Red Hat build of Apache Camel Extensions for Quarkus 2.13

Camel Extensions for Quarkus 入门

Camel Extensions for Quarkus 入门

Last Updated: 2023-09-19

Red Hat build of Apache Camel Extensions for Quarkus 2.13 Camel Extensions for Quarkus 入门

Camel Extensions for Quarkus 入门

法律通告

摘要

本指南介绍了 Camel Extensions for Quarkus，并解释了使用 Camel Extensions for Quarkus 创建和部署应用程序的各种方法。

目录

前言

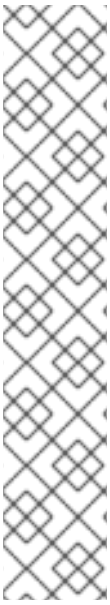
使开源包含更多

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

第 1 章 CAMEL EXTENSIONS FOR QUARKUS 入门

本指南介绍了 Camel Extensions for Quarkus，这是创建项目以及如何开始使用 Camel Extensions for Quarkus 构建应用程序的各种方法：

- 第 1.1 节 “Camel Extensions for Quarkus 概述”
- 第 1.2 节 “工具”
- 第 1.3 节 “使用 Camel Extensions for Quarkus 构建第一个项目”



注意

红帽提供了 Maven 存储库，用于托管我们随我们的产品附带的内容。这些软件仓库可从软件下载页面下载。

对于 Camel Extensions for Quarkus，需要以下软件仓库：

- rhi-camel-extensions-for-quarkus

在这个发行版本中，在断开连接的环境中（离线模式）中运行 Camel Extensions for Quarkus 不被支持，但不支持在离线模式下构建 Camel Extensions for Quarkus。

有关将 Apache Maven 存储库用于 Camel Quarkus 的详情，请参考 Chapter 2.4。配置 Quarkus Maven 存储库

第 2.2 章：" *使用 Apache Maven 开发并编译 Quarkus 应用程序指南中的下载并配置 Quarkus Maven 存储库*"

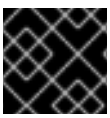
1.1. CAMEL EXTENSIONS FOR QUARKUS 概述

Camel Extensions for Quarkus 将 Apache Camel 的集成功能及其主要组件库引入 Quarkus 运行时。

使用 Camel Extensions for Quarkus 的好处包括：

- 允许用户利用 Quarkus 提供的性能优势、developer joy 和容器。
- 为许多 Apache Camel 组件提供 Quarkus 扩展。
- 利用 Camel 3 中进行的许多性能改进，从而降低了内存占用量，从而减少对反映和启动时间的依赖。
- 您可以使用 Java DSL 定义 Camel 路由。

1.2. 工具



重要

红帽不提供对这些开发人员工具的支持。

1.2.1. IDE 插件

Quarkus 对于大多数流行的开发 IDE 的插件，提供 Quarkus 语言支持、代码/配置完成、项目创建向导等。该插件可在每个相应的 IDE 市场中找到。

- Eclipse 插件
- IntelliJ 插件
- VS Code 扩展

查看插件文档，以了解如何为您的首选 IDE 创建项目。

1.2.2. Camel 内容帮助

以下插件在编辑 Camel 路由和 **application.properties** 时提供对内容协助的支持：

- VS Code Language support for Camel - Camel extension pack 的一部分
- Apache Camel 的 debug Adapter，在本地调试 使用 Java、YAML 或 XML 编写的 Camel 集成。
  - 有关 开发支持范围的更多信息，请参阅开发支持覆盖范围
- Camel 的 Eclipse 桌面语言支持 - WebLogic 工具和 CodeReady Studio的一部分
- Apache Camel IDEA 插件 （并不总是最新）
- 支持语言服务器协议的其他 IDE 用户可以选择手动安装和配置 Camel 语言服务器

1.3. 使用 CAMEL EXTENSIONS FOR QUARKUS 构建第一个项目

1.3.1. 概述

您可以使用 code.quarkus.redhat.com 生成 Quarkus Maven 项目，该项目会自动添加和配置要在应用程序中使用的扩展。

本节介绍了使用 Camel Extensions for Quarkus 创建 Quarkus Maven 项目的过程，包括：

- 使用 code.quarkus.redhat.com创建框架应用程序
- 添加简单的 Camel 路由
- 探索应用程序代码
- 以开发模式编译应用程序
- 测试应用程序

