Red Hat build of Quarkus 3.8

使用 Apache Maven 开发并编译您的红帽构建的 Quarkus 应用程序

Last Updated: 2024-05-17

Red Hat build of Quarkus 3.8 使用 Apache Maven 开发并编译您的红帽构建的 Quarkus 应用程序

法律通告

摘要

本指南论述了如何使用 Apache Maven 工具开发和编译红帽构建的 Quarkus 应用程序。

目录

提供有关红帽构建的 QUARKUS 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. 单击以下链接 **以创建 ticket**。
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 属性中有问题的语言。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 APACHE MAVEN 开发并编译您的红帽构建的 QUARKUS 应用程序

作为应用程序开发人员，您可以使用 Red Hat build of Quarkus 创建使用 Java 编写的基于微服务的应用程序，这些应用程序在 OpenShift Container Platform 和无服务器环境中运行。编译到原生可执行文件的应用程序具有较少的内存占用率和快速启动时间。

使用 Quarkus Apache Maven 插件创建红帽构建的 Quarkus 项目。

注意

如果适用，提供了使用 Quarkus 命令行界面(CLI)的替代说明。Quarkus CLI 仅用于 dev 模式。红帽不支持在生产环境中使用 Quarkus CLI。

先决条件

- 已安装 OpenJDK 17 或 21。
  - 要下载红帽构建的 OpenJDK，请登录到红帽客户门户网站，再进入 Software Downloads。
- 您已设置了 **JAVA_HOME** 环境变量，以指定 Java SDK 的位置。
- 已安装 Apache Maven 3.8.6 或更高版本。
  - 要下载 Maven，请转至 Apache Maven 项目网站。

1.1. 关于红帽构建的 QUARKUS

Red Hat build of Quarkus 是一个 Kubernetes 原生 Java 堆栈，针对容器和 Red Hat OpenShift Container Platform 进行了优化。Quarkus 设计为使用流行的 Java 标准、框架和库，如 Eclipse MicroProfile、Eclipse Vert.x、Apache Camel、Apache Kafka、Hibernate ORM 和 RESTEasy Reactive (Jakarta REST)。

作为开发人员，您可以选择 Java 应用所需的 Java 框架，您可以在 Java 虚拟机(JVM)模式下运行，或者以原生模式运行。Quarkus 提供了构建 Java 应用程序的容器优先方法。容器先行方法促进微服务和功能的容器化和高效执行。因此，Quarkus 应用程序具有较小的内存空间和更快的启动时间。

Quarkus 还通过统一配置、自动配置未配置的服务、实时编码和持续测试等功能优化应用程序开发流程，为您提供对代码更改的即时反馈。

有关 Quarkus 社区版本和红帽构建的 Quarkus 之间的差异的详情，请参考 Quarkus 社区版本和红帽构建的 Quarkus 之间的 Differences。

1.2. 关于 APACHE MAVEN 和红帽构建的 QUARKUS

Apache Maven 是一个分布式构建自动化工具，用于 Java 应用程序开发，用于创建、管理和构建软件项目。

要了解有关 Apache Maven 的更多信息，请参阅 Apache Maven 文档。

Maven 存储库

Maven 存储库存储 Java 库、插件和其他构建构件。默认公共存储库是 Maven 2 Central Repository，但存储库可以是私有和内部的，以在开发团队之间共享通用工件。也可从第三方提供存储库。

您可以将 Red Hat-hosted Maven 存储库与 Quarkus 项目搭配使用，也可以下载红帽构建的 Quarkus Maven 存储库。

Maven 插件

Maven 插件是 POM 文件的定义部分，用于运行一个或多个任务。红帽构建的 Quarkus 应用程序使用以下 Maven 插件：

- *Quarkus Maven 插件(**quarkus-maven-plugin**)* ：启用 Maven 来创建 Quarkus 项目，将应用程序打包到 JAR 文件中，并提供 dev 模式。
- *Maven Surefire 插件(**maven-surefire-plugin**)*: 当 Quarkus 启用 **测试** 配置集时，会在构建生命周期的测试阶段使用 Maven Surefire 插件在应用程序上运行单元测试。**该插件生成包含测试报告的文本和 XML 文件。**

