Maturity Model 中的 Level 0 对应。

REST API 提升器引入了与使用 HTTP 协议的远程（由 Apache Camel 公开）服务交互的机制。通过使用此 Fuse 提升，您可以快速构建并灵活地设计 REST API。

使用这个 booster 来：

- 在 camel/greetings/{name} 端点上执行 HTTP GET 请求。此请求以 JSON 格式生成响应，

有效负载为 Hello, $name!（其中 $name 替换为 HTTP GET 请求中的 URL 参数的值）。

- 更改 URL {name} 参数的值，以查看响应中已更改的值。
- 查看 REST API 的 Swagger 页面。

先决条件

- 您已完成了 第 2.1 节 “生成您的 booster 项目” 部分所述的步骤。

流程

按照以下步骤，在本地机器上构建并运行 REST API booster 作为独立项目：

1. 下载项目并在本地文件系统中提取存档。
2. 构建项目：

```
cd PROJECT_DIR
mvn clean package
```

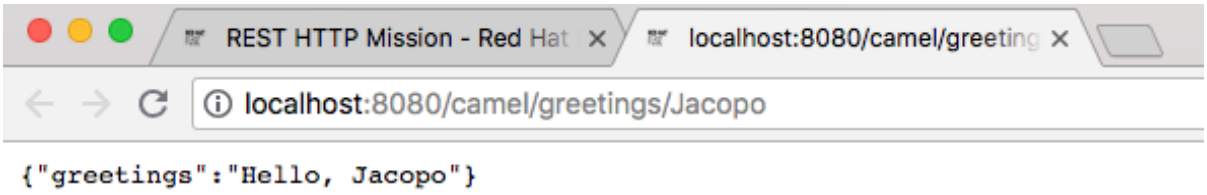
3. 运行该服务：

```
mvn spring-boot:run
```

4. 打开 Web 浏览器 ：http://localhost:8080
5. 要执行 HTTP GET 请求示例，请单击 camel/greetings/{name} 按钮。

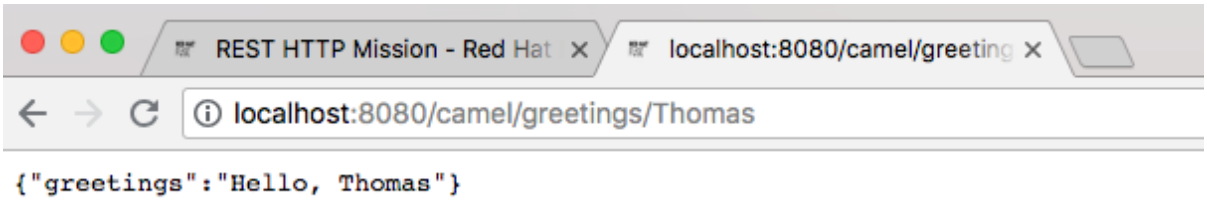
此时会打开一个新的 Web 浏览器窗口，其中包含 localhost:8080/camel/greetings/Jacopo URL。URL {name} 参数的默认值为 Jacopo。

JSON 响应会出现在浏览器窗口中：



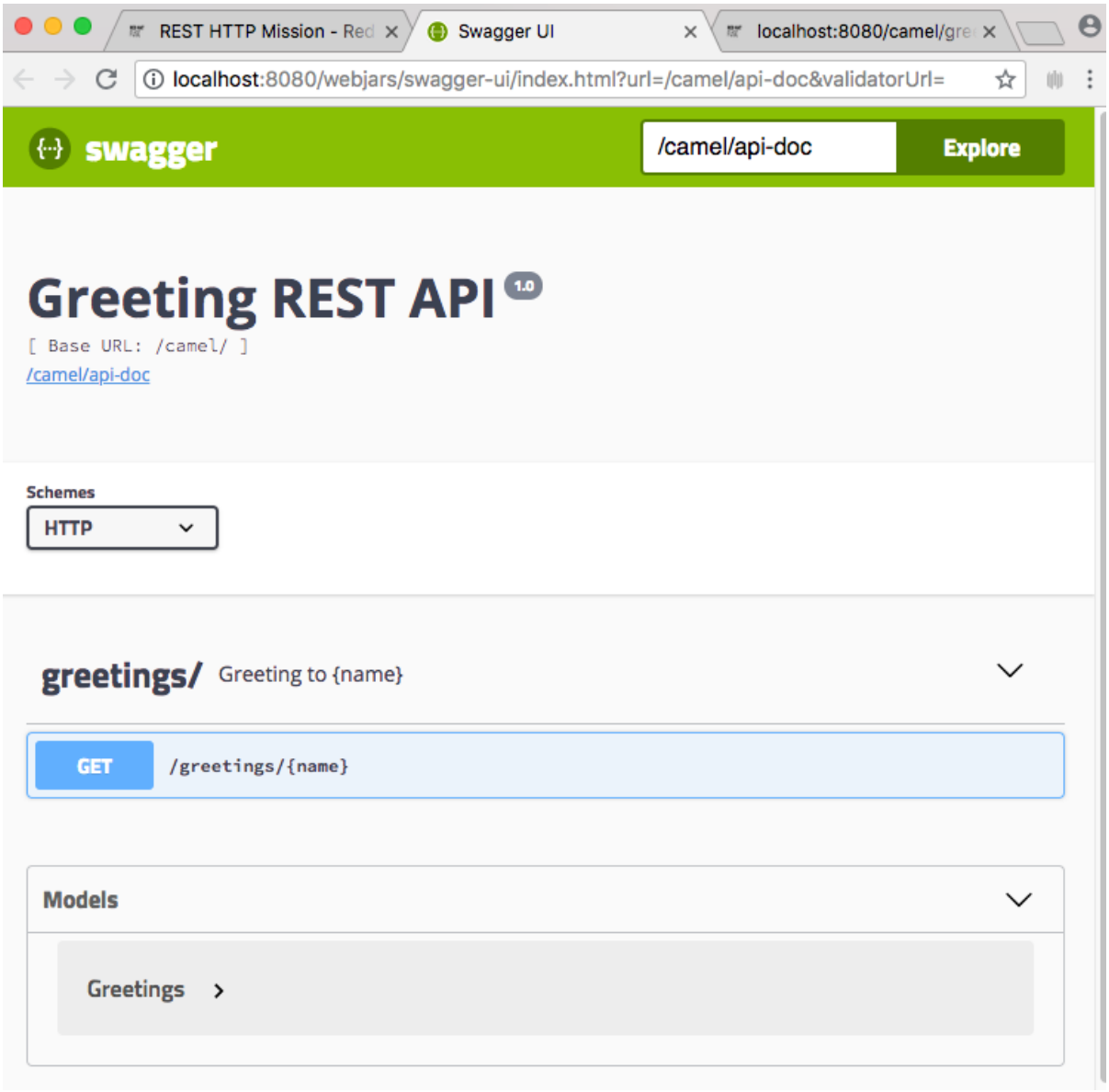
6. **要更改 {name} 参数的值，请更改 URL。例如，要将名称更改为，可使用此 URL：localhost:8080/camel/greetings/ Thomas 。**

