Red Hat build of Apache Camel 4.4

工具指南

红帽提供的工具指南

Last Updated: 2024-03-27

Red Hat build of Apache Camel 4.4 工具指南

红帽提供的工具指南

法律通告

摘要

工具指南本指南引入了用于 Red Hat build for Apache Camel 的工具扩展。

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。详情请查看 CTO Chris Wright 的信息。

第 1 章 关于工具指南

本指南介绍了红帽构建的 Apache Camel 的 VS Code 扩展以及如何安装和使用 Camel CLI。

重要

Apache Camel 的 VS Code 扩展列为开发支持。有关 开发支持范围的更多信息，请参阅红帽构建的 Apache Camel 开发支持范围。

用于红帽构建的 Apache Camel 的 VS Code 扩展。

本指南中介绍了以下 VS Code 扩展。

- Apache Camel 扩展的语言支持支持 Apache Camel 扩展，增加了对 Apache Camel for Java、Yaml 或 XML DSL 代码的语言支持。
- Apache Camel 的调试适配器 Apache Camel 的 Debug Adapter 增加了 Camel Debugger 电源，方法是附加到使用 Java、Yaml 或 XML DSL 编写的正在运行的 Camel 路由。

Camel CLI

这是一个基于 JBang 的 Camel 应用程序，可用于运行 Camel 路由。

第 2 章 红帽为 APACHE CAMEL 安装扩展包

重要

Apache Camel 的 VS Code 扩展列为开发支持。有关 开发支持范围的更多信息，请参阅红帽构建的 Apache Camel 开发支持范围。

本节解释了如何为 Apache Camel 安装扩展包。

流程

1. 打开 VS Code 编辑器。
2. 在 **VS Code** 编辑器中，选择 **View > Extensions**。
3. 在搜索栏中，键入 **Camel**。从搜索结果中选择 **Extension Pack for Apache Camel by Red Hat** 选项，然后点 Install。

这会在 VS Code 编辑器中安装包括 Apache Camel 扩展的扩展软件包。

第 3 章 使用 APACHE CAMEL 扩展的语言支持

重要

Apache Camel 的 VS Code 扩展列为开发支持。有关 开发支持范围的更多信息，请参阅红帽构建的 Apache Camel 开发支持范围。

Visual Studio Code 语言支持扩展添加了对 Apache Camel for XML DSL 和 Java DSL 代码的语言支持。

3.1. 关于 APACHE CAMEL 扩展的语言支持

此扩展直接在 Visual Studio Code 编辑器中为 Apache Camel URI 元素提供完成、验证和文档功能。它充当使用 Microsoft 语言服务器协议的客户端，该协议与 Camel 语言服务器通信以提供所有功能。

3.2. APACHE CAMEL 扩展的语言支持特性

下面列出了语言支持扩展的重要特性：

- Apache Camel URI 的语言服务支持。
- 当您将光标悬停在 Camel 组件上时，快速参考文档。
- Camel URI 的诊断。
- Java 和 XML langauge 的导航.
- 使用 Camel CLI 创建 Yaml DSL 指定的 Camel 路由.
- 创建 Camel Quarkus 项目
- 在 SpringBoot 项目中创建一个 Camel
- 特定的 Camel Catalog 版本
- Camel Catalog 的特定运行时供应商

3.3. 要求

使用 Apache Camel 语言服务器时必须考虑以下点：

- 目前，启动 Apache Camel 语言服务器需要 Java 17。**java.home** VS Code 选项使用不同于在机器上安装的默认 JDK 版本的不同版本。
- 对于某些功能，必须在系统命令行中提供 JBang。
- 对于 XML DSL 文件：
  - 使用 **.xml** 文件扩展名。
  - 指定 Camel 命名空间 以供参考，请参阅 http://camel.apache.org/schema/blueprint 或 http://camel.apache.org/schema/spring。
- 对于 Java DSL 文件：

- 使用 **.java** 文件扩展名。
- 指定 Camel 软件包（通常来自导入的软件包），例如 **import org.apache.camel.builder.RouteBuilder**。
- 要引用 Camel 组件，请使用 from 或 to，以及没有空格的字符串。字符串不能是一个变量。例如，**from ("timer:timerName")** 可以正常工作，但 **from ("timer:timerName")** 和 **from (aVariable)** 无法正常工作。

3.4. 安装 APACHE CAMEL 扩展的语言支持

您可以从 VS Code 扩展 Marketplace 和 Open VSX Registry 下载 Apache Camel 扩展的语言支持。您还可以在 Microsoft VS Code 中直接安装 Apache Camel 扩展的语言支持。

流程

1. 打开 VS Code 编辑器。
2. 在 **VS Code** 编辑器中，选择 **View > Extensions**。
3. 在搜索栏中，键入 **Camel**。从搜索结果中选择 **Language Support for Apache Camel**选项，然后点 Install。

这会在编辑器中安装语言支持扩展。

3.5. 使用特定的 CAMEL 目录版本

您可以使用特定的 Camel 目录版本。点 **File** > **Preferences** > **Settings** > **Apache Camel Tooling** > **Camel catalog version**。对于版本标识符中包含 redhat 的红帽产品化版本，会自动添加 Maven Red Hat 存储库。

注意

首次使用版本时，需要几秒钟/分钟才可用，具体取决于在后台下载依赖项的时间。

限制

- 使用的 Kamelet 目录只是社区支持的版本。有关支持的 Kamelets 列表，请参阅链接： 支持的 Kamelets
- Modeline 配置仅基于社区。并非所有特征和模式行参数都被支持。

其他资源

- 红帽对 Apache Camel 的语言支持

第 4 章 使用 VS CODE DEBUG ADAPTER FOR APACHE CAMEL 扩展

重要

Apache Camel 的 VS Code 扩展列为开发支持。有关 开发支持范围的更多信息，请参阅红帽构建的 Apache Camel 开发支持范围。

这是 Visual Studio Code 扩展，它通过附加到使用 Java、Yaml 或 XML DSL 编写的运行的 Camel 路由来添加 Camel Debugger 电源。

4.1. DEBUG ADAPTER 的功能

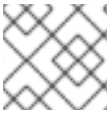
用于 Apache Camel 扩展的 VS Code Debug Adapter 支持以下功能：

- 仅针对 XML 的 Camel Main 模式.
- Camel 调试器通过使用 JMX url 将其附加到使用 Java、Yaml 或 XML 运行的 Camel 路由。
- Camel 调试器本地使用 PID 将其附加到使用 Java、Yaml 或 XML 运行的 Camel 路由。
- 您可以将它用于单个 Camel 上下文。
- 添加或删除断点。
- 具有简单语言的条件断点。
- 在暂停的断点上检查变量值。
- 恢复单个路由实例并恢复所有路由实例。
- 当路由定义位于同一文件中时的步骤。
- 允许更新 scope Debugger 中的变量，在消息正文中，在类型为 String 的消息标头中更新变量，以及类型为 String 的 exchange 属性
- 支持命令 **Run Camel Application with JBang 和 Debug**。
  - 此命令允许在简单情况下单击 start 和 Camel debug。这个命令通过以下方式提供：
    - 命令面板.它要求在当前编辑器中打开有效的 Camel 文件。
    - File explorer 中的上下文菜单.所有 If. **xml**、..**java**、..**yaml** 和 If **yml** 可见。
    - Camel 文件顶部的 Codelens （代码形象检查是否存在来自的，以及对 **java**、**xml** 和 **yaml** 文件的日志）。
- 支持命令 **Run Camel application with JBang**。
  - 它需要一个在编辑器中打开的 Yaml DSL (.yaml|.yml)中定义的有效 Camel 文件。
- Camel 调试器启动配置片断
- 配置片断用于启动 Camel 应用程序已准备好使用 JBang 接受 Camel 调试器连接，或 Maven with Camel maven 插件

4.2. 要求

在对 Apache Camel 扩展使用 VS Code Debug Adapter 时，必须考虑以下点：

- 必须安装带有 **com.sun.tools.attach.VirtualMachine** （大多数 JVM ）的 Java Runtime Environment 17 或更高版本，如 Hotspot 和 OpenJDK。
- 要调试的 Camel 实例必须遵循以下要求：
  - Camel 3.16 或更高版本
  - 在 classpath 上具有 camel-debug。
  - 启用 JMX。