其他资源

- **配置红帽构建的 Quarkus 应用程序**

1.2.1. 为在线存储库配置 Maven settings.xml 文件

要将 Red Hat-hosted Quarkus 存储库与您的 Quarkus Maven 项目搭配使用，请为您的用户配置 settings.xml 文件。与存储库管理器或共享服务器上的存储库一起使用的 Maven 设置可以提供更好的项目控制和易管理性。

注意

当您修改 Maven settings.xml 文件来配置存储库时，这些更改将应用到所有 Maven 项目。如果只想将配置应用到特定的项目，请使用 -s 选项并指定特定于项目的 settings.xml 文件的路径。

流程

1. **在文本编辑器中或集成开发环境(IDE)中打开 Maven $HOME/.m2/settings.xml 文件。**

注意

如果 $HOME/.m2/ 目录中没有 settings.xml 文件，请将 settings.xml 文件从 $MAVEN_HOME/conf/ 目录中复制到 $HOME/.m2/ 目录中。

2. **在 settings.xml 文件的 <profiles > 元素中添加以下行：**

```
<!-- Configure the Red Hat build of Quarkus Maven repository -->
<profile>
 <id>red-hat-enterprise-maven-repository</id>
 <repositories>
  <repository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>https://maven.repository.redhat.com/ga/</url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>
    <enabled>false</enabled>
   </snapshots>
  </repository>
 </repositories>
 <pluginRepositories>
  <pluginRepository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>https://maven.repository.redhat.com/ga/</url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>
    <enabled>false</enabled>
   </snapshots>
  </pluginRepository>
 </pluginRepositories>
</profile>
```

3. 在 **settings.xml** 文件的 **< activeProfiles >** 元素中添加以下行并保存文件。

```
<activeProfile>red-hat-enterprise-maven-repository</activeProfile>
```

1.3. 在命令行中创建红帽构建的 QUARKUS 项目

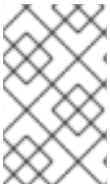
在命令行中使用 **Red Hat build of Quarkus Maven** 插件创建一个 **Quarkus** 项目，方法是在命令行上提供属性和值，或者在交互模式中使用插件。您还可以使用 **Quarkus** 命令行界面(CLI)创建 **Quarkus** 项目。生成的项目包括以下元素：

- **Maven** 结构
- 关联的单元测试
- 启动应用程序后，可在 **http://localhost:8080** 上访问的登录页面

- src/main/docker中的 JVM 和原生模式的 Dockerfile 文件示例
- 应用程序配置文件

先决条件

- 已安装 OpenJDK 17 或 21。
  - 要下载红帽构建的 OpenJDK，请登录到红帽客户门户网站，再进入 Software Downloads。
- 您已设置了 JAVA_HOME 环境变量，以指定 Java SDK 的位置。
- 已安装 Apache Maven 3.8.6 或更高版本。
  - 要下载 Maven，请转至 Apache Maven 项目网站。
- 已安装 Quarkus 命令行界面(CLI)，这是可用于创建 Quarkus 项目的方法之一。如需更多信息，请参阅安装 Quarkus CLI。



注意

Quarkus CLI 仅用于 dev 模式。红帽不支持在生产环境中使用 Quarkus CLI。

流程

1. 在命令终端中，输入以下命令验证 Maven 是否使用 OpenJDK 17 或 21，并且 Maven 版本是否为 3.8.6 或更高版本：

```
mvn --version
```

2. 如果前面的命令没有返回 OpenJDK 17 或 21，请将到 OpenJDK 17 或 21 的路径添加到 PATH 环境变量中，然后再次输入前面的命令。

