Red Hat build of Apache Camel 4.4

使用红帽构建的 Apache Camel for Quarkus 开发应用程序

使用红帽构建的 Apache Camel for Quarkus 开发应用程序

Last Updated: 2024-06-29

Red Hat build of Apache Camel 4.4 使用红帽构建的 Apache Camel for Quarkus 开发应用程序

使用红帽构建的 Apache Camel for Quarkus 开发应用程序

法律通告

摘要

本指南适用于在红帽构建的 Apache Camel for Quarkus 上编写 Camel 应用程序的开发人员。

目录

前言

提供有关红帽构建的 APACHE CAMEL 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

流程

1. 点击以下链接 来创建票据
2. 在 Summary 中输入问题的简短描述。
3. 在 Description 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
4. 点 Submit 创建问题，并将问题路由到适当的文档团队。

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。详情请查看 CTO Chris Wright 的信息。

第 1 章 使用红帽构建的 APACHE CAMEL FOR QUARKUS 开发应用程序简介

本指南适用于在红帽构建的 Apache Camel for Quarkus 上编写 Camel 应用程序的开发人员。

Red Hat build of Apache Camel for Quarkus 支持的 Camel 组件有一个关联的红帽构建的 Apache Camel for Quarkus 扩展。有关此发行版本支持的红帽构建的 Apache Camel for Quarkus 扩展的更多信息，请参阅 Red Hat build of Apache Camel for Quarkus Extensions 指南。

第 2 章 依赖项管理

特定的 Red Hat build of Apache Camel for Quarkus 发行版本应该只适用于特定的 Quarkus 发行版本。

2.1. 用于启动新项目的 QUARKUS 工具

在新项目中获取依赖项版本的最简单且最简单的方法是使用 Quarkus 工具之一：

- code.quarkus.redhat.com - an online project generator,
- Quarkus Maven 插件

这些工具允许您选择扩展并构建新的 Maven 项目。

提示

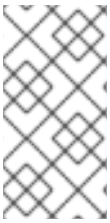
可用的扩展范围跨越 Quarkus Core、Camel Quarkus 和其他几个第三方参与项目，如 Hazelcast、Cassandra、Kogito 和 OptaPlanner。

生成的 **pom.xml** 类似如下：

```
<project>
  ...
  <properties>
    <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 3.8.x version from
https://maven.repository.redhat.com/ga/com/redhat/quarkus/platform/quarkus-bom -->
    </quarkus.platform.version>
    ...
  </properties>
  <dependencyManagement>
    <dependencies>
      <!-- The BOMs managing the dependency versions -->
      <dependency>
        <groupId>${quarkus.platform.group-id}</groupId>
        <artifactId>quarkus-bom</artifactId>
        <version>${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>
    </dependencies>
  </dependencyManagement>

  <dependencies>
    <!-- The extensions you chose in the project generator tool -->
```

```
    <dependency>
      <groupId>org.apache.camel.quarkus</groupId>
      <artifactId>camel-quarkus-sql</artifactId>
      <!-- No explicit version required here and below -->
    </dependency>
    ...
  </dependencies>
  ...
</project>
```



注意

BOM 代表 "Bill of Materials" - 它是 **pom.xml**，其主要目的是管理工件版本，以便最终用户在其项目中导入 BOM 时不需要处理哪些特定版本应该一起工作。换句话说，在 **pom.xml** 的 &**lt;depependencyManagement** > 部分中导入了一个 BOM，您可以避免为给定 BOM 管理的依赖项指定版本。

哪些特定的 BOM 最终在 **pom.xml** 文件中取决于您在生成器工具中选择的扩展。生成器工具需要注意选择最小一致的集合。

如果您选择稍后添加扩展，该扩展不由 **pom.xml** 文件中的任何 BOM 管理，则不需要手动搜索适当的 BOM。

使用 **quarkus-maven-plugin**，您可以选择扩展，工具则根据需要添加适当的 BOM。您还可以使用 **quarkus-maven-plugin** 升级 BOM 版本。

com.redhat.quarkus.platform BOMs 相互一致，这意味着如果工件在多个 BOM 中管理，它总是使用相同的版本进行管理。这使应用程序开发人员不需要注意可能来自不同独立项目的单个工件的兼容性。