1.3.2. 生成框架应用程序

项目可以在 code.quarkus.redhat.com 中引导和生成。Camel Extensions for Quarkus 扩展位于"Integration"标题下。

使用 'search' 字段查找您需要的扩展。

选择您要使用的组件扩展，并点击"Generate your application"来下载基本框架项目。另外，还可以选择将项目直接推送到 GitHub。

有关使用 **code.quarkus.redhat.com** 生成 Quarkus Maven 项目的更多信息，请参阅 *Getting Started with Quarkus 指南中的使用 code.quarkus.redhat.com 创建 Quarkus Maven 项目*。

流程

1. 使用 *code.quarkus.redhat.com* 网站，为这个示例选择以下扩展：
   - **camel-quarkus-rest**
   - **camel-quarkus-jackson**

注意

您不应该按照上述步骤的最后步骤对应用程序进行编译，因为您将作为本指南的一部分执行该任务。

2. 进入您在上一步中提取生成的项目文件的目录：

```
$ cd <directory_name>
```

1.3.3. 探索应用程序代码

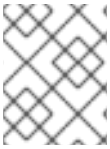
应用程序有两个编译的依赖关系，这些依赖项在 **com.redhat.quarkus.platform:quarkus-camel-bom** 中进行管理，这些依赖项在 < **dependencyManagement>** 中导入：

pom.xml

```
<quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
<quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
<quarkus.platform.version>
    <!-- The latest 2.13.x version from
https://maven.repository.redhat.com/ga/com/redhat/quarkus/platform/quarkus-bom -->
</quarkus.platform.version>

...

<dependency>
    <groupId>${quarkus.platform.group-id}</groupId>
    <artifactId>${quarkus.platform.artifact-id}</artifactId>
    <version>${quarkus.platform.version}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
<dependency>
    <groupId>${quarkus.platform.group-id}</groupId>
    <artifactId>quarkus-camel-bom</artifactId>
    <version>${quarkus.platform.version}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```



注意

有关 BOM 依赖项管理的更多信息，请参阅使用 Camel Extensions for Quarkus 开发应用程序

应用程序由 **src/main/resources/application.properties** 中定义的属性进行配置，例如：**camel.context.name** 可以在此处设置。

1.3.4. 添加简单的 Camel 路由

流程

1. 在 **src/main/java/org/acme/** 子文件夹中创建一个名为 **Routes.java** 的文件。
2. 添加 Camel Rest 路由，如以下代码片段所示：

Routes.java

```
package org.acme;

import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.concurrent.CopyOnWriteArrayList;

import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.model.rest.RestBindingMode;

import io.quarkus.runtime.annotations.RegisterForReflection;

public class Routes extends RouteBuilder {
    private final List<Fruit> fruits = new CopyOnWriteArrayList<>(Arrays.asList(new
Fruit("Apple")));

    @Override
    public void configure() throws Exception {
        restConfiguration().bindingMode(RestBindingMode.json);

        rest("/fruits")
                .get()
                .route()
                .setBody(e -> fruits)
                .endRest()

                .post()
                .type(Fruit.class)
                .route()
                .process().body(Fruit.class, (Fruit f) -> fruits.add(f))
                .endRest();

    }

    @RegisterForReflection // Let Quarkus register this class for reflection during the native
build
    public static class Fruit {
        private String name;

        public Fruit() {
        }
```

```
        public Fruit(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        @Override
        public int hashCode() {
            return Objects.hash(name);
        }

        @Override
        public boolean equals(Object obj) {
            if (this == obj)
                return true;
            if (obj == null)
                return false;
            if (getClass() != obj.getClass())
                return false;
            Fruit other = (Fruit) obj;
            return Objects.equals(name, other.name);
        }

    }

}
```

1.3.5. 开发模式

```
$ mvn clean compile quarkus:dev
```

此命令编译项目，启动应用程序，并允许 Quarkus 工具监控工作区中的更改。项目中的任何修改都会在正在运行的应用程序中自动生效。

在浏览器中检查应用程序，例如 http://localhost:8080/fruits，用于 **rest-json** 示例。

如果您更改了应用程序代码，例如，将 'Apple' 更改为 'Orange'，则应用程序会自动更新。要查看应用的更改，请刷新浏览器。

有关 开发模式的详情， 请参阅 Quarkus 文档开发模式部分。

1.3.6. 测试

1.3.6.1. JVM 模式

要测试在 JVM 模式中创建的 Camel Rest 路由，请添加测试类，如下所示：

流程

1. 在 **src/test/java/org/acme/** 子文件夹中创建一个名为 **RoutesTest.java** 的文件。
2. 添加 **RoutesTest** 类，如以下代码片段所示：

RoutesTest.java