3. 要创建项目，请使用以下方法之一：

- 使用 Quarkus Maven 插件。输入以下命令：

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:3.8.4.redhat-00002:create \
    -DprojectGroupId=<project_group_id> \
    -DprojectArtifactId=<project_artifact_id> \
    -DplatformGroupId=com.redhat.quarkus.platform \
    -DplatformArtifactId=quarkus-bom \
    -DplatformVersion=3.8.4.redhat-00002
    -DpackageName=getting.started
```

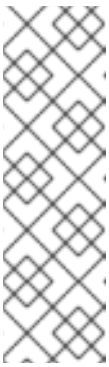
在这个命令中，替换以下值：

  - <project_group_id >：项目的唯一标识符
  - <project_artifact_id >：项目名称和项目目录的名称

- 以互动模式创建项目：

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:3.8.4.redhat-00002:create
```

出现提示时，输入所需的属性值。



注意

您还可以输入以下命令使用项目属性的默认值创建项目：

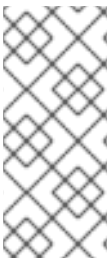
```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:3.8.4.redhat-00002:create -B
```

- 使用 Quarkus CLI。输入以下命令：

```
quarkus create app my-groupId:my-artifactId --package-name=getting.started
```

- 您还可以通过以下方式获取可用选项列表：

```
quarkus create app --help
```



注意

默认情况下，Quarkus Maven 插件使用最新的 quarkus-bom 版本。quarkus-bom 文件聚合扩展，以便您可以从应用程序中引用它们以匹配依赖项版本。当您离线时，Quarkus Maven 插件使用从 Maven 存储库拉取更新的最新 quarkus-bom 版本。

1.4. 通过配置 POM.XML 文件来创建红帽构建的 QUARKUS 项目

您可以通过配置 Maven pom.xml 文件来创建 Quarkus 项目。

流程

1. 在文本编辑器中打开 pom.xml 文件。
2. 添加包含以下项目的配置属性：
   - Maven Compiler 插件版本
   - Quarkus BOM groupID,artifactID, 和 version
   - Maven Surefire 插件版本
   - skipITs 属性。

```
<properties>
  <compiler-plugin.version>3.11.0</compiler-plugin.version>
  <quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
  <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
  <quarkus.platform.version>3.8.4.redhat-00002</quarkus.platform.version>
```

```
    <surefire-plugin.version>3.1.2</surefire-plugin.version>
    <skipITs>true</skipITs>
</properties>
```

3. **添加 Quarkus GAV （组、工件、版本），并使用 quarkus-bom 文件省略不同 Quarkus 依赖项的版本：**

```
<dependencyManagement>
  <dependencies>
    <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>
  </dependencies>
</dependencyManagement>
```

4. **添加 Quarkus Maven 插件、Maven Compiler 插件和 Maven Surefire 插件：**

```
<build>
  <plugins>
    <plugin>
      <groupId>${quarkus.platform.group-id}</groupId>
      <artifactId>quarkus-maven-plugin</artifactId>
      <version>${quarkus.platform.version}</version>
      <extensions>true</extensions>
      <executions>
        <execution>
          <goals>
            <goal>build</goal>
            <goal>generate-code</goal>
            <goal>generate-code-tests</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${compiler-plugin.version}</version>
      <configuration>
        <compilerArgs>
          <arg>-parameters</arg>
        </compilerArgs>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <configuration>
```

```
            <systemPropertyVariables>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager
>
                <maven.home>${maven.home}</maven.home>
            </systemPropertyVariables>
        </configuration>
      </plugin>
    </plugins>
</build>
```

注意

maven-surefire-plugin 运行应用程序的单元测试。

5. **可选：要构建原生应用程序，请添加包括 Maven Failsafe 插件 的特定原生配置集：**

```
<build>
  <plugins>
    ...
    <plugin>
      <artifactId>maven-failsafe-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>integration-test</goal>
            <goal>verify</goal>
          </goals>
          <configuration>
            <systemPropertyVariables>

<native.image.path>${project.build.directory}/${project.build.finalName}-runner
                </native.image.path>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager
>
                <maven.home>${maven.home}</maven.home>
            </systemPropertyVariables>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
...
<profiles>
  <profile>
    <id>native</id>
    <activation>
      <property>
```

```
            <name>native</name>
          </property>
        </activation>
        <properties>
          <skipITs>false</skipITs>
          <quarkus.package.type>native</quarkus.package.type>
        </properties>
      </profile>
    </profiles>
```