注意

对于某些功能，必须在系统命令行中提供 JBang。

4.3. 为 APACHE CAMEL 安装 VS CODE DEBUG ADAPTER

您可以从 VS Code 扩展 Marketplace 和 Open VSX Registry 下载用于 Apache Camel 扩展的 VS Code Debug Adapter。您还可以在 Microsoft VS Code 中直接安装 Apache Camel 扩展的 Debug Adapter。

流程

1. 打开 VS Code 编辑器。
2. 在 **VS Code** 编辑器中，选择 **View > Extensions**。
3. 在搜索栏中，键入 **Camel Debug**。从搜索结果中选择 **Debug Adapter for Apache Camel**选项，然后点 Install。

这会在 VS Code 编辑器中安装 Apache Camel 的 Debug Adapter。

4.4. 使用 DEBUG ADAPTER

以下流程解释了如何使用 debug 适配器调试 camel 应用程序。

流程

1. 确保 **jbang** 二进制文件在系统命令行中可用。
2. 打开可使用 Camel CLI 启动的 Camel 路由。
3. 使用键 **Ctrl + Shift + P** 调用 **命令面板**，然后选择 **Run Camel Application with JBang and Debug** 命令，或者点击文件上出现的 codelens **Camel Debug with JBang**。
4. 等待路由启动并且调试器连接。
5. 在 Camel 路由上放置断点。
6. Debug。

其他资源

- 红帽 Apache Camel 的调试适配器

第 5 章 使用 CAMEL CLI

5.1. 安装 CAMEL CLI

先决条件

1. 必须在您的机器上安装 JBang。请参阅有关如何下载和安装 JBang 的说明。

安装 JBang 后，您可以从命令 shell 执行以下命令来验证 JBang 是否正常工作：

```
jbang version
```

这会输出已安装的 JBang 版本。

流程

1. 运行以下命令来安装 Camel CLI 应用程序：

```
jbang app install camel@apache/camel
```

这会在 JBang 内将 Apache Camel 安装为 **camel** 命令。这意味着，您可以通过只执行 **camel** 命令从命令行运行 Camel。

5.2. 使用 CAMEL CLI

Camel CLI 支持多个命令。**camel help** 命令可以显示所有可用的命令。

```
camel --help
```

注意

第一次运行此命令时，可能会导致依赖项被缓存，因此需要几秒钟来运行。如果您已使用 JBang，并且获得 **线程 "main" java.lang.NoClassDefFoundError: "org/apache/camel/dsl/jbang/core/commands/CamelJBangMain"** 等错误，请尝试再次清除 JBang 缓存和重新安装。

所有命令都支持 **--help**，如果提供了该标志，则会显示适当的帮助。

5.2.1. 启用 shell 完成

Camel CLI 为 bash 和 zsh 提供 shell 补全功能。要为 Camel CLI 启用 shell 完成，请运行：

```
source <(camel completion)
```

要使它永久生效，请运行：

```
echo 'source <(camel completion)' >> ~/.bashrc
```

5.3. 创建并运行 CAMEL 路由

您可以使用 **init** 命令创建新的基本路由。例如，要创建一个 XML 路由，请运行以下命令：

```
camel init cheese.xml
```

这会创建带有示例路由的文件 **cheese.xml** （当前目录中）。

要运行该文件，请运行：

```
camel run cheese.xml
```

注意

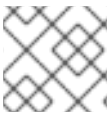
您可以在 Camel 中创建并运行任何受支持的 DSL，如 YAML、XML、Java、Groovy。

要创建新的 **.java** 路由，请运行：

```
camel init foo.java
```

当您使用 init 命令时，Camel 默认在当前目录中创建该文件。但是，您可以使用 **--directory** 选项在指定的目录中创建文件。例如，要在名为 **foobar** 的文件夹中创建，请运行：

```
camel init foo.java --directory=foobar
```



注意

当您使用 **--directory** 选项时，Camel 会自动清理此目录（如果已存在）。

5.3.1. 从多个文件运行路由

您可以从多个文件运行路由，例如运行两个 YAML 文件：

```
camel run one.yaml two.yaml
```

您可以从两个不同的文件（如 yaml 和 Java）运行路由：

```
camel run one.yaml hello.java
```

您可以使用通配符（例如，**X**）匹配多个文件，例如运行所有 yaml 文件：

```
camel run *.yaml
```

您可以运行以 foo* 开头的所有文件：

```
camel run foo*
```

要运行 目录中的所有文件，请使用：

```
camel run *
```

注意

run 目标也可以检测是 **属性** 的文件，如 **application.properties**。

5.3.2. 从输入参数运行路由

对于非常小的 Java 路由，可以将路由作为 CLI 参数提供，如下所示：

```
camel run --code='from("kamelet:beer-source").to("log:beer")'
```

这非常有限，因为 CLI 参数比文件使用点太繁琐。当您运行来自输入参数的路由时，请记住：

- 仅支持 Java DSL 代码。
- 代码以单引号括起，因此您可以在 Java DSL 中使用双引号。
- 代码仅限于可以从终端和 JBang 提供哪些字面值。
- 所有路由都必须在单个 **--code** 参数中定义。

注意

使用 **--code** 仅适用于非常快速和小型原型。

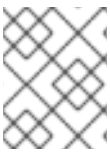
5.3.3. 带有实时重新加载的 dev 模式

当源文件被更新（保存），您可以使用 **--dev** 选项启用包含路由的实时重新加载的 dev 模式，如下所示：

```
camel run foo.yaml --dev
```

然后，在 Camel 集成运行时，您可以更新 YAML 路由并在保存时更新。这个选项可用于包括 **java** 的所有 DLS，例如：

```
camel run hello.java --dev
```



注意

live reload 选项仅用于开发目的，如果您遇到重新加载 JVM 类问题的问题，则可能需要重新启动集成。

5.3.4. 开发人员控制台

您可以启用开发人员控制台，为开发人员提供各种信息。要启用开发人员控制台，请运行：

```
camel run hello.java --console
```

然后可以从 Web 浏览器通过 http://localhost:8080/q/dev （默认）访问控制台。当 Camel 启动时，该链接也会在日志中显示。

控制台可让您了解正在运行的 Camel 集成，例如报告处理消息的最多路由。然后，您可以在这些路由中识别最慢的 EIP。

开发人员控制台也可以以 **JSON** 格式输出数据，供第三方工具用于捕获信息。例如，要通过 curl 输出顶级路由，请运行：

```
curl -s -H "Accept: application/json"  http://0.0.0.0:8080/q/dev/top/
```

如果您安装了 **jq**，则可以以 colour 格式并输出 JSON 数据，请运行：

```
curl -s -H "Accept: application/json"  http://0.0.0.0:8080/q/dev/top/ | jq
```

5.3.5. 使用配置集

Camel CLI 中的 **配置集** 是一个名称(id)，它引用使用 Camel CLI 自动载入的配置。默认配置集被命名为为应用程序，它是一个(smart default)，以便 Camel CLI 自动加载 **application.properties** （如果存在）。这意味着您可以创建与具有相同名称的特定属性文件匹配的配置集。

例如，使用名为 **local** 的配置集运行 意味着 Camel CLI 将加载 **local.properties** 而不是 **application.properties**。要使用配置集，请指定命令行选项 **--profile**，如下所示：

```
camel run hello.java --profile=local
```

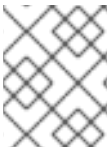
一次只能指定一个配置集名称，如 **--profile=local,two** 无效。

在 **属性文件中**，您可以配置 Camel Main 中的所有配置。要关闭并启用日志屏蔽，请运行以下命令：

```
camel.main.streamCaching=false
camel.main.logMask=true
```

您还可以配置 Camel 组件，如 **camel-kafka** 来声明代理的 URL：

```
camel.component.kafka.brokers=broker1:9092,broker2:9092,broker3:9092
```



注意

以 **camel.jbang** 开头的键是 Camel CLI 内部使用的保留密钥，并允许为 Camel CLI 命令预配置参数。

5.3.6. 通过互联网下载 JAR

默认情况下，Camel CLI 会自动解析运行 Camel 所需的依赖项，这由 JBang 和 Camel 分别完成。如果组件对当前在类路径上不可用的 JAR 的需求，则 Camel 本身会在运行时检测，然后可以自动下载 JAR。