   更新的 JSON 响应会出现在浏览器窗口中：



7. **要查看 REST API 的 Swagger 页面，请点 API Swagger 页面按钮。**

   API swagger 页面在浏览器窗口中打开。


REST HTTP Mission - Red
Swagger UI
localhost:8080/camel/gre
localhost:8080/webjars/swagger-ui/index.html?url=/camel/api-doc&validatorUrl=
swagger
/camel/api-doc
Explore
Greeting REST API
1.0
[Base URL: /camel/]
/camel/api-doc
Schemes
HTTP
greetings/
Greeting to {name}
GET
/greetings/{name}
Models
Greetings

第 3 章 使用带有 SPRING BOOT 的 RED HAT SINGLE SIGN-ON

Red Hat Single Sign-On 客户端适配器是库，它非常容易使用 Red Hat Single Sign-On 来保护应用程序和服务。您可以使用 Keycloak Spring Boot 适配器来保护 Spring Boot 项目。

3.1. 在 SPRING BOOT CONTAINER 中使用 RED HAT SINGLE SIGN-ON

要保护 Spring Boot 应用程序，请将 Keycloak Spring Boot 适配器 JAR 添加到您的项目中。Keycloak Spring Boot 适配器利用 Spring Boot 的自动配置功能，因此您需要做的都是将 Keycloak Spring Boot Start 添加到项目中。

流程

1. 要手动添加 Keycloak Spring Boot starter，请将以下内容添加到项目的 pom.xml 中。

```
<dependency>
      <groupId>org.keycloak</groupId>
      <artifactId>keycloak-spring-boot-starter</artifactId>
</dependency>
```

2. 添加 Adapter BOM 依赖项。

```
<dependencyManagement>
 <dependencies>
      <dependency>
        <groupId>org.keycloak.bom</groupId>
        <artifactId>keycloak-adapter-bom</artifactId>
        <version>3.4.17.Final-redhat-00001</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
 </dependencies>
</dependencyManagement>
```

3. 将您的 Spring Boot 项目配置为使用 Keycloak。您可以使用正常的 Spring Boot 配置为 Spring Boot Keycloak 适配器配置域，而不是 keycloak.json 文件。例如，将以下配置添加到 src/main/resources/application.properties 文件中。

```
keycloak.realm = demorealm
keycloak.auth-server-url = http://127.0.0.1:8080/auth
keycloak.ssl-required = external
keycloak.resource = demoapp
keycloak.credentials.secret = 11111111-1111-1111-1111-111111111111
keycloak.use-resource-role-mappings = true
```

-

您可以通过设置 keycloak.enabled = false 来禁用 Keycloak Spring Boot Adapter （例如在测试中）。要配置 Policy Enforcer，与 keycloak.json 不同，必须使用policy-enforcer-config 而不是只使用 policy-enforcer。

4. **在 web.xml 中指定 Java EE 安全配置。Spring Boot Adapter 会将 login-method 设置为 KEYCLOAK，并在启动时配置 security- constraint。下面是一个示例配置。**

```
keycloak.securityConstraints[0].authRoles[0] = admin
keycloak.securityConstraints[0].authRoles[1] = user
keycloak.securityConstraints[0].securityCollections[0].name = insecure stuff
keycloak.securityConstraints[0].securityCollections[0].patterns[0] = /insecure


keycloak.securityConstraints[1].authRoles[0] = admin
keycloak.securityConstraints[1].securityCollections[0].name = admin stuff
keycloak.securityConstraints[1].securityCollections[0].patterns[0] = /admin
```

注： 如果您计划将 Spring 应用程序部署为 WAR，则不要使用 Spring Boot Adapter。将专用适配器用于您正在使用的应用服务器或 servlet 容器。Spring Boot 也应包含 web.xml 文件。

第 4 章 如何在 SPRING BOOT 中使用加密属性占位符

在保护容器时，不建议在配置文件中使用纯文本密码。避免使用纯文本密码的一种方法是尽可能使用加密属性占位符。

4.1. 关于用于加密值的 MASTER 密码

要使用 Jasypt 加密值，需要一个 master 密码。供您或管理员选择 master 密码。Jasypt 提供了多种方式来设置 master 密码。Jasypt 可以集成到 Spring 配置框架中，以便在加载配置文件时解密属性值。其中一种方法是在 Spring 引导配置中以纯文本形式指定 master 密码。

Spring 使用 PropertyPlaceholder 框架将令牌替换为属性文件中的值，Jasypt 的方法将 PropertyPlaceholderConfigurer 类替换为可识别加密字符串并解密它们的方法。

Example

```
<bean id="propertyPlaceholderConfigurer"
    class="org.jasypt.spring.properties.EncryptablePropertyPlaceholderConfigurer">
  <constructor-arg ref="configurationEncryptor" />
  <property name="location" value="/WEB-INF/application.properties" />
</bean>

<bean id="configurationEncryptor" class="org.jasypt.encryption.pbe.StandardPBEStringEncryptor">
  <property name="config" ref="environmentVariablesConfiguration" />
</bean>

<bean id="environmentVariablesConfiguration"
    class="org.jasypt.encryption.pbe.config.EnvironmentStringPBEConfig">
  <property name="algorithm" value="PBEWithMD5AndDES" />
  <property name="password" value="myPassword" />
</bean>
```

您可以使用环境变量来设置 master 密码，而不是以纯文本形式指定 master 密码。在 Spring Boot 配置文件中，将这个环境变量指定为 passwordEnvName 属性的值。例如，如果您将 MASTER_PW 环境变量设置为 master 密码，则 Spring Boot 配置文件中会具有此条目：

```
<property name="passwordEnvName" value="MASTER_PW">
```

4.2. 在 SPRING BOOT 中使用 ENCRYPTED PROPERTY PLACEHOLDERS

通过使用 Jasypt，您可以为属性源提供加密，应用可以解密加密的属性并检索原始值。以下流程解释了如何加密和解密 Spring Boot 中的属性源。

流程

1. 将 jasypt 依赖项添加到项目的 pom.xml 文件中。

```
<dependency>
    <groupId>com.github.ulisesbocchio</groupId>
    <artifactId>jasypt-spring-boot-starter</artifactId>
    <version>3.0.3</version>
</dependency>
```