```
package org.acme;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;
import org.hamcrest.Matchers;

@QuarkusTest
public class RoutesTest {

    @Test
    public void testFruitsEndpoint() {

        /* Assert the initial fruit is there */
        given()
                .when().get("/fruits")
                .then()
                .statusCode(200)
                .body(
                        "$.size()", Matchers.is(1),
                        "name", Matchers.contains("Orange"));

        /* Add a new fruit */
        given()
                .body("{\"name\": \"Pear\"}")
                .header("Content-Type", "application/json")
                .when()
                .post("/fruits")
                .then()
                .statusCode(200);

        /* Assert that pear was added */
        given()
                .when().get("/fruits")
                .then()
                .statusCode(200)
                .body(
                        "$.size()", Matchers.is(2),
                        "name", Matchers.contains("Orange", "Pear"));
    }

}
```

JVM 模式测试在 **test** Maven 阶段由 **maven-surefire-plugin** 运行：

```
$ mvn clean test
```

1.3.6.2. 原生模式

要测试以原生模式创建的 Camel Rest 路由，请添加测试类，如下所示：

流程

1. 在 **src/test/java/org/acme/** 子文件夹中，创建一个名为 **NativeRoutesIT.java** 的文件。
2. 添加 **NativeRoutesIT** 类，如以下代码片段所示：

   NativeRoutesIT.java

   ```
   package org.acme;

   import io.quarkus.test.junit.NativeImageTest;

   @NativeImageTest
   public class NativeRoutesIT extends RoutesTest {

       // Execute the same tests but in native mode.
   }
   ```

   在验证阶段，通过 **maven-failsafe-plugin** 验证 原生模式测试。

3. 传递 **native** 属性来激活运行它们的配置集：

   ```
   $ mvn clean verify -Pnative
   ```

提示

如需了解更多详细信息，以及如何使用 **CamelTestSupport** 测试风格，请参阅 测试 Camel Quarkus 扩展。

1.3.7. 打包并运行应用程序

1.3.7.1. JVM 模式

流程

1. 运行 **mvn 软件包**，以准备在库存 JVM 上运行的精简 **jar** ：

   ```
   $ mvn clean package
   $ ls -lh target/quarkus-app
   ...
   -rw-r--r--. 1 user user 238K Oct 11 18:55  quarkus-run.jar
   ...
   ```

注意

瘦 **jar** 仅包含应用程序代码。您还需要 **target/quarkus-app/lib** 中的依赖项来运行它。

2. 运行 jar，如下所示：

```
$ java -jar target/quarkus-app/quarkus-run.jar
...
[io.quarkus] (main) Quarkus started in 1.163s. Listening on: http://[::]:8080
```

注意

引导时间应该大约为一秒钟。

1.3.7.2. 原生模式

流程

要准备原生可执行文件，请执行以下操作：

1. 运行命令 **mvn clean package -Pnative** ：

```
$ mvn clean package -Pnative
$ ls -lh target
...
-rwxr-xr-x. 1 user user  46M Oct 11 18:57  code-with-quarkus-1.0.0-SNAPSHOT-runner
...
```

注意

运行程序 没有 **.jar** 扩展，并且设置了 **x** （可执行）权限。您可以直接运行：

```
$ ./target/*-runner
...
[io.quarkus] (main) Quarkus started in 0.013s. Listening on: http://[::]:8080
...
```

应用程序以 13 毫秒开始。

2. 使用 **ps -o rss,command -p $(pgrep code-with)** 命令查看内存用量：