Camel 按照以下顺序下载这些 JAR：

1. 来自 **~/.m2/repository**中的本地磁盘
2. 来自 Maven Central 的互联网
3. 从互联网的自定义第三方 Maven 存储库
4. 从 **~/.m2/settings.xml** 的活动配置文件中找到的所有软件仓库，或使用 **--maven-settings** 选项指定的设置文件。

如果您不希望 Camel CLI 通过互联网下载，您可以使用 **--download** 选项关闭它，如下所示：

```
camel run foo.java --download=false
```

5.3.7. 添加自定义 JAR

Camel CLI 会自动检测 Camel 组件、语言和数据格式的依赖项。这意味着不需要指定要使用的 JAR。但是，如果您需要添加第三方自定义 JAR，您可以在 Maven GAV 语法中使用 **--deps** 指定为 CLI 参数（**groupId:artifactId:version**），例如：

```
camel run foo.java --deps=com.foo:acme:1.0
```

```
To add a Camel dependency explicitly you can use a shorthand syntax (starting with `camel:` or `camel-`):
```

```
camel run foo.java --deps=camel-saxon
```

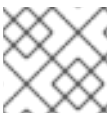
您可以指定用逗号分开的多个依赖项：

```
camel run foo.java --deps=camel-saxon,com.foo:acme:1.0
```

5.3.8. 使用第三方 Maven 存储库

Camel CLI 首先从本地存储库下载，然后从在线 Maven Central 存储库下载。要从第三方 Maven 存储库下载，您必须将此项指定为 CLI 参数，或者在 **application.properties** 文件中指定。

```
camel run foo.java --repos=https://packages.atlassian.com/maven-external
```



注意

您可以指定用逗号分开的多个软件仓库。

第三方 Maven 存储库的配置在 **application.properties** 文件中配置，其键为 **camel.jbang.repos**，如下所示：

```
camel.jbang.repos=https://packages.atlassian.com/maven-external
```

运行 Camel 路由时，会自动载入 **application.properties** ：

```
camel run foo.java
```

您还可以明确指定要使用的属性文件：

```
camel run foo.java application.properties
```

或者您可以将其指定为配置集：

```
camel run foo.java --profile=application
```

其中 profile id 是属性文件的名称。

5.3.9. 配置 Maven 使用情况

默认情况下，会加载现有的 **~/.m2/settings.xml** 文件，因此可以更改 Maven 解析过程的行为。Maven 设置文件提供有关 Maven 镜像、凭证配置（可能加密的）或活动配置集和其他存储库的信息。

Maven 存储库可以使用身份验证，Maven-way 来配置凭证是通过 **<server>** 元素：

```
<server>
   <id>external-repository</id>
   <username>camel</username>
   <password>{SSVqy/PexxQHvubrWhdguYuG7HnTvHlaNr6g3dJn7nk=}</password>
</server>
```

虽然可以使用纯文本指定密码，但我们建议您首先配置 maven master 密码，然后使用它配置存储库密码：

```
$ mvn -emp
Master password: camel
{hqXUuec2RowH8dA8vdqkF6jn4NU9ybOsDjuTmWvYj4U=}
```

以上密码必须添加到 **~/.m2/settings-security.xml** 文件中，如下所示：

```
<settingsSecurity>
  <master>{hqXUuec2RowH8dA8vdqkF6jn4NU9ybOsDjuTmWvYj4U=}</master>
</settingsSecurity>
```

然后，您可以配置常规密码：

```
$ mvn -ep
Password: camel
{SSVqy/PexxQHvubrWhdguYuG7HnTvHlaNr6g3dJn7nk=}
```

然后您可以在 **<server>/<password>** 配置中使用此密码。

默认情况下，Maven 从 **~/.m2/settings-security.xml** 文件中读取 master 密码，但您可以覆盖它。可以按照如下所示指定 **settings.xml** 文件本身的位置：

```
camel run foo.java --maven-settings=/path/to/settings.xml --maven-settings-
security=/path/to/settings-security.xml
```

如果要在不假定任何位置的情况下运行 Camel 应用程序（甚至 **~/.m2/settings.xml**），请使用这个选项：

```
camel run foo.java --maven-settings=false
```

5.3.10. 运行托管在 GitHub 上的路由

您可以使用 Camels 资源加载程序运行托管在 GitHub 上的路由。例如，要运行其中一个 Camel K 示例，请使用：

```
camel run github:apache:camel-kamelets-examples:jbang/hello-java/Hey.java
```

您还可以将 **https** URL 用于 GitHub。例如，您可以从 web-browser 中浏览示例，然后从浏览器窗口中复制 URL，并使用 Camel CLI 运行示例：

```
camel run https://github.com/apache/camel-kamelets-examples/tree/main/jbang/hello-java
```

您还可以使用通配符（例如 *）匹配多个文件，如运行所有 groovy 文件：

```
camel run https://github.com/apache/camel-kamelets-examples/tree/main/jbang/languages/*.groovy
```

或者，您可以运行以 rou* 开头的所有文件：

```
camel run https://github.com/apache/camel-kamelets-examples/tree/main/jbang/languages/rou*
```

5.3.10.1. 从 GitHub gists 运行路由

使用 GitHub 中的 gists 是一种快速共享您可轻松运行的小型 Camel 路由的方法。例如，要运行 gist，请使用：

```
camel run https://gist.github.com/davsclaus/477ddff5cdeb1ae03619aa544ce47e92
```

Gist 可以包含一个或多个文件，Camel CLI 将收集所有相关文件，因此 gist 可以包含多个路由、属性文件和 Java Bean。

5.3.11. 下载托管在 GitHub 上的路由

您可以使用 Camel CLI 将现有的示例从 GitHub 下载到本地磁盘，这允许修改示例并在本地运行。例如，您可以通过运行以下命令来下载 **依赖项注入** 示例：

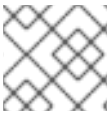
```
camel init https://github.com/apache/camel-kamelets-examples/tree/main/jbang/dependency-injection
```

然后，文件（不是子文件夹）下载到当前目录中。然后您可以使用以下内容在本地运行示例：

```
camel run *
```

您还可以使用 **--directory** 选项将文件下载到新文件夹中，例如要将文件下载到名为 **myproject** 的文件夹，请运行：

```
camel init https://github.com/apache/camel-kamelets-examples/tree/main/jbang/dependency-injection
--directory=myproject
```



注意

使用 **--directory** 选项时，如果已存在，Camel 将自动清理此目录。

您可以在 dev 模式下运行示例，以热部署源代码更改。

```
camel run * --dev
```

您可以下载单个文件，例如要下载其中一个 Camel K 示例，请运行：

```
camel init https://github.com/apache/camel-k-examples/blob/main/generic-examples/languages/simple.groovy
```

这是一个 groovy 路由，您可以使用 （或使用 *）运行：

```
camel run simple.groovy
```

5.3.11.1. 下载路由表单 GitHub gists

您可以从 gists 下载文件，如下所示：

```
camel init https://gist.github.com/davsclaus/477ddff5cdeb1ae03619aa544ce47e92
```

这会将文件下载到本地磁盘，稍后您可以运行：

```
camel run *
```

您可以使用 **--directory** 选项下载至新文件夹，例如，要下载到名为 **foobar** 的文件夹，请运行：

```
camel init https://gist.github.com/davsclaus/477ddff5cdeb1ae03619aa544ce47e92 --directory=foobar
```

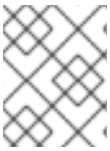
注意

使用 **--directory** 选项时，如果已存在，Camel 会自动清理此目录。

5.3.12. 使用特定的 Camel 版本

您可以指定要运行哪个 Camel 版本，如下所示：

```
jbang run -Dcamel.jbang.version=4.0.0 camel@apache/camel [command]
```



注意

较旧版本的 Camel 可能无法使用 Camel CLI 作为最新版本。建议您使用从 Camel 3.18 开始的版本。

您还可以使用 SNAPSHOT 尝试加快边缘开发，例如：

```
jbang run --fresh -Dcamel.jbang.version=4.0.0-SNAPSHOT camel@apache/camel [command]
```

5.3.13. 运行 Camel K 集成或绑定

Camel 支持以 CRD 格式(Kubernetes 自定义资源定义)。对于运行名为 joke.yaml 的 kamelet 绑定文件，运行名为 **joke.yaml** 的 Camel K 集成和绑定文件：

```
#!/usr/bin/env jbang camel@apache/camel run
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: joke
```

```
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1
      name: chuck-norris-source
    properties:
      period: 2000
  sink:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1
      name: log-sink
    properties:
      show-headers: false
```

```
camel run joke.yaml
```