- 在其名称中包含 IT 的测试，并包含 @QuarkusIntegrationTest 注释，针对原生可执行文件运行。

- 有关原生模式与 JVM 模式的详情，请参阅 Quarkus "Getting Started" 指南中的 Difference between JVM 和 native mode。

1.5. 使用 CODE.QUARKUS.REDHAT.COM 创建 GETTING STARTED 项目

作为应用程序开发人员，您可以使用 code.quarkus.redhat.com 生成 Quarkus Maven 项目，并自动添加并配置要在应用程序中使用的扩展。另外，code.quarkus.redhat.com 会自动管理将项目编译为原生可执行文件所需的配置参数。

您可以生成 Quarkus Maven 项目，包括以下活动：

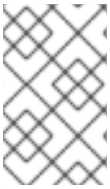
- 指定应用程序的基本详情
- 选择您要包含在项目中的扩展
- 使用项目文件生成可下载归档
- 使用自定义命令编译和启动应用程序

先决条件

- 您有一个 Web 浏览器。

- 您已准备了环境以使用 Apache Maven。如需更多信息，请参阅准备您的环境。
- 您已配置了 Quarkus Maven 存储库。要使用 Maven 创建 Quarkus 应用程序，请使用 Red Hat-hosted Quarkus 存储库。如需更多信息，请参阅为在线存储库配置 Maven settings.xml 文件。
- 可选 ： 已安装 Quarkus 命令行界面(CLI)，这是您可以在 dev 模式中启动 Quarkus 的方法之一。

  如需更多信息 ， 请参阅安装 Quarkus CLI。

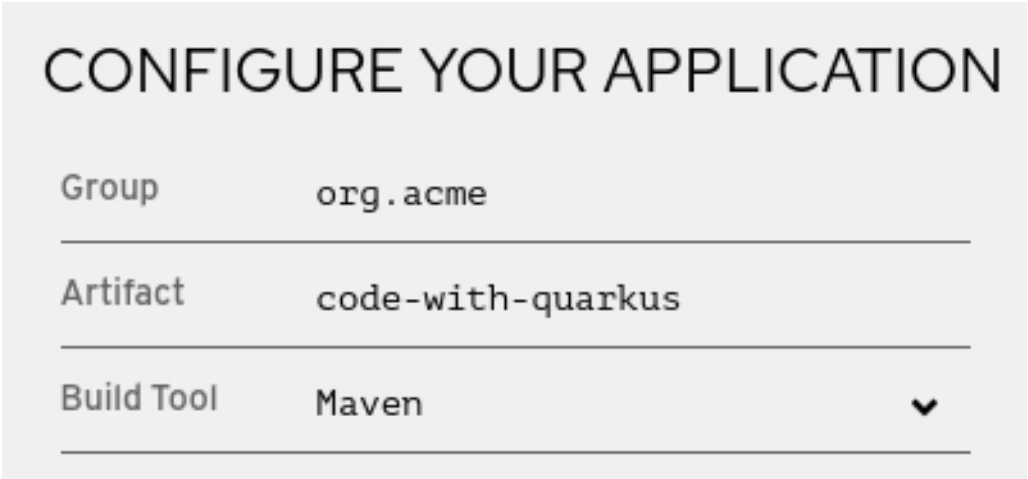


注意

Quarkus CLI 仅用于 dev 模式。红帽不支持在生产环境中使用 Quarkus CLI。

流程

1. 在您的 Web 浏览器中，访问 https://code.quarkus.redhat.com。
2. 指定项目的基本详情：



   a. 输入项目的组名称。name 格式遵循 Java 软件包命名规则，例如 org.acme。

   b. 输入项目生成的 Maven 工件的名称，如 code-with-quarkus。

c.