2. 将 Maven 存储库添加到项目的 pom.xml 中。

```
<repository>
<id>jasypt-basic</id>
<name>Jasypt Repository</name>
<url>https://repo1.maven.org/maven2/</url>
</repository>
```

3. 将 Jasypt Maven 插件添加到项目中，并允许您使用 Maven 命令加密和解密。

```
<plugin>
  <groupId>com.github.ulisesbocchio</groupId>
  <artifactId>jasypt-maven-plugin</artifactId>
  <version>3.0.3</version>
 </plugin>
```

4. 将插件存储库添加到 pom.xml。

```
<pluginRepository>
  <id>jasypt-basic</id>
  <name>Jasypt Repository</name>
  <url>https://repo1.maven.org/maven2/</url>
</pluginRepository>
```

5. 要加密 application.properties 文件中列出的用户名和密码，请将这些值嵌套在 DEC （） 中，如下所示。

```
spring.datasource.username=DEC(root)
spring.datasource.password=DEC(Password@1)
```

6. 运行以下命令来加密用户名和密码。

```
mvn jasypt:encrypt -Djasypt.encryptor.password=mypassword
```

这将 **application.properties** 文件中的 **DEC （）占位符替换为加密值，例如：**

```
spring.datasource.username=ENC(3UtB1NhSZdVXN9xQBwkT0Gn+UxR832XP+tOOfFTlNL5
7FiMM7BWPRTeychVtLLhB)
spring.datasource.password=ENC(4ErqElyCHjjFnqPOCZNAaTdRC7u7yJSy16UsHtVkwPIr+3z
LyabNmQwwpFo7F7LU)
```

7. 要解密 **Spring** 应用配置文件中的凭据，请运行以下命令：

```
mvn jasypt:decrypt -Djasypt.encryptor.password=mypassword
```

这会输出 **application.properties** 文件的内容，因为它在加密前显示。但是，这不会更新配置文件。

第 5 章 使用 MAVEN 构建

在 Fuse 中为 Spring Boot 开发应用程序的标准方法是使用 Apache Maven 构建工具，并将源代码构建为 Maven 项目。Fuse 提供 Maven 快速入门，以便您快速启动，许多 Fuse 构建工具都作为 Maven 插件提供。因此，强烈建议您使用 Maven 作为 Fuse 中的 Spring Boot 项目的构建工具。

5.1. 生成 MAVEN 项目

Fuse 提供基于 Maven archetypes 的快速入门选择，您可以使用它来为 Spring Boot 应用程序生成初始 Maven 项目。为了防止您记住各种 Maven 构架类型的位置信息和版本，Fuse 提供了工具来帮助您为独立的 Spring Boot 项目生成 Maven 项目。

5.1.1. developer.redhat.com/launch 的项目生成器

在 Fuse 中使用 Spring Boot 独立入门的最快速方法是导航到 developers.redhat.com/launch，并按照 Spring Boot 独立运行时的说明进行操作，以生成新的 Maven 项目。在遵循屏幕说明后，系统会提示您下载存档文件，其中包含一个完整的 Maven 项目，您可以在本地构建和运行。

5.1.2. Developer Studio 中的 Fuse 工具向导

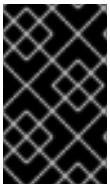
或者，您可以下载并安装红帽 JBoss Developer Studio （包括 Fuse 工具）。使用 Fuse New Integration Project 向导，您可以生成新的 Spring Boot 独立项目，并继续开发基于 Eclipse 的 IDE。

5.2. 使用 SPRING BOOT BOM

创建并构建第一个 Spring Boot 项目后，您很快会想添加更多组件。但是，您如何知道要添加到项目的 Maven 依赖项？最简单的（和推荐）方法是使用相关的 Bill of Materials (BOM)文件，该文件会自动为您定义所有版本依赖项。

5.2.1. Spring Boot 的 BOM 文件

Maven Bill of Materials (BOM) 文件的目的是提供一组可正常工作的 Maven 依赖项版本，从而防止您必须为每个 Maven 工件单独定义版本。

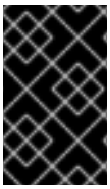


重要

确保您使用基于您正在使用的 Spring Boot 版本的正确 Fuse BOM。

Spring Boot 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。

5.2.2. 合并 BOM 文件

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，可能在父 POM 文件中），如两个 Spring Boot 2 的示例所示：

- Spring Boot 2 BOM

Spring Boot 2 BOM

```
<?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-springboot-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
```

```
    </dependency>
  </dependencies>
 </dependencyManagement>
 ...
</project>
```

在使用依赖项管理机制指定 BOM 后，可以在没有指定工件版本 *的情况下将* Maven 依赖项添加到您的 POM 中。例如，要为 camel-hystrix 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-hystrix-starter</artifactId>
</dependency>
```

请注意，Camel 工件 ID 如何通过 -starter 后缀，将 Camel Hystrix 组件指定为 camel-hystrix-starter，而不是 camel-hystrix。Camel 初学者组件以针对 Spring Boot 环境进行了优化的方式进行打包。

5.2.3. Spring Boot Maven 插件

Spring Boot Maven 插件由 Spring Boot 提供，它是一个用于构建和运行 Spring Boot 项目的开发人员实用程序：

- 通过在项目目录中输入 mvn package，为您的 Spring Boot 应用程序 *构建* 可执行文件 Jar 软件包。构建的输出放置在 Maven 项目的 target/ 子目录中。
- 为方便起见，您可以使用命令 mvn spring-boot:start 运行新构建的应用程序。

要将 Spring Boot Maven 插件合并到项目 POM 文件中，请将插件配置添加到 pom.xml 文件的 project/build/plugins 部分，如下例所示。

Example