5.3.14. 从剪贴板中运行

您可以直接从 OS 剪贴板运行 Camel 路由。这允许复制一些代码，然后快速运行路由。

```
camel run clipboard.<extension>
```

其中 **<extension>**，是剪贴板内容的类型，如 **java**、**xml** 或 **yaml**。

例如，您可以将它复制到剪贴板，然后运行路由：

```
<route>
  <from uri="timer:foo"/>
  <log message="Hello World"/>
</route>
```

```
camel run clipboard.xml
```

5.3.15. 控制本地 Camel 集成

要列出当前运行的 Camel 集成，请使用 **ps** 选项：

```
camel ps
  PID   NAME                    READY  STATUS    AGE
61818  sample.camel.MyCamelApplica…  1/1   Running  26m38s
62506  test1                   1/1   Running   4m34s
```

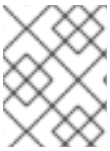
这将列出集成的 PID、名称和年龄。

您可以使用 **stop** 命令停止任何正在运行的 Camel 集成。例如，要停止 **test1**，请运行：

```
camel stop test1
Stopping running Camel integration (pid: 62506)
```

您可以使用 PID 来停止集成：

```
camel stop 62506
Stopping running Camel integration (pid: 62506)
```



注意

您不必输入全名，因为 stop 命令将与以输入开头的集成匹配，例如，您可以键入 **camel stop t** 来停止所有以 **t** 开头的集成。

要停止所有集成，请使用 **--all** 选项，如下所示：

```
camel stop --all
Stopping running Camel integration (pid: 61818)
Stopping running Camel integration (pid: 62506)
```

5.3.16. 控制 Spring Boot 和 Quarkus 集成

默认情况下，Camel CLI 仅控制使用 CLI 运行的 Camel 集成，例如 **camel run foo.java**。

要使 CLI 能够控制和管理 Spring Boot 或 Quarkus 应用程序，您需要向这些项目添加依赖项，以便与 Camel CLI 集成。

Spring Boot

在 Spring Boot 应用程序中，添加以下依赖项：

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-cli-connector-starter</artifactId>
</dependency>
```

Quarkus

在 Quarkus 应用程序中，添加以下依赖项：

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-cli-connector</artifactId>
</dependency>
```

5.3.17. 获取 Camel 集成状态

Camel CLI 中的 **get** 命令用于为一个或多个正在运行的 Camel 集成获取 Camel 特定状态。要显示正在运行的 Camel 集成的状态，请运行：

```
camel get
  PID   NAME      CAMEL          PLATFORM          READY  STATUS    AGE    TOTAL  FAILED
INFLIGHT  SINCE-LAST
 61818  MyCamel   3.20.1-SNAPSHOT  Spring Boot v2.7.3   1/1   Running  28m34s    854      0
0     0s/0s/-
 63051  test1     3.20.1-SNAPSHOT  JBang             1/1   Running     18s     14       0        0     0s/0s/-
 63068  mygroovy  3.20.1-SNAPSHOT  JBang             1/1   Running      5s      2       0        0
0s/0s/-
```

camel get 命令显示默认集成，它等同于输入 **camel get integrations** 或 **camel get int** 命令。

这会显示每个 Camel 集成的整体信息，您可以在其中查看处理的消息总数。列 **Since Last** 显示最后一次处理三个阶段的消息前的时长(started/completed/failed)。

0s/0s/- 值表示刚刚发生最后一个启动和完成的消息(0 秒前)，并且还没有任何失败的消息。在这个示例中，**9s/9s/1h3m** 表示最后启动和完成的消息是 9 秒前，最后失败为 1 小时，3 分钟以前。

您还可以查看每个路由的状态，来自所有本地 Camel 与 **camel get 路由** 集成：

```
camel get route
  PID   NAME     ID      FROM                    STATUS    AGE   TOTAL  FAILED  INFLIGHT  MEAN
MIN  MAX  SINCE-LAST
 61818  MyCamel  hello   timer://hello?period=2000   Running  29m2s   870      0        0     0    0   14
0s/0s/-
 63051  test1    java    timer://java?period=1000    Running   46s    46       0        0     0    0    9
0s/0s/-
 63068  mygroovy groovy  timer://groovy?period=1000  Running   34s    34       0        0     0    0    5
0s/0s/-
```

注意

使用 **camel get --help** 显示所有可用的命令。

5.3.17.1. Camel 集成的主要状态

camel top 命令用于获取正在运行的 Camel 集成的最高利用率统计（最高为最低堆使用的内存）。

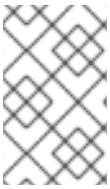
```
camel top
  PID  NAME     JAVA    CAMEL          PLATFORM          STATUS    AGE       HEAP        NON-
HEAP     GC     THREADS   CLASSES
 22104  chuck    11.0.13  3.20.1-SNAPSHOT  JBang             Running   2m10s  131/322/4294 MB
70/73 MB  17ms (6)     7/8  7456/7456
 14242  MyCamel  11.0.13  3.20.1-SNAPSHOT  Spring Boot v2.7.3  Running  33m40s
115/332/4294 MB  62/66 MB  37ms (6)    16/16  8428/8428
 22116  bar      11.0.13  3.20.1-SNAPSHOT  JBang             Running    2m7s   33/268/4294 MB
54/58 MB  20ms (4)     7/8  6104/6104
```

HEAP 列显示堆内存(used/committed/max)和 non-heap (used/committed)。GC 列显示垃圾回收信息（时间和总计运行）。CLASSES 列显示类数(loaded/total)。

您还可以从与 camel 顶部 路由的所有本地 Camel 集成中看到每个路由的最高执行路由（最高为最低的意味着处理时间）：

```
camel top route
  PID  NAME     ID                  FROM                          STATUS    AGE    TOTAL  FAILED
INFLIGHT  MEAN  MIN  MAX  SINCE-LAST
 22104  chuck    chuck-norris-source-1  timer://chuck?period=10000        Started    10s     1
```

```
0       0   163  163  163        9s
 22116  bar     route1             timer://yaml2?period=1000          Started     7s     7      0       0
1    0   11        0s
 22104  chuck   chuck              kamelet://chuck-norris-source      Started     10s    1      0
0    0    0    0        9s
 22104  chuck   log-sink-2         kamelet://source?routeId=log-sink-2  Started     10s    1      0
0    0    0    0        9s
 14242  MyCamel  hello             timer://hello?period=2000          Started  31m41s    948      0
0    0    0    4        0s
```



注意

使用 **camel top --help** 显示所有可用的命令。

5.3.17.2. 启动和停止路由

camel cmd 用于在运行的 **Camel** 集成中执行各种命令，例如，用于启动和停止路由的命令。

要停止 **chuck** 集成中的所有路由，请运行：

```
camel cmd stop-route chuck
```

然后，对于 **chuck** 集成，状态将更改为 **Stopped** ：

```
camel get route
  PID   NAME     ID                 FROM                          STATUS    AGE   TOTAL  FAILED
INFLIGHT  MEAN  MIN  MAX  SINCE-LAST
 81663  chuck    chuck              kamelet://chuck-norris-source      Stopped         600      0
0    0    0    1        4s
 81663  chuck    chuck-norris-source-1  timer://chuck?period=10000         Stopped         600
0       0    65   52  290        4s
 81663  chuck    log-sink-2         kamelet://source?routeId=log-sink-2  Stopped         600      0
0    0    0    1        4s
 83415  bar      route1             timer://yaml2?period=1000          Started  5m30s    329      0
0    0    0   10        0s
 83695  MyCamel  hello              timer://hello?period=2000          Started  3m52s    116      0
0    0    0    9        1s
```

要启动路由，请运行：

```
camel cmd start-route chuck
```

要停止每个 Camel 集成中的所有路由，请使用 -- all 标志，如下所示：

```
camel cmd stop-route --all
```

要启动 所有路由，请使用：

```
camel cmd start-route --all
```

注意

您可以使用逗号分隔一个或多个路由来停止一个或多个路由，例如 camel cmd start-route --id=route1,hello。使用 camel cmd start-route --help 命令获取更多详细信息。

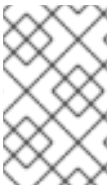
5.3.17.3. 配置日志记录级别

您可以通过以下方法查看正在运行的 Camel 集成的当前日志记录级别：

```
camel cmd logger
 PID   NAME   AGE   LOGGER  LEVEL
90857  bar   2m48s  root    INFO
91103  foo    20s   root    INFO
```

