Red Hat build of Apache Camel Extensions for Quarkus 2.13

使用 Camel Extensions for Quarkus 开发应用程序

使用 Camel Extensions for Quarkus 开发应用程序

Last Updated: 2023-09-19

Red Hat build of Apache Camel Extensions for Quarkus 2.13 使用 Camel Extensions for Quarkus 开发应用程序

使用 Camel Extensions for Quarkus 开发应用程序

法律通告

摘要

本指南面向在 Camel Extensions for Quarkus 上编写 Camel 应用程序的开发人员。

目录

前言 **3**
使开源包含更多 3

第 1 章 使用 CAMEL EXTENSIONS FOR QUARKUS 开发应用程序简介 **4**

第 2 章 依赖项管理 **5**
2.1. 用于启动新项目的QUARKUS 工具 5

第 3 章 定义 CAMEL 路由 **7**
3.1. JAVA DSL 7

第 4 章 配置 **8**
4.1. 配置 CAMEL 组件 8
4.2. 按惯例配置 10

第 5 章 CAMEL QUARKUS 中的上下文和依赖注入(CDI) **11**
5.1. 访问 CAMELCONTEXT 11
5.2. CDI 和 CAMEL BEAN 组件 12

第 6 章 OBSERVABILITY（可观察性） **13**
6.1. 健康和存活度检查 13
6.2. 指标 13

第 7 章 原生模式 **14**
7.1. 字符编码 14
7.2. LOCALE 14
7.3. 在原生可执行文件中嵌入资源 14
7.4. 在原生模式中使用 ONEXCEPTION 子句 14
7.5. 注册类以进行反映 15
7.6. 为序列化注册类 15

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 使用 CAMEL EXTENSIONS FOR QUARKUS 开发应用程序简介

本指南面向在 Camel Extensions for Quarkus 上编写 Camel 应用程序的开发人员。

Camel 组件在 Camel Extensions for Quarkus 中支持的 Camel 组件具有关联的 Camel Extensions for Quarkus 扩展。有关此发行版本中支持的 Camel Extensions for Quarkus 扩展的更多信息，请参阅 Camel Extensions for Quarkus 参考指南。

第 2 章 依赖项管理

2.1. 用于启动新项目的QUARKUS 工具

特定的 Camel Extensions for Quarkus 发行版本应该只用于特定的 Quarkus 版本。

在新项目中获取依赖关系版本的最简单、最简单的方法是使用 Quarkus 工具之一：

- code.quarkus.redhat.com - 在线项目生成器，
- Quarkus Maven 插件

这些工具允许您选择扩展和构建新的 Maven 项目。

生成的 **pom.xml** 将类似如下：

```
<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 2.13.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>
```

提示

可用的扩展统一跨越 Quarkus Core、Camel Quarkus 和其他第三方参与项目，如 Hazelcast、Cassandsand、Jasraer 和 OptaPlanner。

BOM 代表"Bill of Materials" - 这是一个 **pom.xml**，其主要用途是管理工件的版本，因此最终用户在其项目中导入 BOM 不需要负责哪些工件的特定版本。换句话说，在 **pom.xml** 的 **<depependencyManagement** > 部分中导入了 BOM，您可以避免为给定 BOM 管理的依赖项指定版本。

pom.xml 中包含的特定 BOM 取决于您使用配置为使用一组一致性 BOM 的生成器工具选择的扩展。

如果您选择在稍后由 **pom.xml** 文件中的任何 BOM 管理扩展，则不需要手动搜索适当的 BOM。使用 **quarkus-maven-plugin**，您可以选择扩展，该工具会根据需要添加适当的 BOM。您还可以使用 **quarkus-maven-plugin** 来升级 BOM 版本。

com.redhat.quarkus.platform BOM 相互一致，这意味着如果工件在多个 BOM 中管理，则始终与同一版本一起管理。这样做具有应用程序开发人员不需要处理可能来自各种独立项目的单个工件的兼容性。

第 3 章 定义 CAMEL 路由

Camel Extensions 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**，并使用流畅的构建器方法来定义 Camel 路由。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. 端点 DSL

从 Camel 3.0 开始，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。如果出现以上示例，它将是 **camel-quarkus-timer**。

第 4 章 配置

Camel Quarkus 会自动配置和部署 Camel Context bean，默认情况下根据 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，它由 Camel Quarkus 扩展驱动，该扩展可使用 Camel Quarkus 特定的 **quarkus.camel114** 属性进行调整。