选择您要用来编译和启动应用程序的构建工具。您选择的构建工具决定了以下设置：

- 生成的项目的目录结构
- 您生成的项目中使用的配置文件格式
- 在生成项目后，会显示用于编译和启动 code.quarkus.redhat.com 的自定义构建脚本和命令。

注意

红帽仅支持使用 code.quarkus.redhat.com 创建 Quarkus Maven 项目。

3.

指定应用程序项目的更多详情：

a.

要显示包含更多应用程序详情的字段，请选择 ***More options***。

b.

输入您要用于项目生成的工件的版本。此字段的默认值为 **1.0.0-SNAPSHOT**。使用 语义版本 是首选的；但是，您可以选择指定不同类型的版本控制。

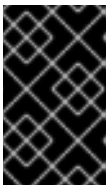
c.

选择是否希望 code.quarkus.redhat.com 将初学者代码添加到项目中。当您添加标记为 "***STARTER-CODE***" 的扩展时，您可以启用此选项在生成项目时自动为这些扩展创建示例类和资源文件。但是，如果您没有添加提供示例代码的任何扩展，这个选项不会影响生成的项目。

注意

code.quarkus.redhat.com 应用程序自动使用最新版本的 Red Hat build of Quarkus。但是，如果需要，可以在生成项目后手动更改为 pom.xml 文件中的较早 BOM 版本，但不建议这样做。

4. 选择要使用的扩展。Quarkus 应用程序包含您选择为依赖项的扩展。Quarkus 平台还确保这些扩展与将来的版本兼容。



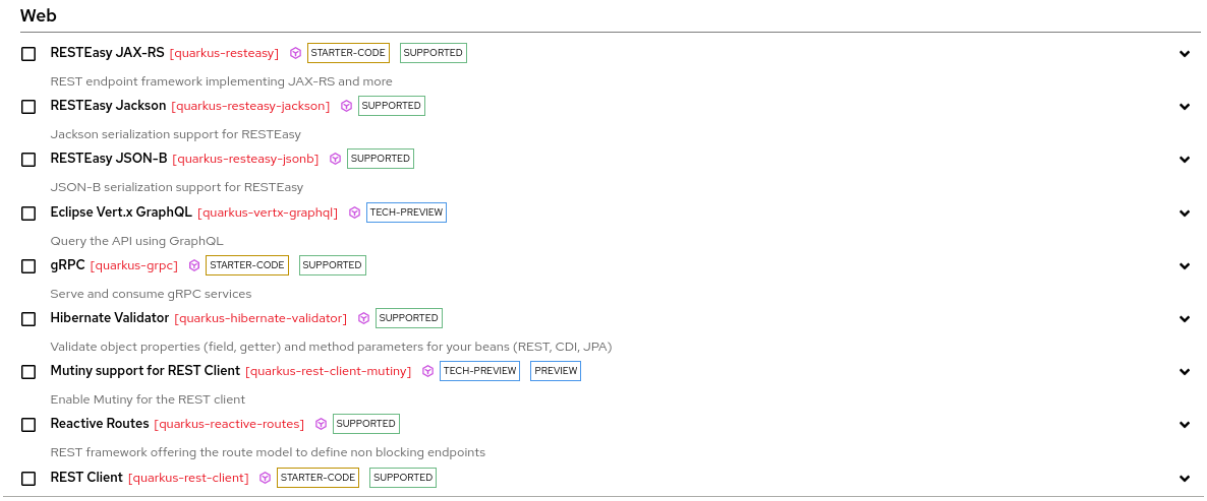
重要

不要在同一项目中使用 RESTEasy 和 RESTEasy 主动扩展。

扩展旁边的 quark 图标(

)表示扩展是 Red Hat build of Quarkus 平台发行版本的一部分。红帽更喜欢使用同一平台中的扩展，因为它们被一起测试和验证，因此更易于使用和升级。