```
$ ps -o rss,command -p $(pgrep code-with)
  RSS COMMAND
65852 ./target/code-with-quarkus-1.0.0-SNAPSHOT-runner
```

应用程序使用 65 MB 内存。

提示

有关准备原生可执行文件的更多信息，请参阅 Compiling *your Quarkus application to native* executables 指南中的 Producing a native executable。

提示

Quarkus 原生可执行文件指南 包含更多详细信息，包括 创建容器镜像的步骤。

1.4. 测试 CAMEL QUARKUS 扩展

测试提供了一种好的方法来确保 Camel 路由在一段时间内的行为正常。如果您还没有，请阅读 Camel Quarkus 用户指南，首先步骤和 Quarkus 文档链接：测试应用程序 部分。

在 Quarkus 中测试路由的最简单方法是编写本地集成测试。它具有涵盖 JVM 和原生模式的优势。

在 JVM 模式中，您可以使用 **CamelTestSupport** 测试风格。

1.4.1. 在 JVM 模式下运行

在 JVM 模式中，使用 **@QuarkusTest** 注释引导 Quarkus，并在 **@Test** 逻辑执行前启动 Camel 路由。

例如：

```
import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

@QuarkusTest
class MyTest {
    @Test
    public void test() {
        // Use any suitable code that sends test data to the route and then assert outcomes
        ...
    }
}
```

提示

您可以在 Camel Quarkus 源中找到一个示例实现：

- *MessageTest.java*

1.4.2. 以原生模式运行

注意

始终测试您的应用程序是否以原生模式对所有支持的扩展工作。

您可以通过从相应的 JVM 模式类继承逻辑来重复使用为 JVM 模式定义的测试逻辑。

*添加 **@QuarkusIntegrationTest** 注释，以指示 Quarkus JUnit 扩展，以在测试到原生镜像下编译应用程序，并在运行测试前启动它。*

```
import io.quarkus.test.junit.QuarkusIntegrationTest;

@QuarkusIntegrationTest
class MyIT extends MyTest {
    ...
}
```

提示

您可以在 Camel Quarkus 源中找到一个示例实现：

- MessageRecordIT.java

1.4.3. @QuarkusTest 和 @QuarkusIntegrationTest之间的区别

原生可执行文件不需要 JVM 运行，且无法在 JVM 中运行，因为它是原生代码，而不是字节码。

对原生代码进行编译测试时没有点，以便使用传统的 JVM 运行。

这意味着测试和应用程序之间的通信必须经过网络(HTTP/REST)，或通过监视文件系统（例如，日志文件）或其他进程间通信。

1.4.3.1. JVM 模式中的 @QuarkusTest

在 JVM 模式中，使用 **@QuarkusTest** 注解的测试会在与测试下应用程序相同的 JVM 中执行。

这意味着，您可以使用 **@Inject** 将应用中的 Bean 添加到测试代码中。

您还可以使用 **@javax.enterprise.inject.Alternative** 和 **@javax.annotation.Priority** 来定义新的 Bean 甚至覆盖应用程序中的 Bean。

1.4.3.2. @QuarkusIntegrationTest in native mode

在原生模式中，使用 **@QuarkusIntegrationTest** 注解的测试在独立于正在运行的原生应用程序的 JVM 中执行。

QuarkusIntegrationTest 提供了通过 **@QuarkusTest** 不提供的额外功能：

- 在 JVM 模式中，您可以启动并测试 Quarkus 构建生成的可运行的应用程序 JAR。
- 在原生模式中，您可以启动并测试 Quarkus 构建生成的原生应用程序。
- 如果您将容器镜像添加到构建中，容器会启动，并测试对其执行。

有关 **QuarkusIntegrationTest** 的更多信息，请参阅 Quarkus 测试指南。

1.4.4. 使用外部服务测试

1.4.4.1. testcontainers

有时您的应用程序需要访问一些外部资源，如消息传递代理、数据库或其他服务。

如果容器镜像可用于感兴趣的服务，您可以使用 Testcontainers 在测试过程中启动和配置服务。

1.4.4.1.1. 使用 QuarkusTestResourceLifecycleManager传递配置数据

要使应用正常工作，通常需要在启动之前将连接配置数据（主机、端口、用户、远程服务密码）传递给应用程序。

在 Quarkus 生态系统中，**QuarkusTestResourceLifecycleManager** 提供了这一目的。

您可以使用 **start** （） 方法启动一个或多个 Testcontainers，并以 映射 的形式从方法返回连接配置。

然后，此映射的条目以不同的方式传递给应用程序，具体取决于模式：

- *原生模式：命令行(**-Dkey=value**)*
- *JVM 模式：特殊的 MicroProfile 配置提供程序*

注意

*这些设置的优先级高于 **application.properties** 文件中的设置。*