```
<?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>
...
<build>
  <plugins>
    <plugin>
      <groupId>org.jboss.redhat-fuse</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <version>${fuse.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
...
</project>
```

第 6 章 在 SPRING BOOT 中运行 APACHE CAMEL 应用程序

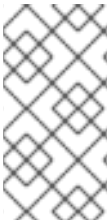
Apache Camel Spring Boot 组件为 Spring Boot 自动配置 Camel 上下文。Camel 上下文的自动配置会自动检测到 Spring 上下文中提供的 Camel 路由，并将生成者模板、消费者模板以及类型转换器注册为 Bean。Apache Camel 组件包含一个 Spring Boot starter 模块，允许您使用 starters 开发 Spring Boot 应用程序。

6.1. CAMEL SPRING BOOT 组件简介

每个 Camel Spring Boot 应用程序都必须使用项目的 pom.xml 中的 dependencyManagement 元素来指定依赖项的产品化版本。这些依赖项在 Red Hat Fuse BOM 中定义，并受特定版本的 Red Hat Fuse 的支持。您可以为额外的启动者省略 version number 属性，以便不覆盖 BOM 中的版本。如需更多信息，请参阅 Quickstart pom。

Example

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



注意

camel-spring-boot jar 包含 spring.factories 文件，该文件允许您将依赖项添加到 classpath 中，以便 Spring Boot 自动配置 Camel 上下文。

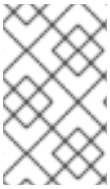
6.2. CAMEL SPRING BOOT STARTER 模块简介

Starters 是应该在 Spring Boot 应用程序中使用的 Apache Camel 模块。每个 Camel 组件都有一个 camel-xxx-starter 模块（在 第 6.3 节 “没有入门模块的 Camel 组件列表” 部分列出了几个例外）。

开始者满足以下要求：

- 使用与 IDE 工具兼容的原生 Spring Boot 配置系统允许自动配置组件。
- 允许自动配置数据格式和语言。
- 管理传输日志记录依赖项，以与 Spring Boot 日志记录系统集成。
- 包括额外的依赖项并对齐传输的依赖关系，以最大程度降低创建正常工作的 Spring Boot 应用程序的工作量。

每个入门程序在 test /camel-itest-spring-boot 中都有自己的集成测试，用于验证与 Spring Boot 的当前发行版本的兼容性。



注意

如需了解更多详细信息，请参阅链接： Apache Camel Spring-Boot 示例。

6.3. 没有入门模块的 CAMEL 组件列表

由于兼容性问题，以下组件没有初学者模块：

- camel-blueprint （只针对 OSGi 引入）
- camel-cdi （仅限 CDI 引入）
- camel-core-osgi （仅限 OSGi 修改）
- camel-ejb （仅限 JUnit 修改）

- *camel-eventadmin （仅限 OSGi 引入）*
- *camel-ibatis （包含camel-mybatis-starter ）*
- *camel-jclouds*
- *camel-mina （包含camel-mina2-starter ）*
- *camel-paxlogging （仅限 OSGi 修改）*
- *camel-quartz （包含camel-quartz2-starter ）*
- *camel-spark-rest*
- *camel-openapi-java (camel-openapi-java-starter)*

6.4. 使用 CAMEL SPRING BOOT STARTER

Apache Camel 提供了一个初学者模块，可让您快速开始开发 Spring Boot 应用程序。

流程

1. *将以下依赖项添加到 Spring Boot pom.xml 文件中：*

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

2. *使用 Camel 路由添加类，如下面的代码片段中所示。将这些路由添加到类路径中后，路由会自动启动。*

```
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");
    }
}
```

3. 可选。要让主线程阻止，以便 Camel 保持工作，请执行以下操作之一：

   a. 包含 spring-boot-starter-web 依赖项，

   b. 或将 camel.springboot.main-run-controller=true 添加到 application.properties 或 application.yml 文件中。

   您可以使用 camel.springboot mdadm 属性自定义 application.properties 或 application. yml 文件中的 Camel 应用程序。

4. 可选。要使用 bean 的 ID 名称引用自定义 bean，请在 src/main/resources/application.properties （或 application.yml）文件中配置选项。以下示例显示了 xslt 组件如何使用 bean ID 来引用自定义 bean。

   a. 请参阅 id myExtensionFactory 的自定义 bean。

   ```
   camel.component.xslt.saxon-extension-functions=myExtensionFactory
   ```

   b. 然后，使用 Spring Boot @Bean 注释创建自定义 bean。

   ```
   @Bean(name = "myExtensionFactory")
   public ExtensionFunctionDefinition myExtensionFactory() {
     }
   ```

   或者，对于 Jackson ObjectMapper，在 camel-jackson data-format 中：

```
camel.dataformat.json-jackson.object-mapper=myJacksonMapper
```

6.5. 关于 SPRING BOOT 的 CAMEL 上下文自动配置

Camel Spring Boot 自动配置提供 CamelContext 实例，并创建一个 SpringCamelContext。它还初始化并执行该上下文的关机。此 Camel 上下文在 camelContext bean 名称下的 Spring 应用程序上下文中注册，您可以像其他 Spring bean 一样访问它。您可以访问 camelContext，如下所示。

Example

```
@Configuration
public class MyAppConfig {

  @Autowired
  CamelContext camelContext;

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

}
```

6.6. SPRING BOOT APPLICATIONS 中的自动探测 CAMEL 路由

Camel 自动配置从 Spring 上下文收集所有 RouteBuilder 实例，并将它们自动注入到 CamelContext 中。这简化了使用 Spring Boot 启动程序创建新的 Camel 路由的过程。您可以创建路由，如下所示：

Example

将 @Component annotated 类添加到您的 classpath 中。

```
@Component
public class MyRouter extends RouteBuilder {

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

}
```

或者在您的 @Configuration 类中创建一个新的路由 RouteBuilder bean。

```
@Configuration
public class MyRouterConfiguration {

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

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

    };
  }

}
```

6.7. 为 CAMEL SPRING BOOT 自动配置配置 CAMEL 属性

Spring Boot 自动配置连接到 Spring Boot 外部配置，如属性占位符、OS 环境变量或带有 Camel 属性的系统属性。

流程

1. 在 application.properties 文件中定义属性：

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

或者，将 Camel 正确设置为系统属性，例如：

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

2. 按照如下所示，将配置的属性用作 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}}");
```

```
    }

}
```

6.8. 配置自定义 CAMEL 上下文

要对 Camel Spring Boot 自动配置创建的 CamelContext bean 执行操作，请在 Spring 上下文中注册 CamelContextConfiguration 实例。

流程

- 在 Spring 上下文中注册 CamelContextConfiguration 实例，如下所示。

```
@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，CamelContext Configuration 和 beforeApplicationStart (CamelContext) 方法被调用，因此传递给此回调的 CamelContext 实例会被完全自动配置。您可以将许多 CamelContextConfiguration 实例添加到 Spring 上下文中，并且所有实例都将被执行。

6.9. 在自动配置的 CAMELCONTEXT 中禁用 JMX

要在自动配置的 CamelContext 中禁用 JMX，您可以使用 camel.springboot.jmxEnabled 属性启用 JMX。

流程

- 在 application.properties 文件中添加以下属性，并将其设置为 false ：