2.2. 与其他 BOM 结合使用

将 **camel-quarkus-bom** 与任何其他 BOM 结合使用时，请仔细考虑导入它们的顺序，因为导入顺序定义了优先级。

例如，如果在 **camel-quarkus-bom** 之前导入 **my-foo-bom**，则 **my-foo-bom** 中定义的版本将具有优先权。这可能不是您想要的，具体取决于 **my-foo-bom** 和 **camel-quarkus-bom** 之间是否存在重叠，具体取决于那些具有较高优先级的版本是否与 **camel-quarkus-bom** 中管理的其余工件一起工作。

第 3 章 定义 CAMEL 路由

红帽构建的 Apache Camel for Quarkus 支持 Java DSL 语言来定义 Camel 路由。

3.1. JAVA DSL

扩展 **org.apache.camel.builder.RouteBuilder** 并使用 fluent builder 方法，这是定义 Camel 路由的最常见方法。以下是使用计时器组件的路由的简单示例：

```
import org.apache.camel.builder.RouteBuilder;

public class TimerRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from("timer:foo?period=1000")
                .log("Hello World");
    }
}
```

3.1.1. Endpoint DSL

自 Camel 3.0 起，您还可以使用流畅的构建器来定义 Camel 端点。以下示例等同于上一个示例：

```
import org.apache.camel.builder.RouteBuilder;
import static org.apache.camel.builder.endpoint.StaticEndpointBuilders.timer;

public class TimerRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from(timer("foo").period(1000))
                .log("Hello World");
    }
}
```

注意

所有 Camel 组件的构建器方法都通过 **camel-quarkus-core** 提供，但您仍需要添加给定组件的扩展作为路由正常工作的依赖项。如果是上例，它会是 **camel-quarkus-timer**。

第 4 章 配置

Camel Quarkus 会自动配置和部署 Camel Context bean，它默认根据 Quarkus 应用程序生命周期启动/停止。配置步骤在 Quarkus 的增强阶段进行，由 Camel Quarkus 扩展驱动，该扩展可以使用 Camel Quarkus 特定的 **quarkus.camel prerequisites** 属性进行调优。

注意

quarkus.camel prerequisites 配置属性记录在单独的扩展页面中 - 例如，请参阅 Camel Quarkus Core。

配置完成后，会在 RUNTIME_INIT 阶段编译并启动最小的 Camel Runtime。

4.1. 配置 CAMEL 组件

4.1.1. application.properties

要通过属性配置 Apache Camel 组件和其他方面，请确保您的应用程序直接依赖于 **camel-quarkus-core** 或 传输。因为大多数 Camel Quarkus 扩展都依赖于 **camel-quarkus-core**，因此通常不需要显式添加它。

camel-quarkus-core 将功能从 Camel Main 引入到 Camel Quarkus。

在以下示例中，您可以通过 **application.properties** 在 **LogComponent** 中设置特定的 **ExchangeFormatter** 配置：

```
camel.component.log.exchange-formatter =
#class:org.apache.camel.support.processor.DefaultExchangeFormatter
camel.component.log.exchange-formatter.show-exchange-pattern = false
camel.component.log.exchange-formatter.show-body-type = false
```

4.1.2. CDI

您还可以使用 CDI 以编程方式配置组件。

推荐的方法是观察 **ComponentAddEvent**，并在路由和 **CamelContext** 启动前配置组件：

```
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.event.Observes;
import org.apache.camel.quarkus.core.events.ComponentAddEvent;
import org.apache.camel.component.log.LogComponent;
import org.apache.camel.support.processor.DefaultExchangeFormatter;

@ApplicationScoped
public static class EventHandler {
    public void onComponentAdd(@Observes ComponentAddEvent event) {
        if (event.getComponent() instanceof LogComponent) {
            /* Perform some custom configuration of the component */
            LogComponent logComponent = ((LogComponent) event.getComponent());
            DefaultExchangeFormatter formatter = new DefaultExchangeFormatter();
            formatter.setShowExchangePattern(false);
            formatter.setShowBodyType(false);
            logComponent.setExchangeFormatter(formatter);
```

```
    }
  }
}
```