```
import java.util.Map;
import java.util.HashMap;

import io.quarkus.test.common.QuarkusTestResourceLifecycleManager;
import org.testcontainers.containers.GenericContainer;
import org.testcontainers.containers.wait.strategy.Wait;

public class MyTestResource implements QuarkusTestResourceLifecycleManager {

    private GenericContainer myContainer;

    @Override
    public Map<String, String> start() {
        // Start the needed container(s)
        myContainer = new GenericContainer(...)
                .withExposedPorts(1234)
                .waitingFor(Wait.forListeningPort());

        myContainer.start();

        // Pass the configuration to the application under test
        return new HashMap<>() {{
                put("my-container.host", container.getContainerIpAddress());
                put("my-container.port", "" + container.getMappedPort(1234));
        }};
    }

    @Override
    public void stop() {
        // Stop the needed container(s)
        myContainer.stop();
        ...
    }
```

*使用 **@QuarkusTestResource** 引用测试类中定义的 test 资源：*

```
import io.quarkus.test.common.QuarkusTestResource;
import io.quarkus.test.junit.QuarkusTest;

@QuarkusTest
@QuarkusTestResource(MyTestResource.class)
class MyTest {
    ...
}
```

提示

您可以在 Camel Quarkus 源中找到一个示例实现：

- NatsTestResource.java

1.4.4.2. WireMock

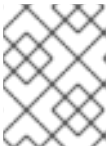
您可以使用测试连接到实时端点，例如，如果它们不可用、不可靠或昂贵，您可以与第三方服务和 API 进行根 HTTP 交互。

您可以使用 WireMock 来模拟和记录 HTTP 交互。它在整个 Camel Quarkus 测试套件中广泛用于各种组件扩展。

1.4.4.2.1. 设置 WireMock

流程

1. 设置 WireMock 服务器。



注意

在 test 下配置 Camel 组件以通过 WireMock 代理传递任何 HTTP 交互非常重要。您可以通过配置决定 API 端点 URL 的组件属性来实现此目的。

```
import static com.github.tomakehurst.wiremock.client.WireMock.aResponse;
import static com.github.tomakehurst.wiremock.client.WireMock.get;
import static com.github.tomakehurst.wiremock.client.WireMock.urlEqualTo;
import static
com.github.tomakehurst.wiremock.core.WireMockConfiguration.wireMockConfig;

import java.util.HashMap;
import java.util.Map;

import com.github.tomakehurst.wiremock.WireMockServer;

import io.quarkus.test.common.QuarkusTestResourceLifecycleManager;

public class WireMockTestResource implements QuarkusTestResourceLifecycleManager {

    private WireMockServer server;

    @Override
    public Map<String, String> start() {
        // Setup & start the server
        server = new WireMockServer(
            wireMockConfig().dynamicPort()
        );
        server.start();

        // Stub a HTTP endpoint. Note that WireMock also supports a record and playback mode
        // http://wiremock.org/docs/record-playback/
        server.stubFor(
            get(urlEqualTo("/api/greeting"))
```

```
                .willReturn(aResponse()
                    .withHeader("Content-Type", "application/json")
                    .withBody("{\"message\": \"Hello World\"}")));

        // Ensure the camel component API client passes requests through the WireMock proxy
        Map<String, String> conf = new HashMap<>();
        conf.put("camel.component.foo.server-url", server.baseUrl());
        return conf;
    }

    @Override
    public void stop() {
        if (server != null) {
            server.stop();
        }
    }
}
```

注意

有时，这不太简单，还需要一些额外的工作来配置 API 客户端库。例如，对于 Twilio。

2. *确保您的测试类具有* ***@QuarkusTestResource*** *注解，并将适当的 test 资源类指定为值。*

```
import io.quarkus.test.common.QuarkusTestResource;
import io.quarkus.test.junit.QuarkusTest;

@QuarkusTest
@QuarkusTestResource(WireMockTestResource.class)
class MyTest {
    ...
}
```

在所有测试完成后，WireMock 服务器在所有测试执行和关闭之前启动。

提示

您可以在 Camel Quarkus 集成测试源树中找到一个示例实现：

- Geocoder.

1.4.5. 使用 ***CamelQuarkusTestSupport***

从 Camel Quarkus 2.13.0 开始，您可以使用 ***CamelQuarkusTestSupport*** *进行测试。它是* ***CamelTestSupport*** *的替代品。*

注意

这只能在 JVM 模式下工作。

1.4.5.1. 在 JVM 模式中使用 ***CamelQuarkusTestSupport*** *测试*

将以下依赖项添加到您的模块（可以在 测试范围内引用 ）：

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-junit5</artifactId>
    <scope>test</scope>
</dependency>
```