```
camel.springboot.jmxEnabled = false
```

6.10. 将自动配置的消费者和制作者模板注入 SPRING 管理的 BEAN

Camel 自动配置提供预配置的 ConsumerTemplate 和 ProducerTemplate 实例。您可以将它们注入到 Spring 管理的 Bean 中。

Example

```
@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随端点缓存大小设置为 1000。您可以通过将以下 Spring 属性设置为所需缓存大小来更改这些值，例如：

```
camel.springboot.consumerTemplateCacheSize = 100
camel.springboot.producerTemplateCacheSize = 200
```

6.11. 关于 SPRING 上下文中自动配置的 TYPECONVERTER

Camel 自动配置在 Spring 上下文中注册一个名为 typeConverter 的 TypeConverter 实例。

Example

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

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

}
```

6.12. SPRING 类型转换 API 网桥

Spring 由强大的 类型转换 API 组成。Spring API 与 Camel 类型转换器 API 类似。由于两个 API Camel Spring Boot 之间的相似性会自动注册桥接转换器(SpringTypeConverter)，它委托给 Spring 转换 API。这意味着开箱即用的 Camel 将对待与 Camel 类似的 Spring Converters。

这可让您使用 Camel TypeConverter API 访问 Camel 和 Spring 转换器，如下所示：

Example

```
@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());
  }

}
```

这里，Spring Boot 将转换委托给应用程序上下文中提供的 Spring ConversionService 实例。如果没有 ConversionService 实例，Camel Spring Boot 自动配置会创建一个 ConversionService 实例。

6.13. 禁用类型转换功能

要禁用 Camel Spring Boot 类型转换功能，请将 camel.springboot.typeConversion 属性设置为 false。当此属性设置为 false 时，自动配置不会注册类型转换器实例，且不会启用将类型转换为 Spring Boot 类型转换 API。

流程

- 要禁用 Camel Spring Boot 组件的类型转换功能，将 camel.springboot.typeConversion 属性设置为 false，如下所示：

```
camel.springboot.typeConversion = false
```

6.14. 在类路径中添加 XML 路由以进行自动配置

默认情况下，Camel Spring Boot 组件会自动探测，并包含 camel 目录中的 classpath 中的 Camel XML 路由。您可以使用配置选项配置目录名称或禁用此功能。

流程

- 在类路径中配置 Camel Spring Boot XML 路由，如下所示：

```
// turn off
camel.springboot.xmlRoutes = false
// scan in the com/foo/routes classpath
camel.springboot.xmlRoutes = classpath:com/foo/routes/*.xml
```

注意

XML 文件应该定义 Camel XML 路由元素而不是 CamelContext 元素，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>
```

使用 Spring XML 文件

要将 Spring XML 文件与 <camelContext> 搭配使用，您可以在 Spring XML 文件或 application.properties 文件中配置 Camel 上下文。要设置 Camel 上下文的名称并打开流缓存，请在 application.properties 文件中添加以下内容：

```
camel.springboot.name = MyCamel
camel.springboot.stream-caching-enabled=true
```

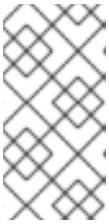
6.15. 为自动配置添加 XML REST-DSL 路由

Camel Spring Boot 组件会自动探测并嵌入 camel-rest 目录下的 classpath 中添加的 Camel Rest-DSL XML 路由。您可以使用配置选项配置目录名称或禁用此功能。

流程

- *在 classpath 中配置 Camel Spring Boot Rest-DSL XML 路由，如下所示：*

```
// turn off
camel.springboot.xmlRests = false
// scan in the com/foo/routes classpath
camel.springboot.xmlRests = classpath:com/foo/rests/*.xml
```



注意

Rest-DSL XML 文件应该定义 Camel XML REST 元素，而不是 CamelContext 元素，例如：

```
<rests xmlns="http://camel.apache.org/schema/spring">
  <rest>
    <post uri="/persons">
      <to uri="direct:postPersons"/>
    </post>
    <get uri="/persons">
      <to uri="direct:getPersons"/>
    </get>
    <get uri="/persons/{personId}">
      <to uri="direct:getPersionId"/>
    </get>
    <put uri="/persons/{personId}">
      <to uri="direct:putPersionId"/>
    </put>
    <delete uri="/persons/{personId}">
      <to uri="direct:deletePersionId"/>
```

```
    </delete>
  </rest>
</rests>
```

6.16. 使用 CAMEL SPRING BOOT 测试

当 Camel 在 Spring Boot 上运行时，Spring Boot 会自动嵌入 Camel 及其所有路由，这些路由使用 @Component 标注。使用 Spring Boot 测试时，使用 @SpringBootTest 而不是 @ContextConfiguration 来指定要使用的配置类。

当您在不同的 RouteBuilder 类中有多个 Camel 路由时，Camel Spring Boot 组件会在运行应用程序时自动嵌入所有这些路由。因此，当您希望只从一个 RouteBuilder 类测试路由时，您可以使用以下模式包含或排除要启用的 RouteBuilders：

- java-routes-include-pattern: 用于包括与模式匹配的 RouteBuilder 类。
- java-routes-exclude-pattern: 用于排除与模式匹配的 RouteBuilder 类。exclude 优先于 include。

流程

1. 将单元测试类中的 include 或 exclude 模式指定为 @SpringBootTest 注释的属性，如下所示：

```
@RunWith(CamelSpringBootRunner.class)
@SpringBootTest(classes = {MyApplication.class);
  properties = {"camel.springboot.java-routes-include-pattern=**/Foo*"})
public class FooTest {
```

在 FooTest 类中，include 模式是 **/Foo*，它代表 Ant 样式模式。在这里，模式以双星号开头，该星号与任何前导软件包名称匹配。/foo* 表示类名称必须以 Foo 开头，例如 FooRoute。

2. 使用以下 maven 命令运行测试：

```
mvn test -Dtest=FooTest
```

其它资源

- 编写组件
- 组件
- 端点
- 开始使用

6.17. 使用 SPRING BOOT、APACHE CAMEL 和外部消息传递代理

Fuse 使用外部消息传递代理。有关支持的代理、客户端和 Camel 组件组合的更多信息，请参阅 支持的配置。

Camel 组件必须连接到 JMS connection-factory。以下示例演示了如何将 camel-amqp 组件连接到 JMS connection-factory。

```
import org.apache.activemq.jms.pool.PooledConnectionFactory;
import org.apache.camel.component.amqp.AMQPComponent;
import org.apache.qpid.jms.JmsConnectionFactory;
...

AMQPComponent amqpComponent(AMQPConfiguration config) {
    JmsConnectionFactory qpid = new JmsConnectionFactory(config.getUsername(),
config.getPassword(), "amqp://"+ config.getHost() + ":" + config.getPort());
    qpid.setTopicPrefix("topic://");

    PooledConnectionFactory factory = new PooledConnectionFactory();
    factory.setConnectionFactory(qpid);

AMQPComponent amqpcomp = new AMQPComponent(factory);
```