您可以启用选项，为标记为"***STARTER-CODE***"的扩展自动生成初学程序代码。



5. **要确认您的选择，请选择 Generate your application。显示的对话框显示以下项目：**

- 下载包含您生成的项目的存档的链接
- 可用于编译和启动应用程序的命令

6. 要将带有生成的项目文件的存档保存到机器中，请选择 **Download the ZIP**。
7. 提取存档的内容。
8. 进入包含您提取的项目文件的目录：

```
cd <directory_name>
```

9. 要在 dev 模式中编译并启动应用程序，请使用以下方法之一：
   - 使用 **Maven**：

```
./mvnw quarkus:dev
```

   - 使用 **Quarkus CLI**：

```
quarkus dev
```

其他资源

- "Red Hat build of Quarkus"指南中的红帽构建的 Quarkus 扩展支持级别。

1.6. 配置 JAVA 编译器

默认情况下，Quarkus Maven 插件将编译器标志传递给来自 maven-compiler-plugin 的 javac 命令。

流程

- 要自定义开发模式中使用的编译器标志，请将配置部分添加到 插件 块中，并设置 编译器 Args 属性。您还可以设置 源、目标 和 jvmArgs。例如，要将 -verbose 传给 JVM 和 javac 命

令，请添加以下配置：

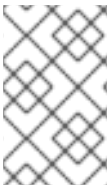
```
<plugin>
  <groupId>com.redhat.quarkus.platform</groupId>
  <artifactId>quarkus-maven-plugin</artifactId>
  <version>${quarkus.platform.version}</version>

  <configuration>
    <source>${maven.compiler.source}</source>
    <target>${maven.compiler.target}</target>
    <compilerArgs>
      <arg>-verbose</arg>
    </compilerArgs>
    <jvmArgs>-verbose</jvmArgs>
  </configuration>

  ...
</plugin>
```

1.7. 安装和管理扩展

在红帽构建的 Quarkus 中，您可以使用扩展来扩展应用程序的功能，并将框架配置、引导并将框架集成到应用程序中。此流程演示了如何查找并添加 Quarkus 项目的扩展。

先决条件

- 您已创建了 Quarkus Maven 项目。
- 已安装 Quarkus 命令行界面(CLI)，这是可用于管理 Quarkus 扩展的方法之一。如需更多信息，请参阅安装 Quarkus CLI。



注意

Quarkus CLI 仅用于 dev 模式。红帽不支持在生产环境中使用 Quarkus CLI。

流程

1. 进入您的 Quarkus 项目目录。
2. 使用以下方法之一列出所有可用扩展：

- 使用 Maven：

```
./mvnw quarkus:list-extensions
```

- 使用 Quarkus CLI：

```
quarkus extension --installable
```

3. 使用以下方法之一为项目添加扩展：

- 使用 Maven，输入以下命令，其中 <extension> 是您要添加的扩展的组、工件和版本 (GAV)：

```
./mvnw quarkus:add-extension -Dextensions="<extension>"
```

例如，要添加 Agroal 扩展，请输入以下命令：

```
./mvnw quarkus:add-extension -Dextensions="io.quarkus:quarkus-agroal"
```

- 使用 Quarkus CLI，输入以下命令，其中 <extension> 是您要添加的扩展的组、工件和版本(GAV)：

```
quarkus extension add '<extension>'
```

4. 要搜索特定扩展，请在 -Dextensions= 之后输入扩展名称或部分名称。以下示例搜索名称中包含文本的扩展：

```
./mvnw quarkus:add-extension -Dextensions=agroal
```

这个命令返回以下结果：

```
[SUCCESS]    Extension io.quarkus:quarkus-agroal has been installed
```

同样，使用 Quarkus CLI，您可以输入：

```
quarkus extension add 'agroal'
```

1.8. 将项目导入到 IDE 中

虽然您可以在文本编辑器中开发红帽构建的 Quarkus 项目，但您可能会更轻松地使用集成开发环境(IDE)。以下说明演示了如何将项目导入到特定的 IDE 中。

先决条件

- 您有一个 Quarkus Maven 项目。
- 已安装 Quarkus 命令行界面(CLI)，这是以 dev 模式启动项目所必需的。如需更多信息，请参阅安装 Quarkus CLI。