您可以在测试中使用 **CamelQuarkusTestSupport**，如下所示：

```
@QuarkusTest
@TestProfile(SimpleTest.class) //necessary only if "newly created" context is required for the test (worse performance)
public class SimpleTest extends CamelQuarkusTestSupport {
    ...
}
```

1.4.5.2. 使用 CamelQuarkusTestSupport时的限制

当使用 'CamelQuarkusTestSupport 时，有几个限制：

1.4.5.2.1. Methods

某些方法无法执行。使用以 **do** 开头的新方法：

未执行	改为使用
afterAll	**doAfterAll**
afterEach	**doAfterEach**
afterTestExecution	**doAfterTestExecution**
beforeAll	**doBeforeAll**
beforeEach	**doBeforeEach**

注意

如果您使用 **@TestInstance (TestInstance.Lifecycle.PER_METHOD)**，则 **doAfterConstruct** 表示每个测试前的回调。这与 **All** 之前 的不同。

1.4.5.2.2. 注解

您必须使用 **@io.quarkus.test.junit.QuarkusTest** 注解测试类，并扩展 **org.apache.camel.quarkus.test.CamelQuarkusTestSupport**。

1.4.5.2.3. 启动和停止

- 您不能在单个应用程序生命周期中停止并重启相同的 **CamelContext** 实例。您可以调用 **CamelContext.stop** （），但 **CamelContext.start** （） 将无法正常工作。

- 在 测试时，CamelContext 通常也会绑定到启动和停止应用程序。
- test 下的应用程序会针对给定 Maven/Gradle 模块的所有测试类启动一次。Quarkus JUnit 扩展控制应用程序的启动和停止。您必须明确告知应用程序停止。

1.4.5.2.4. 重启应用程序

要强制 Quarkus JUnit 扩展来重新启动给定测试类的应用程序和 **CamelContext**，您需要将唯一的 **@io.quarkus.test.junit.TestProfile** 分配给该类。

具体步骤请参阅 Quarkus 文档中的 测试不同的配置集。

要实现类似效果，您还可以使用 **@io.quarkus.test.common.QuarkusTestResource**。

1.4.5.2.5. Bean 生产环境

Camel Quarkus 在构建阶段执行 Bean 的生产环境。由于测试是一起构建的，因此排除行为在 **CamelQuarkusTestSupport** 中实现。如果一个测试中使用特定类型的和名称的制作者，则实例将与其余测试相同。

1.4.5.2.6. JUnit Jupiter 回调可能无法正常工作

这些 JUnit Jupiter 回调和注解可能无法正常工作：

回调	注解
BeforeEachCallback	@BeforeEach
AfterEachCallback	@AfterEach
AfterAllCallback	@AfterAll
BeforeAllCallback	@BeforeAll
BeforeTestExecutionCallback	
AfterTestExecutionCallback	

如需更多信息，请参阅通过 QuarkusTest114 回调文档进行增强。

1.4.5.2.7. 使用 recommendationsWith

当 **recommendations** With 设为 true 时，所有未推荐的路由都不会启动。您必须执行 **CamelQuarkusTestSupport.startRouteDefinitions** （） 方法才能启动它们。

1.4.5.2.8. 使用 @Produces

使用带有覆盖方法 **createRouteBuilder** （） 的 **@Produces**。**@Produces** 和 **RouteBuilder** （） 的组合可能无法正常工作。

1.4.5.2.9. 配置路由

*要配置应用程序中的哪些路由(**src/main/java**)以包含或排除，您可以使用以下内容：*

- ***quarkus.camel.routes-discovery.exclude-patterns***
- ***quarkus.camel.routes-discovery.include-patterns***

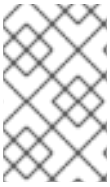
如需了解更多详细信息，请参阅 核心文档。

第 2 章 部署 QUARKUS 应用程序

您可以使用以下构建策略之一在 OpenShift 上部署 Quarkus 应用程序：

- Docker 构建
- S2I Binary
- 源 S2I

有关这些构建策略的详情，请参阅 Chapter 1。OpenShift 构建策略和 Quarkus （将 Quarkus 应用部署到 OpenShift 指南）。