第 7 章 修补 RED HAT FUSE 应用程序

使用新的 patch-maven-plugin 机制，您可以对 Red Hat Fuse 应用程序应用补丁。此机制允许您更改由不同 Red Hat Fuse BOMS 提供的独立版本，如 fuse-springboot-bom 和 fuse-karaf-bom。

7.1. 关于 PATCH-MAVEN-PLUGIN

patch-maven-plugin 执行以下操作：

- 检索与当前 Red Hat Fuse 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）。替换了版本后，Maven 构建将继续，并通过标准 Maven 项目阶段进行进度。

7.2. 将补丁应用到 RED HAT FUSE 应用程序

patch-maven-plugin 的目的是将 Red Hat Fuse 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。

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 Fuse BOM 的版本相同。

```
<build>
  <plugins>
    <plugin>
      <groupId>org.jboss.redhat-fuse</groupId>
      <artifactId>patch-maven-plugin</artifactId>
      <version>${version.org.jboss-redhat-fuse}</version>
      <extensions>true</extensions>
    </plugin>
  </plugins>
</build>
```

2. 当您运行 mvn clean deploy 或 mvn dependency:tree 命令中的任何一个时，插件会搜索项目模块来检查是否使用其中一个 Red Hat Fuse BOM。只有两个被认为是支持的 BOM：

   - org.jboss.redhat-fuse:fuse-karaf-bom: 用于 Fuse Karaf BOM
   - org.jboss.redhat-fuse:fuse-springboot-bom: 用于 Fuse Spring Boot BOM

3. 如果没有找到以上 BOM，插件将显示以下信息：

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

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

[INFO] [PATCH] No project in the reactor uses Fuse Karaf or Fuse Spring Boot BOM.
Skipping patch processing.
[INFO] [PATCH] Done in 3ms
```

4. 如果同时找到了两个 Fuse BOM，patch-maven-plugin 会停止，并显示以下警告：

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

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

[WARNING] [PATCH] Reactor uses both Fuse Karaf and Fuse Spring Boot BOMs. Please
use only one. Skipping patch processing.
[INFO] [PATCH] Done in 3ms
```

5. patch-maven-plugin 尝试获取以下 Maven 元数据值之一。

   - 对于使用 Fuse Karaf BOM 的项目，org.jboss.redhat-fuse/fuse-karaf-patch-metadata/maven-metadata.xml 已解决。这是带有 org.jboss.redhat-fuse:fuse-karaf-patch-metadata:RELEASE 的工件的元数据。
   - 对于使用 Fuse Spring Boot BOM 项目的项目，org.jboss.redhat-fuse/fuse-springboot-patch-metadata/maven-metadata.xml 已解决。这是带有 org.jboss.redhat-

fuse:fuse-springboot-patch-metadata:RELEASE 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>org.jboss.redhat-fuse</groupId>
  <artifactId>fuse-springboot-patch-metadata</artifactId>
  <versioning>
    <release>7.8.1.fuse-sb2-781025</release>
    <versions>
      <version>7.8.0.fuse-sb2-780025</version>
      <version>7.7.0.fuse-sb2-770010</version>
      <version>7.7.0.fuse-770010</version>
      <version>7.8.1.fuse-sb2-781025</version>
    </versions>
    <lastUpdated>20201023131724</lastUpdated>
  </versioning>
</metadata>
```

6. *patch-maven-plugin 解析元数据，以选择适用于当前项目的版本。这只适用于使用带有版本 7.8.xxx 的 Fuse BOM 的 Maven 项目。只有与版本范围 7.8、7.9 或更高版本匹配的元数据才适用，且只获取最新版本的元数据。*

7. *patch-maven-plugin 收集在下载由 groupId、artifactId 和版本中找到的 groupId、artifactId 和版本 标识时使用的远程 Maven 存储库列表。这些 Maven 存储库是活跃配置集的项目 < repositories> 元素中列出的 Maven 存储库，以及来自 settings.xml 文件中的存储库。*

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

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

[INFO] [PATCH] Reading patch metadata and artifacts from 2 project repositories
[INFO] [PATCH]  - local-nexus: http://everfree.forest:8081/repository/maven-releases/
[INFO] [PATCH]  - central: https://repo.maven.apache.org/maven2
Downloading from local-nexus: http://everfree.forest:8081/repository/maven-
releases/org/jboss/redhat-fuse/fuse-springboot-patch-metadata/maven-metadata.xml
...
```

8. *另外，如果要使用离线存储库，您可以使用 -Dpatch 选项指定由 fuse-karaf/fuse-karaf-*

patch-repository 或 fuse-springboot/fuse-springboot-patch - repository 模块生成的 ZIP 文件。这些 ZIP 文件与 Maven 存储库结构具有相同的内部结构。例如，

```
$ mvn clean install -Dpatch=../../../test/resources/patch-3.zip
[INFO] Scanning for projects...
[INFO]

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

[INFO] [PATCH] Reading metadata and artifacts from /data/sources/github.com/jboss-fuse/redhat-fuse/fuse-tools/patch-maven-plugin/src/test/resources/patch-3.zip
Downloading from fuse-patch: zip:file:/tmp/patch-3.zip-17429742145982057 45/org/jboss/redhat-fuse/fuse-springboot-patch-metadata/maven-metadata.xml
Downloaded from fuse-patch: zip:file:/tmp/patch-3.zip-17429742145982057 45/org/jboss/redhat-fuse/fuse-springboot-patch-metadata/maven-metadata.xml (406 B at 16 kB/s)
Downloading from fuse-patch: zip:file:/tmp/patch-3.zip-17429742145982057 45/org/jboss/redhat-fuse/fuse-springboot-patch-metadata/7.8.0.fuse-sb2-781023/fuse-springboot-patch-metadata-7.8.0.fuse-sb2-781023.xml
Downloaded from fuse-patch: zip:file:/tmp/patch-3.zip-17429742145982057 45/org/jboss/redhat-fuse/fuse-springboot-patch-metadata/7.8.0.fuse-sb2-781023/fuse-springboot-patch-metadata-7.8.0.fuse-sb2-781023.xml (926 B at 309 kB/s)
[INFO] [PATCH] Resolved patch descriptor: /home/user/.m2/repository/org/jboss/redhat-fuse/fuse-springboot-patch-metadata/7.8.0.fuse-sb2-781023/fuse-springboot-patch-metadata-7.8.0.fuse-sb2-781023.xml
...
```