日志记录级别可以在运行时更改。例如，要将 foo 的级别改为 DEBUG，请运行：

```
camel cmd logger --level=DEBUG foo
```



注意

您可以使用 --all 更改所有正在运行的集成的日志级别。

5.3.17.4. 列出服务

有些 Camel 集成可以托管客户端可以使用 TCP 协议调用的服务，如 REST 或 SOAP-WS 或套接字级服务。您可以列出可用的服务，如下例所示：

```
camel get service
 PID   NAME      COMPONENT     PROTOCOL  SERVICE
1912  netty     netty         tcp       tcp:localhost:4444
```

```
2023 greetings platform-http rest     http://0.0.0.0:7777/camel/greetings/{name} (GET)
2023 greetings platform-http http     http://0.0.0.0:7777/q/dev
```

在这里，您可以看到两个 Camel 集成。netty 集成托管端口 4444 上可用的 TCP 服务。其他 Camel 集成托管只能通过 GET 调用的 REST 服务。第三个集成附带嵌入式 Web 控制台（通过 --console 选项启动）。

注意

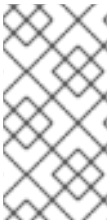
若要列出服务，Camel 组件必须能够使用 Camel 控制台 来 公告服务。

5.3.17.4.1. 列出 Circuit Breakers 的状态

如果您的 Camel 集成使用 link:https://camel.apache.org/components/3.20.x/eips/circuitBreaker-eip.html [Circuit Breaker]，您可以使用 Camel CLI 输出断器的状态，如下所示：

```
camel get circuit-breaker
 PID  NAME  COMPONENT    ROUTE  ID             STATE      PENDING  SUCCESS  FAIL
REJECT
 56033  mycb  resilience4j  route1  circuitBreaker1  HALF_OPEN      5      2    3      0
```

在这里，我们可以看到断路器处于 一半的开放 状态，即在故障开始丢弃时 breaker 试图过渡到关闭的状态。

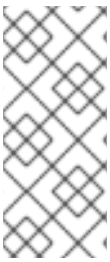


注意

您可以使用 watch 选项运行 命令，以显示最新的状态，例如 watch camel get circuit-breaker。

5.3.18. 使用管道从终端的脚本

您可以将 Camel CLI 文件作为脚本执行，该脚本用于通过管道和过滤器进行终端脚本。



注意

每次执行 JVM 时，都通过 Camel 启动脚本。这不适用于内存用量，因此请使用 Camel CLI 终端脚本，例如，使用许多 Camel 组件或 Kamelets 来更轻松地从不同的 IT 系统发送或接收数据。

这需要在文件顶部添加以下行，例如在 upper.yaml 文件中：

```
///usr/bin/env jbang --quiet camel@apache/camel pipe "$0" "$@" ; exit $?

# Will upper-case the input
- from:
    uri: "stream:in"
    steps:
      - setBody:
          simple: "${body.toUpperCase()}"
      - to: "stream:out"
```

要将其作为脚本执行，您需要设置执行文件权限：

```
chmod +x upper.yaml
```

然后您可以将其作为脚本执行：

```
echo "Hello\nWorld" | ./upper.yaml
```

这个输出：

```
HELLO
WORLD
```

您可以使用 --logging=true 打开日志，然后记录到 .camel-jbang/camel-pipe.log 文件。无法配置日志记录文件的名称。

```
echo "Hello\nWorld" | ./upper.yaml --logging=true
```

5.3.18.1. 使用 stream:in with line vs raw 模式

当使用 stream:in 从 System 读取数据时，流组件 以两种模式工作：

- 行模式（默认）- 以单行形式读取输入（用换行符分开）。邮件正文 是一个字符串。
- raw 模式 - 读取整个流，直到 *流 结束*。邮件正文是一个 字节[]。

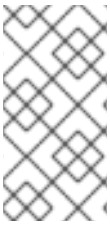
注意

默认模式是因为过去是如何创建流组件造成的。因此，您可能希望将 stream:in?readLine=false 设置为使用 raw 模式。

5.3.19. 运行本地 Kamelets

您可以使用 Camel CLI 尝试本地 Kamelets，而无需将其发布到 GitHub 或以 jar 打包它们。

```
camel run --local-kamelet-dir=/path/to/local/kamelets earthquake.yaml
```



注意

当 kamelets 来自本地文件系统时，当以 --dev 模式运行 Camel CLI 时，可以实时重新加载它们。

您还可以指向 GitHub 存储库中的文件夹。例如：

```
camel run --local-kamelet-dir=https://github.com/apache/camel-kamelets-examples/tree/main/custom-kamelets user.java
```

注意

如果从 GitHub 加载了 kamelet，则无法实时迁移它们。

5.3.20. 使用 platform-http 组件

从 platform-http 启动路由时，Camel CLI 会自动包含在端口 8080 上运行的 VertX HTTP 服务器。以下示例显示了名为 server.yaml 文件中的路由：

```
- from:
    uri: "platform-http:/hello"
    steps:
      - set-body:
          constant: "Hello World"
```

您可以使用以下方法运行这个示例：

```
camel run server.yaml
```

然后使用以下方法调用 HTTP 服务：

```
$ curl http://localhost:8080/hello
Hello World%
```

5.3.21. 使用 Java Bean 和处理器

基本支持包括常规 Java 源文件以及 Camel 路由，并让 Camel CLI 运行时编译 Java 源。这意味着您可以包含更小的工具类、POJO、Camel Processors。

注意

Java 源文件无法使用软件包名称。

5.3.22. Java 类中的依赖注入

当运行 Camel 与 camel-jbang 的集成时，运行时会基于 camel-main。这意味着没有 Spring Boot 或 Quarkus 可用。但是，支持在 Java 类中使用基于注解的依赖项注入。

5.3.22.1. 使用 Spring Boot 依赖项注入

您可以使用以下 Spring Boot 注解：

- @org.springframework.stereotype.Component 或 @org.springframework.stereotype.Service on 类级别，以创建类的实例并在 Registry 中注册。
- @org.springframework.beans.factory.annotation.Autowired 以在类字段中注入 bean。@org.springframework.beans.factory.annotation.Qualifier 可用于指定 bean id。
- @org.springframework.beans.factory.annotation.Value 以注入 属性占位符。例如 application.properties 中定义的属性。
- @org.springframework.context.annotation.Bean 在调用方法来创建 bean。

5.3.23. 调试

可用的调试有两种：

- Java 调试 - Java 代码调试（标准 Java）
- Camel 路由调试 - 调试 Camel 路由（需要 Camel 工具插件）

5.3.23.1. Java 调试

您可以使用 JBang 提供的 --debug 标志来调试集成脚本。但是，要在启动 JVM 时启用 Java 调试，请使用 jbang 命令，而不是 camel，如下所示：

```
jbang --debug  camel@apache/camel run hello.yaml
Listening for transport dt_socket at address: 4004
```

正如您所见，默认侦听端口为 4004，但可以按照 JBang 调试 中所述进行配置。

这是标准的 Java 调试套接字。然后您可以使用您选择的 IDE。您可以添加处理器，以在路由执行期间放置断点命中（而不是路由定义创建）。

5.3.23.2. Camel 路由调试

Camel 路由调试器默认可用(camel-debug 组件会自动添加到 classpath 中)。默认情况下，可以通过 URL 服务访问 JMX：jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi/camel。然后，您可以使用您选择的集成开发环境(IDE)。

5.3.24. 健康检查

使用 Camel CLI 从 CLI 访问健康检查的状态，如下所示：

```
camel get health
 PID   NAME    AGE  ID          RL  STATE  RATE   SINCE   MESSAGE
61005  mybind   8s  camel/context   R   UP   2/2/-  1s/3s/-
```

您可以在此处看到 Camel 为 UP。该应用已经运行 8 秒，并且调用了两个健康检查。

输出显示 默认 检查级别：

- CamelContext 健康检查
- 组件特定的健康检查（如 camel-kafka 或 camel-aws）
- 自定义健康检查
- 任何不是 UP的检查