4.1.2.1. 生成 @Named 组件实例

或者，您也可以使用 **@Named** producer 方法创建和配置组件。这作为 Camel 使用组件 URI 方案从其 registry 中查找组件。例如，如果是 **LogComponent** Camel 会查找名为 bean **的日志**。

警告

虽然生成 **@Named** 组件 bean 通常可以正常工作，但可能会导致一些组件出现问题。

Camel Quarkus 扩展可以执行以下操作之一或多个：

- 传递默认 Camel 组件类型的自定义子类型。请参阅 Vert.x WebSocket 扩展示例。
- 对组件执行一些特定于 Quarkus 的自定义。请参阅 JPA 扩展 示例。

当您生成自己的组件实例时，不会执行这些操作，因此推荐在观察器方法中配置组件。

```
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Named;

import org.apache.camel.component.log.LogComponent;
import org.apache.camel.support.processor.DefaultExchangeFormatter;

@ApplicationScoped
public class Configurations {
    /**
     * Produces a {@link LogComponent} instance with a custom exchange formatter set-up.
     */
    @Named("log") (1)
    LogComponent log() {
        DefaultExchangeFormatter formatter = new DefaultExchangeFormatter();
        formatter.setShowExchangePattern(false);
        formatter.setShowBodyType(false);

        LogComponent component = new LogComponent();
        component.setExchangeFormatter(formatter);

        return component;
    }
}
```

(1) 如果方法的名称相同，则可以省略 **@Named** 注释的 **"log"** 参数。

4.2. 根据惯例的配置

除了支持通过属性配置 Camel 外，**camel-quarkus-core** 还允许您使用约定来配置 Camel 行为。例如，如果 CDI 容器中有一个 **ExchangeFormatter** 实例，则它将自动将 bean 链接到 **LogComponent**。

其他资源

- 在 OpenShift Container Platform 中配置和使用 Metering

第 5 章 CAMEL QUARKUS 中的上下文和依赖注入(CDI)

CDI 在 Quarkus 中扮演了一个中央角色，而 Camel Quarkus 也为它提供了第一类支持。

您可以使用 **@Inject**、**@ConfigProperty** 和类似注解（例如，将 bean 和配置值注入 Camel **RouteBuilder**），例如：

```
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Inject;
import org.apache.camel.builder.RouteBuilder;
import org.eclipse.microprofile.config.inject.ConfigProperty;

@ApplicationScoped 1
public class TimerRoute extends RouteBuilder {

    @ConfigProperty(name = "timer.period", defaultValue = "1000") 2
    String period;

    @Inject
    Counter counter;

    @Override
    public void configure() throws Exception {
        fromF("timer:foo?period=%s", period)
                .setBody(exchange -> "Incremented the counter: " + counter.increment())
                .to("log:cdi-example?showExchangePattern=false&showBodyType=false");
    }
}
```

1 **@ApplicationScoped** 注释需要 **@Inject** 和 **@ConfigProperty**，才能在 **RouteBuilder** 中工作。请注意，**@ApplicationScoped** Bean 由 CDI 容器管理，其生命周期比其中一个纯文本 **RouteBuilder** 更复杂。换句话说，在 **RouteBuilder** 中使用 **@ApplicationScoped** 会有一些引导时间损失，因此，您应该仅在需要时为 **RouteBuilder** 标注。

2 **timer.period** 属性的值在 example 项目的 **src/main/resources/application.properties** 中定义。