9. *元数据是否来自远程存储库、本地存储库还是 ZIP 文件，它由 patch-maven-plugin 进行分析。获取的元数据包含 CVE 列表，以及每个 CVE 都有一个受影响的 Maven 工件列表（由 glob 模式和版本范围指定），以及一个包含给定 CVE 修复的版本。例如，*

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

<metadata xmlns="urn:redhat:fuse:patch-metadata:1">
    <product-bom groupId="org.jboss.redhat-fuse" artifactId="fuse-springboot-bom" versions="[7.8,7.9)" />
    <cves>
        <cve id="CVE-2020-xyz" description="Jetty can be configured to listen on port 8080"
             cve-link="https://nvd.nist.gov/vuln/detail/CVE-2020-xyz"
             bz-link="https://bugzilla.redhat.com/show_bug.cgi?id=42">
            <affects groupId="org.eclipse.jetty" artifactId="jetty-*" versions="[9.4,9.4.32)" fix="9.4.32.v20200930" />
            <affects groupId="org.eclipse.jetty.http2" artifactId="http2-*" versions="[9.4,9.4.32)" fix="9.4.32.v20200930" />
        </cve>
    </cves>
    <fixes />
</metadata>
```

10. *最后，当迭代当前项目中的所有受管依赖项时，会查阅补丁元数据中指定的修复列表。这些匹*

配的依赖项（和受管依赖项）会改为固定版本。例如：

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

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

[INFO] [PATCH] Reading patch metadata and artifacts from 2 project repositories
[INFO] [PATCH]  - local-nexus: http://everfree.forest:8081/repository/maven-releases/
[INFO] [PATCH]  - central: https://repo.maven.apache.org/maven2
Downloading from local-nexus: http://everfree.forest:8081/repository/maven-releases/org/jboss/redhat-fuse/fuse-springboot-patch-metadata/maven-metadata.xml
Downloading from central: https://repo.maven.apache.org/maven2/org/jboss/redhat-fuse/fuse-springboot-patch-metadata/maven-metadata.xml
Downloaded from local-nexus: http://everfree.forest:8081/repository/maven-releases/org/jboss/redhat-fuse/fuse-springboot-patch-metadata/maven-metadata.xml (363 B at 4.3 kB/s)
[INFO] [PATCH] Resolved patch descriptor: /home/user/.m2/repository/org/jboss/redhat-fuse/fuse-springboot-patch-metadata/7.8.0.fuse-sb2-780032/fuse-springboot-patch-metadata-7.8.0.fuse-sb2-780032.xml
[INFO] [PATCH] Patch metadata found for org.jboss.redhat-fuse/fuse-springboot-bom/[7.8,7.9)
[INFO] [PATCH]  - patch contains 1 CVE fix
[INFO] [PATCH] Processing managed dependencies to apply CVE fixes...
(https://nvd.nist.gov/vuln/detail/CVE-2020-xyz, https://bugzilla.redhat.com/show_bug.cgi?id=42_
[INFO] [PATCH] - CVE-2020-xyz: Jetty can be configured to expose itself on port 8080
[INFO] [PATCH]   Applying change org.eclipse.jetty/jetty-*/[9.4,9.4.32) -> 9.4.32.v20200930
[INFO] [PATCH]    - managed dependency: org.eclipse.jetty/jetty-alpn-client/9.4.30.v20200611 -> 9.4.32.v20200930
...
[INFO] [PATCH]    - managed dependency: org.eclipse.jetty/jetty-openid/9.4.30.v20200611 -> 9.4.32.v20200930
[INFO] [PATCH]   Applying change org.eclipse.jetty.http2/http2-*/[9.4,9.4.32) -> 9.4.32.v20200930
[INFO] [PATCH]    - managed dependency: org.eclipse.jetty.http2/http2-client/9.4.30.v20200611 -> 9.4.32.v20200930
...
[INFO] [PATCH] Done in 635ms

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

跳过补丁

如果您不希望将特定补丁应用到项目，则 **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，您可以使用以下任一方法跳过补丁。

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

```
<build>
   <plugins>
      <plugin>
         <groupId>org.jboss.redhat-fuse</groupId>
         <artifactId>patch-maven-plugin</artifactId>
         <version>${version.org.jboss-redhat-fuse}</version>
         <extensions>true</extensions>
         <configuration>
            <skip>true</skip>
         </configuration>
      </plugin>
   </plugins>
</build>
```

- *运行 **mvn** 命令时或使用 **-DskipPatch** 选项，如下所示：*

```
$ mvn dependency:tree -DskipPatch
[INFO] Scanning for projects...
[INFO]
[INFO] ------< org.jboss.redhat-fuse:cve-dependency-management-module1 >-------
[INFO] Building cve-dependency-management-module1 7.8.0.fuse-sb2-780033
[INFO] --------------------------------[ jar ]---------------------------------
...
```

如上述输出中所示，patch-maven-plugin 没有调用，这会导致补丁不会应用到应用。

附录 A. 准备使用 MAVEN

本节概述了如何准备 Maven 以构建红帽 Fuse 项目，并介绍了用于定位 Maven 工件的 Maven 协调的概念。

A.1. 准备设置 MAVEN

Maven 是一个来自 Apache 的免费开源构建工具。通常，您可以使用 Maven 构建 Fuse 应用程序。

流程

1. 从 Maven 下载页面 下载最新版本的 Maven。
2. 确定您的系统已连接到互联网。

   在构建项目时，默认行为是 Maven 搜索外部存储库并下载所需的工件。Maven 查找可通过互联网访问的存储库。

   您可以更改此行为，以便 Maven 仅搜索位于本地网络上的存储库。也就是说，Maven 可以在离线模式下运行。在离线模式中，Maven 会在其本地存储库中查找工件。请参阅 第 A.3 节 “使用本地 Maven 存储库”。

A.2. 将红帽软件仓库添加到 MAVEN

要访问位于 Red Hat Maven 存储库中的工件，您需要将这些存储库添加到 Maven 的 settings.xml 文件中。Maven 在用户主目录的 .m2 目录中查找 settings.xml 文件。如果没有用户指定的 settings.xml 文件，Maven 将使用 M2_HOME/conf/ settings.xml 中的系统级 settings.xml 文件。

前提条件

您知道要在其中添加红帽软件仓库的 settings.xml 文件的位置。

流程

在 settings.xml 文件中，为红帽 软件仓库 添加软件仓库元素，如下例所示：