RATE 列显示三个数字，用 / 分隔。因此，2/2/- 表示共 2 个检查，2 个成功且没有失败。当健康检查更改状态时，两个列将重置，因为这个数字是成功或失败的连续检查数量。因此，如果健康检查开始失败，则数字可以是：

```
camel get health
  PID   NAME     AGE   ID           RL  STATE   RATE    SINCE    MESSAGE
 61005  mybind   3m2s  camel/context   R   UP    77/-/3  1s/-/17s  some kind of error
```

您可以在此处看到数字已更改为 77/-/3。这意味着检查的总数为 77。没有成功，但检查连续 3 次失败。SINCE 列与 RATE 对应。因此，在这种情况下，您可以看到最后的检查是 1 秒，并且检查在一行中已失败 17 秒。

您可以使用 --level=full 输出包含消费者和路由级别检查的每个健康检查。

健康检查可能会因为抛出异常而失败，这可使用 --trace 标志显示：

```
camel get health --trace
  PID   NAME      AGE   ID                                RL  STATE    RATE      SINCE     MESSAGE
 61038  mykafka  6m19s  camel/context                        R   UP    187/187/-  1s/6m16s/-
 61038  mykafka  6m19s  camel/kafka-consumer-kafka-not-secure...   R  DOWN   187/-/187  1s/-
/6m16s  KafkaConsumer is not ready - Error: Invalid url in bootstrap.servers: value


-------------------------------------------------------------------------------------------------------------------
                                        STACK-TRACE
-------------------------------------------------------------------------------------------------------------------
   PID: 61038
   NAME: mykafka
```

```
   AGE: 6m19s
   CHECK-ID: camel/kafka-consumer-kafka-not-secured-source-1
   STATE: DOWN
   RATE: 187
   SINCE: 6m16s
   METADATA:
      bootstrap.servers = value
      group.id = 7d8117be-41b4-4c81-b4df-cf26b928d38a
      route.id = kafka-not-secured-source-1
      topic = value
   MESSAGE: KafkaConsumer is not ready - Error: Invalid url in bootstrap.servers: value
   org.apache.kafka.common.KafkaException: Failed to construct kafka consumer
      at org.apache.kafka.clients.consumer.KafkaConsumer.<init>(KafkaConsumer.java:823)
      at org.apache.kafka.clients.consumer.KafkaConsumer.<init>(KafkaConsumer.java:664)
      at org.apache.kafka.clients.consumer.KafkaConsumer.<init>(KafkaConsumer.java:645)
      at org.apache.kafka.clients.consumer.KafkaConsumer.<init>(KafkaConsumer.java:625)
      at
org.apache.camel.component.kafka.DefaultKafkaClientFactory.getConsumer(DefaultKafkaClie
ntFactory.java:34)
      at
org.apache.camel.component.kafka.KafkaFetchRecords.createConsumer(KafkaFetchRecords.
java:241)
      at
org.apache.camel.component.kafka.KafkaFetchRecords.createConsumerTask(KafkaFetchRec
ords.java:201)
      at org.apache.camel.support.task.ForegroundTask.run(ForegroundTask.java:123)
      at
org.apache.camel.component.kafka.KafkaFetchRecords.run(KafkaFetchRecords.java:125)
      at java.base/java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:515)
      at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
      at
java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
      at
java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)
      at java.base/java.lang.Thread.run(Thread.java:829)
   Caused by: org.apache.kafka.common.config.ConfigException: Invalid url in
bootstrap.servers: value
      at org.apache.kafka.clients.ClientUtils.parseAndValidateAddresses(ClientUtils.java:59)
      at org.apache.kafka.clients.ClientUtils.parseAndValidateAddresses(ClientUtils.java:48)
      at org.apache.kafka.clients.consumer.KafkaConsumer.<init>(KafkaConsumer.java:730)
      ... 13 more
```

在这里，您可以看到由于 org.apache.kafka.common.config.ConfigException 导致健康检查失败，这源自无效的配置： Invalid url in bootstrap.servers: value。

注意

使用 camel get health --help 查看所有各种选项。

5.4. 列出哪些 CAMEL 组件可用

Camel 附带很多工件，即：

- components
- 数据格式
- 表达式语言
- 其他组件
- kamelets

您可以使用 Camel CLI 使用 camel catalog 命令列出 Camel 提供的 Camel。例如，列出所有组件：

```
camel catalog components
```

查看哪些 Kamelets 可用：

```
camel catalog kamelets
```

注意

使用 camel catalog --help 查看所有可能的命令。

5.4.1. 显示组件文档

文档 目标可以显示每个组件、dataformat 和 kamelets 的快速文档。例如，要查看 kafka 组件运行：

```
camel doc kafka
```

注意

文档不是网站上显示的完整文档，因为 Camel CLI 无法直接访问此信息，并且只能显示组件的基本描述，但包括每个配置选项的表。

要查看 jackson dataformat 的文档：

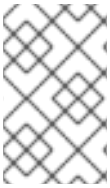
```
camel doc jackson
```

在某些情况下，可能存在一个组件，它的名称相同，doc 目标会优先选择组件。在这种情况下，您可以使用 dataformat 为名称添加前缀，例如：

```
camel doc dataformat:thrift
```

您还可以查看 kamelet 文档，如下所示：

```
camel doc aws-kinesis-sink
```



注意

如需支持的 kamelets 列表，请参阅支持的 Kamelets。

5.4.1.1. 从 Camel 网站浏览在线文档

您可以使用 doc 命令在网页浏览器中快速打开在线文档的 url。例如，要浏览 kafka 组件，请使用 --open-url ：

```
camel doc kafka --open-url
```

这也适用于数据格式、语言、kamelets。

```
camel doc aws-kinesis-sink --open-url
```

注意

要只获取在线文档的链接，请使用 camel doc kafka --url。

5.4.1.2. 过滤表中列出的选项

有些组件可能有多个选项，在这种情况下，您可以使用 --filter 选项只列出与名称、描述或组 (producer, security, advanced)匹配的过滤器的选项。

例如，仅列出与安全相关的选项：

```
camel doc kafka --filter=security
```

仅列出与 超时 相关的内容：

```
camel doc kafka --filter=timeout
```

5.5. 收集依赖项列表

使用 Camel CLI 时，依赖项会自动解决。这意味着您不必使用 Maven 或 Gradle 等构建系统来添加每个 Camel 组件作为依赖项。

但是，您可能希望知道运行 Camel 集成需要哪些依赖项。您可以使用 dependencies 命令查看所需的依赖项。命令输出不会输出详细的树，如 mvn dependencies:tree，因为输出旨在列出所需的 Camel 组件和其他 JAR （使用 Kamelets 时）。

依赖项输出默认为 vanilla Apache Camel，将 camel-main 作为运行时，如下所示：

```
camel dependency
org.apache.camel:camel-dsl-modeline:4.0.0
org.apache.camel:camel-health:4.0.0
org.apache.camel:camel-kamelet:4.0.0
org.apache.camel:camel-log:4.0.0
org.apache.camel:camel-rest:4.0.0
org.apache.camel:camel-stream:4.0.0
org.apache.camel:camel-timer:4.0.0
org.apache.camel:camel-yaml-dsl:4.0.0
org.apache.camel.kamelets:camel-kamelets-utils:0.9.3
org.apache.camel.kamelets:camel-kamelets:0.9.3
```

默认情况下，输出是每个 maven 依赖项(groupId:artifactId:version)的行。

您可以为输出指定 Maven 格式，如下所示：

```
camel dependencies --output=maven
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-main</artifactId>
  <version>4.0.0</version>
</dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-dsl-modeline</artifactId>
  <version>4.0.0</version>
</dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-health</artifactId>
  <version>4.0.0</version>
</dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-kamelet</artifactId>
  <version>4.0.0</version>
</dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-log</artifactId>
  <version>4.0.0</version>
</dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-rest</artifactId>
  <version>4.0.0</version>
</dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-stream</artifactId>
  <version>4.0.0</version>
</dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-timer</artifactId>
  <version>4.0.0</version>
</dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-yaml-dsl</artifactId>
  <version>4.0.0</version>
</dependency>
<dependency>
  <groupId>org.apache.camel.kamelets</groupId>
  <artifactId>camel-kamelets-utils</artifactId>
  <version>0.9.3</version>
</dependency>
<dependency>
```

```
    <groupId>org.apache.camel.kamelets</groupId>
    <artifactId>camel-kamelets</artifactId>
    <version>0.9.3</version>
</dependency>
```