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。

完成配置后，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 的组件和其他方面，请确保您的应用程序直接或传输地依赖于 **camel-quarkus-core**。因为大多数 Camel Quarkus 扩展依赖于 **camel-quarkus-core**，因此您通常不需要显式添加它。

camel-quarkus-core 从 Camel Main 引入到 Camel Quarkus 的功能。

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 javax.enterprise.context.ApplicationScoped;
import javax.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。这可以正常工作，因为 Camel 使用组件 URI 方案从其 registry 中查找组件。例如，如果 **LogComponent** Camel 查找名为 bean **的日志**。

警告

请注意，虽然生成 **@Named** 组件 Bean 通常可以正常工作，但可能会导致某些组件出现微小的问题。

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，因此建议在观察器方法中配置组件是推荐的方法。

```
import javax.enterprise.context.ApplicationScoped;
import javax.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，可以省略 **@Named** 注释的 **"log"** 参数。

4.2. 按惯例配置

除了通过属性配置 Camel 外，还可以使用 **camel-quarkus-core** 来配置 Camel 行为。例如，如果 CDI 容器中只有一个 **ExchangeFormatter** 实例，那么它将自动将 bean 发送到 **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 和 Camel Quarkus 中扮演一个中央角色，同时还为它提供了一个第一个支持。

您可以使用 **@Inject**、**@ConfigProperty** 和类似的注解将 Bean 和配置值注入 Camel **RouteBuilder**，例如：

```
import javax.enterprise.context.ApplicationScoped;
import javax.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** 在 **RouteBuilder** 中工作。请注意，**@ApplicationScoped** Bean 由 CDI 容器管理，其生命周期比普通 **RouteBuilder** 之一更复杂。换句话说，在 **RouteBuilder** 中使用 **@ApplicationScoped** 附带一些引导时间损失，因此您应该仅在真正需要时使用 **@ApplicationScoped** 注解 **RouteBuilder**。

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

提示

如需更多详细信息，请参阅 Quarkus Dependency Injection 指南。

5.1. 访问 CAMELCONTEXT

访问 **CamelContext**，将其注入您的 bean 中：

```
import javax.inject.Inject;
import javax.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. CDI 和 CAMEL BEAN 组件

5.2.1. 按名称引用 bean

要按名称引用路由定义中的 bean，只需使用 **@Named ("myNamedBean")** 和 **@ApplicationScoped** 标注 bean。**@RegisterForReflection** 注释对于原生模式非常重要。

```
import javax.enterprise.context.ApplicationScoped;
import javax.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")
                .to("bean:namedBean?method=hello");
    }
}
```

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 配置。

所有配置的检查都可用于标准 MicroProfile Health 端点 URL：

- 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。

一些基本的 Camel 指标供您开箱即用，可以通过在路由中配置其他指标来补充这些指标。

指标位于标准 Quarkus 指标端点上：

- 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
```

如果您期望您的应用程序需要包括在此集合中的任何编码，或者您以原生模式看到 **UnsupportedCharsetException** thrown，请在 **application.properties** 中添加以下条目：

```
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** 设置区域设置区域设置的方法，因此您不需要依赖 **LANG** 和 **LC** the 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 **-H:IncludeLocales=fr,H:+IncludeAllLocales** 和 **-H:DefaultLocale=de** 选择默认区域设置。您可以通过 Quarkus **quarkus.native.additional-build-args** 属性来设置它们。

7.3. 在原生可执行文件中嵌入资源

通过 **Class.getResource （）**、class **Class.getResourceAsStream （）**、class **Loader.getResource （）**、**classLoader.getResourceAsStream （）** 等等访问的资源需要明确列出，以便在运行时包含原生可执行文件。

这可以通过 Quarkus **quarkus.native.resources.includes** 和 **quarkus.native.resources.excludes** 属性在 **application.properties** 文件中完成，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** 不会被嵌入到原生可执行文件中。

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

如需更多详细信息，请参阅 Camel Extensions for Quarkus 参考 参考。

7.4. 在原生模式中使用 ONEXCEPTION 子句

当在原生模式下使用 camel onException 处理时，应用程序开发人员负责注册异常类以进行反映。

例如，有一个带有 onException 处理的 camel 上下文，如下所示：

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

应注册类 **mypackage.MyException** 进行反映，请参阅 Registering class for reflection。

7.5. 注册类以进行反映

默认情况下，动态反映在原生模式中不可用。需要具有反映访问权限的类，必须在编译时注册以进行反映。

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

然而，在某些情况下，Quarkus 扩展可能会丢失一些类，它是应用程序开发人员注册它们。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。
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')，或使用 'application.properties' - 例如在 'application.properties' -e.g 中注册它们以进行索引。

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

7.6. 为序列化注册类

如果通过 **quarkus.camel.native.reflection.serialization-enabled**，则 CamelSerializationProcessor.BASE_SERIALIZATION_CLASSES 中列出的类会自动注册以进行序列化。

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