提示

如需了解更多详细信息，请参阅 Quarkus Dependency Injection 指南。

5.1. 访问 CAMELCONTEXT

要访问 **CamelContext**，只需将其注入您的 bean:

```
import jakarta.inject.Inject;
import jakarta.enterprise.context.ApplicationScoped;
import java.util.stream.Collectors;
import java.util.List;
import org.apache.camel.CamelContext;

@ApplicationScoped
public class MyBean {
```

```
    @Inject
    CamelContext context;

    public List<String> listRouteIds() {
        return context.getRoutes().stream().map(Route::getId).sorted().collect(Collectors.toList());
    }
}
```

5.2. @ENDPOINTINJECT AND @PRODUCE

如果您使用 **@org.apache.camel.EndpointInject** 和 **@org.apache.camel.Produce** from 普通 Camel 或 SpringBoot 上的 Camel，您也可以继续在 Quarkus 上使用它们。

org.apache.camel.quarkus:camel-quarkus-core 支持以下用例：

```
import jakarta.enterprise.context.ApplicationScoped;
import org.apache.camel.EndpointInject;
import org.apache.camel.FluentProducerTemplate;
import org.apache.camel.Produce;
import org.apache.camel.ProducerTemplate;

@ApplicationScoped
class MyBean {

    @EndpointInject("direct:myDirect1")
    ProducerTemplate producerTemplate;

    @EndpointInject("direct:myDirect2")
    FluentProducerTemplate fluentProducerTemplate;

    @EndpointInject("direct:myDirect3")
    DirectEndpoint directEndpoint;

    @Produce("direct:myDirect4")
    ProducerTemplate produceProducer;

    @Produce("direct:myDirect5")
    FluentProducerTemplate produceProducerFluent;

}
```

您可以使用任何其他 Camel producer 端点 URI 而不是 **direct:myDirect***。

警告

setter 方法不支持 **@EndpointInject** 和 **@Produce** - 请参阅 #2579

org.apache.camel.quarkus:camel-quarkus-bean 支持以下用例：

```
import jakarta.enterprise.context.ApplicationScoped;
import org.apache.camel.Produce;

@ApplicationScoped
class MyProduceBean {

    public interface ProduceInterface {
        String sayHello(String name);
    }

    @Produce("direct:myDirect6")
    ProduceInterface produceInterface;

    void doSomething() {
        produceInterface.sayHello("Kermit")
    }

}
```

5.3. CDI 和 CAMEL BEAN 组件

5.3.1. 按名称引用 bean

要按名称引用路由定义中的 bean，只需为 bean标上 **@Named ("myNamedBean")** 和 **@ApplicationScoped** （或其他支持范围）。https://quarkus.io/guides/cdi-reference#supported_features**@RegisterForReflection** 注释对于原生模式非常重要。

```
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Named;
import io.quarkus.runtime.annotations.RegisterForReflection;

@ApplicationScoped
@Named("myNamedBean")
@RegisterForReflection
public class NamedBean {
    public String hello(String name) {
        return "Hello " + name + " from the NamedBean";
    }
}
```

然后，您可以在路由定义中使用 **myNamedBean** 名称：

```
import org.apache.camel.builder.RouteBuilder;
public class CamelRoute extends RouteBuilder {
    @Override
    public void configure() {
        from("direct:named")
                .bean("myNamedBean", "hello");
        /* ... which is an equivalent of the following: */
        from("direct:named")
                .to("bean:myNamedBean?method=hello");
    }
}
```

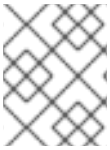
作为 **@Named** 的替代选择，您也可以使用 **io.smallrye.common.annotation.Identifier** 来命名并识别 bean。

```
import jakarta.enterprise.context.ApplicationScoped;
import io.quarkus.runtime.annotations.RegisterForReflection;
import io.smallrye.common.annotation.Identifier;

@ApplicationScoped
@Identifier("myBeanIdentifier")
@RegisterForReflection
public class MyBean {
    public String hello(String name) {
        return "Hello " + name + " from MyBean";
    }
}
```