```
<?xml version="1.0"?>
```

```
<settings>

 <profiles>
  <profile>
   <id>extra-repos</id>
   <activation>
    <activeByDefault>true</activeByDefault>
   </activation>
   <repositories>
    <repository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
    <repository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
    <repository>
      <id>jboss-public</id>
      <name>JBoss Public Repository Group</name>
      <url>https://repository.jboss.org/nexus/content/groups/public/</url>
    </repository>
   </repositories>
   <pluginRepositories>
    <pluginRepository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </pluginRepository>
    <pluginRepository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </pluginRepository>
```

```
      <pluginRepository>
        <id>jboss-public</id>
        <name>JBoss Public Repository Group</name>
        <url>https://repository.jboss.org/nexus/content/groups/public</url>
      </pluginRepository>
    </pluginRepositories>
  </profile>
</profiles>

<activeProfiles>
  <activeProfile>extra-repos</activeProfile>
</activeProfiles>

</settings>
```

A.3. 使用本地 MAVEN 存储库

如果您在没有互联网连接的情况下运行容器，并且需要部署一个具有离线依赖项的应用程序，您可以使用 Maven 依赖项插件将应用的依赖项下载到 Maven 离线存储库中。然后，您可以将此自定义 Maven 离线存储库分发到没有互联网连接的机器。

流程

1. 在包含 pom.xml 文件的项目目录中，运行以下命令来下载 Maven 项目的存储库，如下所示：

```
mvn org.apache.maven.plugins:maven-dependency-plugin:3.1.0:go-offline -
Dmaven.repo.local=/tmp/my-project
```

   在本例中，构建项目所需的 Maven 依赖项和插件将下载到 /tmp/my-project 目录中。

2. 将此自定义 Maven 离线存储库在内部分发到没有互联网连接的任何机器。

A.4. 关于 MAVEN 工件和协调

在 Maven 构建系统中，基本构建块是一个 工件。构建后，工件的输出通常是一个存档，如 JAR 或 WAR 文件。

Maven 的一个关键方面是能够找到工件和管理它们之间的依赖关系。Maven 协调 是一组用于标识特定工件位置的值。基本协调有三个值，格式为：

groupId:artifactId:version

有时，Maven 通过 打包值或使用打包 值和 分类器 值增加一个基本的协调。Maven 协调可以具有以下形式之一：

```
groupId:artifactId:version
groupId:artifactId:packaging:version
groupId:artifactId:packaging:classifier:version
```

以下是值的描述：

groupdId

定义工件名称的范围。您通常使用所有或部分软件包名称作为组 ID。例如，org.fusesource.example。

artifactId

定义相对于组 ID 的工件名称。

version

指定工件的版本。版本号最多可有四个部分：n.n.n.n，其中版本号的最后一部分可以包含非数字字符。例如，1.0-SNAPSHOT 的最后一部分是字母数字字符子字符串 0-SNAPSHOT。

打包

定义构建项目时生成的打包实体。对于 OSGi 项目，打包是 捆绑包。默认值为 jar。

分类器

可让您区分从同一 POM 构建但具有不同内容的工件。

工件的 POM 文件中的元素定义工件的组 ID、工件 ID、打包和版本，如下所示：

```
<project ... >
  ...
  <groupId>org.fusesource.example</groupId>
  <artifactId>bundle-demo</artifactId>
  <packaging>bundle</packaging>
  <version>1.0-SNAPSHOT</version>
  ...
</project>
```

要定义对上述工件的依赖项，您可以在 POM 文件中添加以下 dependencies 元素：

```
<project ... >
  ...
  <dependencies>
    <dependency>
      <groupId>org.fusesource.example</groupId>
      <artifactId>bundle-demo</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
  ...
</project>
```

注意

不需要在前面的依赖项中指定 bundle 软件包类型，因为捆绑包只是特定类型的 JAR 文件，jar 是默认的 Maven 软件包类型。但是，如果您需要在依赖项中明确指定打包类型，您可以使用 type 元素。

附录 B. SPRING BOOT MAVEN 插件

Spring Boot Maven 插件在 Maven 中提供 Spring Boot 支持，并允许您打包可执行文件 jar 或 war 归档并运行应用程序 原位。

B.1. SPRING BOOT MAVEN 插件目标

Spring Boot Maven 插件包括以下目标：

- spring-boot:run 运行 Spring Boot 应用程序。
- spring-boot:repackage 将您的 .jar 和 .war 文件重新打包为可执行文件。
- spring-boot:start 和 spring-boot:stop 均用于管理 Spring Boot 应用程序的生命周期。
- spring-boot:build-info 生成可由 Actuator 使用的构建信息。

B.2. 使用 SPRING BOOT MAVEN 插件

您可以在以下位置找到有关如何使用 Spring Boot 插件的通用说明：https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/#using。以下示例演示了为 Spring Boot 使用 spring-boot-maven-plugin。

- Spring Boot 2 示例

有关 Spring Boot Maven 插件的更多信息，请参阅 https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/ 链接。

B.2.1. 为 Spring Boot 2 使用 Spring Boot Maven 插件

以下示例演示了为 Spring Boot 2 使用 spring-boot-maven-plugin。

Example

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.redhat.fuse</groupId>
  <artifactId>spring-boot-camel</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

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

    <!-- maven plugin versions -->
    <maven-compiler-plugin.version>3.7.0</maven-compiler-plugin.version>
    <maven-surefire-plugin.version>2.19.1</maven-surefire-plugin.version>
  </properties>


  <build>
    <defaultGoal>spring-boot:run</defaultGoal>

    <plugins>
      <plugin>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <version>${fuse.bom.version}</version>
        <executions>
          <execution>
            <goals>
              <goal>repackage</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>

  <repositories>
    <repository>
      <id>redhat-ga-repository</id>
      <url>https://maven.repository.redhat.com/ga</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
    <repository>
      <id>redhat-ea-repository</id>
      <url>https://maven.repository.redhat.com/earlyaccess/all</url>
```

```
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>

    <pluginRepositories>
        <pluginRepository>
            <id>redhat-ga-repository</id>
            <url>https://maven.repository.redhat.com/ga</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
        <pluginRepository>
            <id>redhat-ea-repository</id>
            <url>https://maven.repository.redhat.com/earlyaccess/all</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
    </pluginRepositories>
</project>
```