您还可以选择目标运行时作为'quarkus' 或 spring-boot，如下所示：

```
camel dependencies --runtime=spring-boot
org.springframework.boot:spring-boot-starter-actuator:3.1.4
org.springframework.boot:spring-boot-starter-web:3.1.4
org.apache.camel.springboot:camel-spring-boot-engine-starter:4.0.0
org.apache.camel.springboot:camel-dsl-modeline-starter:4.0.0
org.apache.camel.springboot:camel-kamelet-starter:4.0.0
org.apache.camel.springboot:camel-log-starter:4.0.0
org.apache.camel.springboot:camel-rest-starter:4.0.0
org.apache.camel.springboot:camel-stream-starter:4.0.0
org.apache.camel.springboot:camel-timer-starter:4.0.0
org.apache.camel.springboot:camel-yaml-dsl-starter:3.20
org.apache.camel.kamelets:camel-kamelets-utils:0.9.3
org.apache.camel.kamelets:camel-kamelets:0.9.3
```

5.6. OPEN API

Camel CLI 允许使用 合同第一 方法快速公开 Open API 服务，其中具有现有的 OpenAPI 规格文件。Camel CLI 使用命名规则 direct:<operationId> 将每个 API 端点从 OpenAPI 规格桥接到 Camel 路由。这使得为给定操作实施 Camel 路由速度更快。

如需了解更多详细信息，请参阅 OpenAPI 示例。

5.7. 故障排除

使用 JBang 时，它将状态存储在 ~/.jbang 目录中。这也是 JBang 存储下载的 JAR 的位置。Camel JBang 还在运行时下载所需的依赖项。但是，这些依赖项下载到您的本地 Maven 存储库 ~/.m2 中。因此，当您在运行 Camel JBang 时对一些问题进行故障排除时，请尝试删除这些目录或部分目录。

5.8. 导出到 RED HAT BUILD OF APACHE CAMEL FOR SPRING BOOT

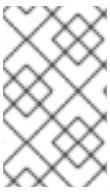
您可以将 Camel CLI 集成 导出到 基于 Java 的传统项目，如 Spring Boot 或 Quarkus。在使用 Camel CLI 构建原型后，可能需要执行此操作，并且需要传统的基于 Java 的项目，并需要更多对 Java 编码的需求，或者使用 Spring Boot、Quarkus 或 vanilla Camel Main 的强大运行时。

5.8.1. 导出到 Red Hat build of Apache Camel for Spring Boot

命令会将 --runtime=spring-boot 导出您的当前 Camel CLI 文件导出到 基于 Maven 的 Spring Boot 项目，文件整理在 src/main/ 文件夹结构中。

例如，要使用 Maven groupId com.foo 和 artifactId acme 和版本 1.0-SNAPSHOT 将 导出到 Spring Boot，请运行：

```
camel export --runtime=spring-boot --gav=com.foo:acme:1.0-SNAPSHOT
```



注意

这将导出到 当前目录，这意味着文件被移到所需的文件夹结构中。

要导出到另一个目录，请运行：

```
camel export --runtime=spring-boot --gav=com.foo:acme:1.0-SNAPSHOT --
directory=../myproject
```

当导出到 Spring Boot 时，pom.xml 或 build.gradle 中定义的 Camel 版本与 Camel CLI 使用的版本相同。但是，您可以指定不同的 Camel 版本，如下所示：

```
camel export --runtime=spring-boot --gav=com.foo:acme:1.0-SNAPSHOT --
directory=../myproject --camel-spring-boot-version=4.0.0.redhat-00039
```

注意

如需了解更多详细信息，运行 camel export --help 命令来查看可能的选项。

5.8.2. 包括使用 Camel CLI 导出

当导出到 Spring Boot、Quarkus 或 Camel Main 时，Camel JBang CLI 不会包含在框中。要继续使用 Camel CLI （即 camel），您需要在 --deps 选项中添加 camel:cli-connector，如下所示：

```
camel export --runtime=quarkus --gav=com.foo:acme:1.0-SNAPSHOT --deps=camel:cli-
connector --directory=../myproject
```

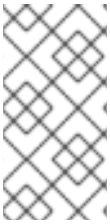
5.8.3. 配置导出

export 命令默认从 application.properties 文件加载配置，该文件用于导出特定参数，如选择运行时和 java 版本。

以下选项与 导出 相关，可在 application.properties 文件中配置：

选项	Description
camel.jbang.runtime	runtime (spring-boot, quarkus, 或 camel-main)
camel.jbang.gav	Maven group:artifact:version
camel.jbang.dependencies	其他依赖项（使用逗号分隔多个依赖项）。请参阅 添加自定义 JAR 的更多详细信息。
camel.jbang.classpathFiles	要添加到 classpath 的其他文件（使用逗号分隔多个文件）。请参阅 添加自定义 JAR 的更多详细信息。
camel.jbang.javaVersion	Java 版本(11 或 17)
camel.jbang.kameletsVersion	Apache Camel Kamelets 版本
camel.jbang.localKameletDir	加载 Kamelets 的本地目录
camel.jbang.camelSpringBootVersion	用于 Spring Boot 的 Camel 版本
camel.jbang.springBootVersion	Spring Boot 版本
camel.jbang.quarkusGroupId	Quarkus Platform Maven groupId
camel.jbang.quarkusArtifactId	quarkus Platform Maven artifactId
camel.jbang.quarkusVersion	Quarkus Platform 版本
camel.jbang.mavenWrapper	在导出的项目中包含 Maven Wrapper 文件
camel.jbang.gradleWrapper	在导出的项目中包括 Gradle Wrapper 文件
camel.jbang.buildTool	使用构建工具(maven 或 gradle)
camel.jbang.repos	用于按需下载的额外 maven 存储库（使用逗号分隔多个存储库）
camel.jbang.mavenSettings	maven setting.xml 文件的可选位置，以配置服务器、存储库、镜像和代理。如果设置为 false，则不会使用默认的 ~/.m2/settings.xml。

选项	Description
camel.jbang.mavenSettingsSecurity	maven settings-security.xml 文件的可选位置来解密 settings.xml
camel.jbang.exportDir	导出项目的目录。
camel.jbang.platform-http.port	运行独立 Camel 时要使用的 HTTP 服务器端口，如启用 --console 时（默认为端口 8080）。
camel.jbang.console	运行独立 Camel 时，位于本地 HTTP 服务器上的 /q/dev 的开发人员控制台（默认为 8080）。
camel.jbang.health	运行独立 Camel 时，本地 HTTP 服务器上的 /q/health （默认为 8080）进行健康检查。



注意

这些是 **export** 命令中的选项。您可以使用 **camel export --help** 查看更多详细信息和默认值。

5.8.4. 配置

Camel CLI config 命令用于存储和使用用户配置。这消除了每次指定 CLI 选项的需求。例如，要运行不同的 Camel 版本，请使用：

```
camel run * --camel-version=4.0.0.redhat-00031
```

camel-version 可以添加到用户配置中，例如：

```
camel config set camel-version=4.0.0.redhat-00031
```

run 命令使用用户配置：

```
camel run *
```

用户配置文件存储在 ~/.camel-jbang-user.properties 中。

5.8.4.1. 设置和取消设置配置

每个 Camel CLI 选项都添加到用户配置中。例如，要导出一个简单的项目，例如：

```
camel init foo.yaml
camel config set gav=com.foo:acme:1.0-SNAPSHOT
camel config set runtime=spring-boot
camel config set deps=org.apache.camel.springboot:camel-timer-starter
camel config set camel-spring-boot-version=4.0.0.redhat-00039
camel config set additional-properties=openshift-maven-plugin-version=1.13.1.redhat-00043

camel export
```

使用以下命令取消设置用户配置密钥：

```
camel config unset camel-spring-boot-version
```

5.8.4.2. 列出和获取配置

使用以下命令列出用户配置密钥：

```
camel config list
```

以上上述配置的输出如下：

```
runtime = spring-boot
deps = org.apache.camel.springboot:camel-timer-starter
gav = com.foo:acme:1.0-SNAPSHOT
```

要获取给定键的值，请使用 get 命令。

```
camel config get gav

com.foo:acme:1.0-SNAPSHOT
```

5.8.4.3. 占位符替换

用户配置值可作为占位符使用命令行属性替换，例如：

```
camel config set repos=https://maven.repository.redhat.com/ga

camel run 'Test.java' --logging-level=info --
repos=#repos,https://packages.atlassian.com/maven-external
```