然后引用 Camel 路由中的标识符值：

```
import org.apache.camel.builder.RouteBuilder;
public class CamelRoute extends RouteBuilder {
    @Override
    public void configure() {
        from("direct:start")
                .bean("myBeanIdentifier", "Camel");
    }
}
```



注意

我们的目标是支持 Camel 文档的 Bean 绑定 部分中列出的所有用例。如果某些 bean 绑定场景无法为您工作，则不要给您 提交问题。

5.3.2. @Consume

自 Camel Quarkus 2.0.0 起，**camel-quarkus-bean** 工件支持 **@org.apache.camel.Consume** - 请参阅 Camel 文档的 Pojo consuming 部分。

声明类，如下所示

```
import org.apache.camel.Consume;
public class Foo {

  @Consume("activemq:cheese")
  public void onCheese(String name) {
    ...
  }
}
```

将自动创建以下 Camel 路由

```
from("activemq:cheese").bean("foo1234", "onCheese")
```

适用于您的.请注意，Camel Quarkus 将隐式地将 **@jakarta.inject.Singleton** 和 **jakarta.inject.Named**

("foo1234") 添加到 bean 类，其中 **1234** 是从完全限定类名称获取的哈希值代码。如果您的 bean 设置了一些 CDI 范围（如 **@ApplicationScoped**）或 **@Named ("someName")**，则会在自动创建的路由中遵守这些范围。

第 6 章 OBSERVABILITY（可观察性）

6.1. 健康和存活度检查

通过 MicroProfile Health 扩展支持健康检查和存活度检查。它们可以通过 Camel Health API 或 Quarkus MicroProfile Health 进行配置。

所有配置的检查都在标准 MicroProfile Health 端点 URL 上提供：

- http://localhost:8080/q/health
- http://localhost:8080/q/health/live
- http://localhost:8080/q/health/ready

6.2. 指标

我们提供 MicroProfile 指标 以公开指标。

为您提供了一些基本的 Camel 指标，可以通过在路由中配置其他指标来补充这些指标。

指标在标准 Quarkus 指标端点上提供：

- http://localhost:8080/q/metrics

第 7 章 原生模式

有关以原生模式编译和测试应用程序的更多信息，请参阅 *Compiling your Quarkus applications to native executables* 指南中的 Producing a native executable。

7.1. 字符编码

默认情况下，并非所有 **Charsets** 都以原生模式提供。

```
Charset.defaultCharset(), US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16
```

如果您希望您的应用程序需要没有包括在这个集合中的编码，或者在原生模式下看到 **UnsupportedCharsetException** thrown，请将以下条目添加到 **application.properties** 中：

```
quarkus.native.add-all-charsets = true
```

另请参阅 Quarkus 文档中的 quarkus.native.add-all-charsets。

7.2. LOCALE

默认情况下，原生镜像中仅包含构建 JVM 默认区域设置。Quarkus 提供了一种通过 **application.properties** 设置区域的方法，因此您不需要依赖 **LANG** 和 **LC** ptr environement 变量：

```
quarkus.native.user-country=US
quarkus.native.user-language=en
```

还支持将多个区域嵌入到原生镜像中，并通过 Mandrel 命令行选项选择默认区域设置 **-H:IncludeLocales=fr,en,H:+IncludeAllLocales** 和 **-H:DefaultLocale=de**。您可以通过 Quarkus **quarkus.native.additional-build-args** 属性设置它们。

7.3. 将资源嵌入到原生可执行文件中

通过 **Class.getResource （）, Class.getResourceAsStream （）, ClassLoader.getResource （）,, ClassLoader.getResourceAsStream （）** 等访问的资源需要在运行时明确列出，以包括在原生可执行文件中。

这可以通过在 **application.properties** 文件中使用 Quarkus **quarkus.native.resources.includes** 和 **quarkus.native.resources.excludes** 属性来完成，如下所示：

```
quarkus.native.resources.includes = docs/*,images/*
quarkus.native.resources.excludes = docs/ignored.adoc,images/ignored.png
```