注意

OpenShift Docker 构建策略是首选的构建策略，它支持用于 JVM 的 Quarkus 应用程序以及编译到原生可执行文件的 Quarkus 应用程序。您可以使用 **quarkus.openshift.build-strategy** 属性来配置部署策略。

第 3 章 例子

下表中列出的快速入门示例可以从 Camel Quarkus Examples Git 存储库克隆或下载。

示例数：1

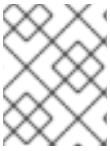
示例	描述
使用 Bindy 和 FTP 的文件消费者	显示如何使用 CSV 文件，marshal & unmarshal 的数据通过 FTP 发送。

3.1. 文件消费者快速入门示例

您可以从 Camel Quarkus Examples Git 存储库下载或克隆快速入门。这个示例位于 **file-bindy-ftp** 目录中。

提取 zip 文件的内容或将存储库克隆到本地文件夹，例如一个名为 **quickstarts** 的新文件夹。

您可以使用命令行在本地机器上以开发模式运行此示例。使用开发模式，您可以快速迭代开发中的集成，并快速获得代码反馈。如需更多详细信息，请参阅 Camel Quarkus 用户指南中的开发模式部分。



注意

如果您需要配置容器资源限值或启用 Quarkus Kubernetes 客户端信任自签名证书，您可以在 **src/main/resources/application.properties** 文件中找到这些配置选项。

先决条件

- 具有 **OpenShift** 集群的 集群管理员访问权限。
- 您可以访问 SFTP 服务器，并且已在应用属性配置文件中设置服务器属性（以 **ftp** 前缀）：**src/main/resources/application.properties**。

流程

1. 使用 Maven 以开发模式构建示例应用程序：

```
$ cd quickstarts/file-bindy-ftp
$ mvn clean compile quarkus:dev
```

应用程序每 10 秒触发计时器组件，生成一些随机"书"数据，并在具有 100 个条目的临时目录中创建 CSV 文件。控制台中显示以下信息：

```
[route1] (Camel (camel-1) thread #3 - timer://generateBooks) Generating randomized books CSV data
```

接下来，CSV 文件由文件消费者读取，而 Bindy 则用于将单个数据行合并到书签对象中：

```
[route2] (Camel (camel-1) thread #1 - file:///tmp/books) Reading books CSV data from 89A0EE24CB03A69-0000000000000000
```

书签对象的后续集合被分成单个项目，并根据 genre 属性聚合：

```
[route3] (Camel (camel-1) thread #0 - AggregateTimeoutChecker) Processed 34 books for genre 'Action'
[route3] (Camel (camel-1) thread #0 - AggregateTimeoutChecker) Processed 31 books for genre 'Crime'
[route3] (Camel (camel-1) thread #0 - AggregateTimeoutChecker) Processed 35 books for genre 'Horror'
```

最后，聚合的书集合被取消回 CSV 格式，并上传到测试 FTP 服务器。

```
[route4] (Camel (camel-1) thread #2 - seda://processed) Uploaded books-Action-89A0EE24CB03A69-0000000000000069.csv
[route4] (Camel (camel-1) thread #2 - seda://processed) Uploaded books-Crime-89A0EE24CB03A69-0000000000000069.csv
[route4] (Camel (camel-1) thread #2 - seda://processed) Uploaded books-Horror-89A0EE24CB03A69-0000000000000069.csv
```

2. 要以 JVM 模式运行应用程序，请输入以下命令：

```
$ mvn clean package -DskipTests
$ java -jar target/*-runner.jar
```

3. 您可以输入以下命令将示例应用程序构建和部署到 OpenShift 中：

```
$ mvn clean package -DskipTests -Dquarkus.kubernetes.deploy=true
```

4. 检查 pod 是否正在运行：

```
$oc get pods

NAME                                    READY   STATUS      RESTARTS   AGE
camel-quarkus-examples-file-bindy-ftp-1-d72mb   1/1     Running     0          5m15s
ssh-server-deployment-5f6f685658-jtr9n          1/1     Running     0          5m28s
```

5. 可选：输入以下命令来监控应用程序日志：

```
oc logs -f camel-quarkus-examples-file-bindy-ftp-5d48f4d85c-sjl8k
```

其他资源

- 使用 Camel Extensions for Quarkus 开发应用程序
- Camel Quarkus 用户指南