-

在本例中，因为 repos 在用户配置(config set)中设置，而 camel run 命令会声明占位符 #repos，因此 camel run 将替换占位符，以便在执行过程中使用这两个存储库。请注意，要引用配置值，语法为 #optionName eg #repos。

注意

占位符替换仅适用于给定 Camel 命令具有的每个选项。您可以查看命令使用 camel run --help 的所有选项。

5.8.5. 故障排除

使用 JBang 时，它将状态存储在 ~/.jbang 目录中。这也是 JBang 存储下载的 JAR 的位置。Camel CLI 还在运行时下载所需的依赖项。但是，这些依赖项下载到您的本地 Maven 存储库 ~/.m2 中。因此，当您在运行 Camel CLI 时对一些问题（如过时的 JAR）进行故障排除时，请尝试删除这些目录或部分目录。

5.9. 导出到 RED HAT BUILD OF APACHE CAMEL FOR QUARKUS

您可以将 Camel CLI 集成 导出到 基于 Java 的传统项目。在使用 Camel CLI 构建原型后，可能需要执行此操作，并且需要传统的基于 Java 的项目，并需要更多 Java 编码，或使用强大的 Quarkus 或 vanilla Camel Main 运行时。

5.9.1. 导出到 Red Hat build of Apache Camel for Quarkus

命令会将 --runtime=quarkus 将您当前的 Camel CLI 文件导出到 基于 Maven 的项目，文件整理在 src/main/ 文件夹结构中。

例如，要使用 quarkus 运行时导出，maven groupID com.foo、artifactId acme，以及版本 1.0-SNAPSHOT 到 camel-quarkus-jbang 目录中，请运行：

```
camel export --runtime=quarkus --gav=com.foo:acme:1.0-SNAPSHOT --quarkus-group-id=com.redhat.quarkus.platform --quarkus-version=3.2.6.SP1_redhat_00001  --deps=org.apache.camel.quarkus:camel-quarkus-timer,org.apache.camel.quarkus:camel-quarkus-management,org.apache.camel.quarkus:camel-quarkus-cli-connector --repos=https://indy.psi.redhat.com/api/content/maven/group/static/ --directory=camel-quarkus-jbang
```

注意

这将导出到 当前目录，这意味着文件被移到所需的文件夹结构中。

要导出到另一个目录，请运行：

```
camel export --runtime=quarkus --gav=com.foo:acme:1.0-SNAPSHOT --directory=../myproject
```

导出时，pom.xml 或 build.gradle 中定义的 Camel 版本与 Camel CLI 使用的版本相同。但是，您可以指定不同的 Camel 版本，如下所示：

```
camel export --runtime=quarkus --gav=com.foo:acme:1.0-SNAPSHOT --directory=../myproject
--quarkus-version=3.2.6.SP1-redhat-00001
```

注意

如需了解更多详细信息，运行 camel export --help 命令来查看可能的选项。

5.9.2. 包括使用 Camel CLI 导出

当导出到 Quarkus 或 Camel Main 时，Camel JBang CLI 不会包含在框中。要继续使用 Camel CLI（即 camel），您需要在 --deps 选项中添加 camel:cli-connector，如下所示：

```
camel export --runtime=quarkus --gav=com.foo:acme:1.0-SNAPSHOT --deps=camel:cli-
connector --directory=../myproject
```

5.9.3. 配置导出

export 命令默认从 application.properties 文件加载配置，该文件用于导出特定参数，如选择运行时和 java 版本。

以下选项与 导出 相关，可在 application.properties 文件中配置：

选项	Description
camel.jbang.runtime	runtime (**quarkus**, 或 **camel-main**)

选项	Description
camel.jbang.gav	Maven group:artifact:version
camel.jbang.dependencies	其他依赖项（使用逗号分隔多个依赖项）。请参阅 添加自定义 JAR 的更多详细信息。
camel.jbang.classpathFiles	要添加到 classpath 的其他文件（使用逗号分隔多个文件）。请参阅 添加自定义 JAR 的更多详细信息。
camel.jbang.javaVersion	Java 版本(11 或 17)
camel.jbang.kameletsVersion	Apache Camel Kamelets 版本
camel.jbang.localKameletDir	加载 Kamelets 的本地目录
camel.jbang.quarkusGroupId	Quarkus Platform Maven groupId
camel.jbang.quarkusArtifactId	quarkus Platform Maven artifactId
camel.jbang.quarkusVersion	Quarkus Platform 版本
camel.jbang.mavenWrapper	在导出的项目中包含 Maven Wrapper 文件
camel.jbang.gradleWrapper	在导出的项目中包括 Gradle Wrapper 文件
camel.jbang.buildTool	使用构建工具(maven 或 gradle)
camel.jbang.repos	用于按需下载的额外 maven 存储库（使用逗号分隔多个存储库）
camel.jbang.mavenSettings	maven setting.xml 文件的可选位置，以配置服务器、存储库、镜像和代理。如果设置为 false，则不会使用默认的 ~/.m2/settings.xml。
camel.jbang.mavenSettingsSecurity	maven settings-security.xml 文件的可选位置来解密 settings.xml
camel.jbang.exportDir	导出项目的目录。
camel.jbang.platform-http.port	运行独立 Camel 时要使用的 HTTP 服务器端口，如启用 --console 时（默认为端口 8080）。
camel.jbang.console	运行独立 Camel 时，位于本地 HTTP 服务器上的 /q/dev 的开发人员控制台（默认为 8080）。
camel.jbang.health	运行独立 Camel 时，本地 HTTP 服务器上的 /q/health （默认为 8080）进行健康检查。

注意

这些是 **export** 命令中的选项。要查看更多详细信息和默认值，请运行： **camel export --help**。

5.9.4. 配置

Camel CLI config 命令用于存储和使用用户配置。这消除了每次指定 CLI 选项的需求。例如，要运行不同的 Camel 版本，请使用：

```
camel run * --camel-version=4.0.0.redhat-00031
```

camel-version 可以添加到用户配置中，例如：

```
camel config set camel-version=4.0.0.redhat-00031
```

run 命令使用用户配置：

```
camel run *
```

用户配置文件存储在 ~/.camel-jbang-user.properties 中。

5.9.4.1. 设置和取消设置配置

每个 Camel CLI 选项都添加到用户配置中。例如：

```
camel config set gav=com.foo:acme:1.0-SNAPSHOT
camel config set runtime=quarkus
camel config set deps=org.apache.camel.quarkus:camel-timer,camel:management,camel:cli-connector
camel config set camel-version=4.0.0.redhat-00031
camel config set camel-quarkus-version=3.2.0.redhat-00018
```

导出配置：

```
camel export
```

初始化 camel 应用程序：

```
camel init foo.yaml
```

运行 camel 应用程序：

```
camel run foo.yaml --repos=https://indy.psi.redhat.com/api/content/maven/group/static/
```

取消设置用户配置密钥：

```
camel config unset camel-quarkus-version
```

5.9.4.2. 列出和获取配置

使用以下命令列出用户配置密钥：

```
camel config list
```

以上上述配置的输出如下：

```
runtime = spring-boot
deps = org.apache.camel.springboot:camel-timer-starter
gav = com.foo:acme:1.0-SNAPSHOT
```

要获取给定键的值，请使用 get 命令。

```
camel config get gav

com.foo:acme:1.0-SNAPSHOT
```

5.9.4.3. 占位符替换

用户配置值可作为占位符使用命令行属性替换，例如：

```
camel config set repos=https://maven.repository.redhat.com/ga

camel run 'Test.java' --logging-level=info --
repos=#repos,https://packages.atlassian.com/maven-external
```

在本例中，因为 repos 在用户配置(config set)中设置，而 camel run 命令会声明占位符 #repos，因此 camel run 将替换占位符，以便在执行过程中使用这两个存储库。请注意，要引用配置值，语法为 #optionName eg #repos。

注意

占位符替换仅适用于给定 Camel 命令具有的每个选项。您可以查看命令使用 camel run --help 的所有选项。

5.9.5. 故障排除

使用 JBang 时，它将状态存储在 ~/.jbang 目录中。这也是 JBang 存储下载的 JAR 的位置。Camel CLI 还在运行时下载所需的依赖项。

但是，这些依赖项下载到您的本地 Maven 存储库 ~/.m2 中。因此，当您在运行 Camel CLI 时对一些问题（如过时的 JAR）进行故障排除时，请尝试删除这些目录或部分目录。