在上例中，名为 **docs/included.adoc** 和 **images/included.png** 的资源将被嵌入到原生可执行文件中，而 **docs/ignored.adoc** 和 **images/ignored.png** 不会被嵌入到。

resources.includes 和 **resources.excludes** 都是用逗号分开的 Ant 路径 glob 模式的列表。

如需了解更多详细信息，请参阅 红帽构建的 Apache Camel for Quarkus 扩展 参考。

7.4. 在原生模式中使用 ONEXCEPTION 子句

在原生模式中使用 Camel **onException** 处理 时，您负责注册异常类来反映。

例如，具有带有 **onException** 处理的 camel 上下文：

```
onException(MyException.class).handled(true);
from("direct:route-that-could-produce-my-exception").throw(MyException.class);
```

应当注册 **mypackage.MyException** 类来反映。如需更多信息，请参阅 注册类以了解反映。

7.5. 注册课程来反映

默认情况下，动态反映在原生模式中不可用。需要反映访问的类，必须在编译时注册以反应。

在很多情况下，应用程序开发人员不需要小心，因为 Quarkus 扩展可以检测需要反映的类并自动注册它们。

然而，在某些情况下，Quarkus 扩展可能会错过一些类，它取决于应用程序开发人员进行注册。有两种方法可以做到这一点：

1. **@io.quarkus.runtime.annotations.RegisterForReflection** 注释可用于注册所使用的类，或者通过 **targets** 属性注册第三方类。

```
import io.quarkus.runtime.annotations.RegisterForReflection;

@RegisterForReflection
class MyClassAccessedReflectively {
}

@RegisterForReflection(
    targets = {
        org.third-party.Class1.class,
        org.third-party.Class2.class
    }
)
class ReflectionRegistrations {
}
```

2. **application.properties** 中的 **quarkus.camel.native.reflection** 选项：

```
quarkus.camel.native.reflection.include-patterns = org.apache.commons.lang3.tuple.*
quarkus.camel.native.reflection.exclude-patterns = org.apache.commons.lang3.tuple.*Triple
```

要使这些选项正常工作，包含所选类的工件必须包含 Jandex 索引('META-INF/jandex.idx')，或者必须使用 'application.properties' 中的 'quarkus.index-dependencyAPPEND' 选项进行索引注册，例如：

```
quarkus.index-dependency.commons-lang3.group-id = org.apache.commons
quarkus.index-dependency.commons-lang3.artifact-id = commons-lang3
```

7.6. 注册类进行序列化

如果通过 **quarkus.camel.native.reflection.serialization-enabled** 来请求序列化支持，则 CamelSerializationProcessor.BASE_SERIALIZATION_CLASSES 中列出的类会自动注册以便序列化。

您可以使用 **@RegisterForReflection (serialization = true)** 注册更多类。

第 8 章 KUBERNETES

本指南描述了在 kubernetes 上配置和部署 Camel Quarkus 应用程序的不同方法。它还描述了 Knative 和 Service Binding 的一些特定用例。

8.1. KUBERNETES

Quarkus 支持为 vanilla Kubernetes、OpenShift 和 Knative 生成资源。另外，Quarkus 通过将生成的清单应用到目标集群的 API 服务器，将应用程序部署到目标 Kubernetes 集群。如需更多信息，请参阅 **Quarkus Kubernetes 指南**。

8.2. KNATIVE

消费者支持 Knative 部署的 Camel Quarkus 扩展是：

- **camel-quarkus-grpc**
- **camel-quarkus-knative**
- **camel-quarkus-platform-http**
- **camel-quarkus-rest**
- **camel-quarkus-servlet**
- **带有 webhook 的 camel-quarkus-telegram**
- **camel-quarkus-vertx-websocket**

8.3. 服务绑定

Quarkus 还支持 Kubernetes 的 Service Binding 规格 将服务绑定到应用程序。

以下 Camel Quarkus 扩展可用于 Service Binding：

- **camel-quarkus-kafka**