注意

Quarkus CLI 仅用于 dev 模式。红帽不支持在生产环境中使用 Quarkus CLI。

流程

为您的 IDE 完成所需步骤。

CodeReady Studio 或 Eclipse

1. 在 CodeReady Studio 或 Eclipse 中，点 File>*Import*。
2. 选择 Maven → Existing Maven Project。
3. 接下来，选择项目的 root 位置。此时会出现可用模块的列表。
4. 选择生成的项目，然后单击 Finish。
5. 要编译并启动应用程序，请使用以下方法之一：

- **使用 Maven：**

```
./mvnw quarkus:dev
```

- **使用 Quarkus CLI：**

```
quarkus dev
```

IntelliJ

1. **在 IntelliJ 中，完成以下任务之一：**
   - 选择 **File > New &gt ; Project from Existing Sources。**
   - **在 Welcome 页面上，选择 Import project。**
2. **选择项目根目录。**
3. **选择 Import project from external model，然后选择 Maven。**
4. **检查选项，然后单击 Next。**
5. **点 Create。**
6. **要编译并启动应用程序，请使用以下方法之一：**
   - **使用 Maven：**

```
./mvnw quarkus:dev
```

   - **使用 Quarkus CLI：**
   -

```
quarkus dev
```

Apache NetBeans

1. 选择 **File > Open Project**。
2. 选择 项目根目录。
3. 单击 **Open Project**。
4. 要编译并启动应用程序，请使用以下方法之一：
   - 使用 **Maven**：

```
./mvnw quarkus:dev
```

   - 使用 **Quarkus CLI**：

```
quarkus dev
```

Visual Studio Code

1. 安装 **Java** 扩展包.
2. 在 **Visual Studio Code** 中，打开您的项目目录。

验证

项目作为 **Maven** 项目加载。

1.9. 配置红帽构建的 QUARKUS 项目输出

在构建应用程序前，您可以通过更改 **application.properties** 文件中属性的默认值来控制构建命令输出。

先决条件

- 您已创建了 Quarkus Maven 项目。

流程

1. 进入 {project}/src/main/resources 文件夹，并在文本编辑器中打开 application.properties 文件。

2. 编辑您要更改的属性值并保存文件。

下表列出了您可以更改的属性：

属性	描述	类型	default
quarkus.package.main-class	应用程序的入口点。在大多数情况下，您必须更改这个值。	string	**io.quarkus.runner.GeneratedMain**
quarkus.package.type	软件包请求的输出类型，您可以将其设置为 'jar'（将 'fast-jar'）、'legacy-jar' 用于 pre-1.12 默认 jar 打包、'uber-jar'、'native' 或 'native-sources'。	string	**jar**
quarkus.package.manifest.add-implementation-entries	确定实施信息是否必须包含在运行程序 JAR 文件的 **MANIFEST.MF** 文件中。	布尔值	**true**
quarkus.package.user-configured-ignored-entries	不得复制到输出工件中的文件。	字符串（列表）	
quarkus.package.runner-suffix	应用到运行程序 JAR 文件的后缀。	string	**-runner**
quarkus.package.output-directory	应用构建的输出文件夹。这相对于构建系统目标目录解析。	string	
quarkus.package.output-name	最终工件的名称。	string	

1.10. 使用自定义配置集在 JVM 模式中测试红帽构建的 QUARKUS 应用程序

与任何其他正在运行的模式类似，测试的配置值是从 src/main/resources/application.properties 文件中读取的。

默认情况下，test 配置集在 JVM 模式下测试过程中处于活动状态，这意味着具有 %test 前缀的属性优先。例如，当使用以下配置运行测试时，属性 消息 返回的值为 Test Value。

```
message=Hello
%test.message=Test Value
```

如果 %test 配置集不活跃(dev、prod)，则属性 消息 返回的值是 Hello。

例如，您的应用可能需要多个测试配置集针对不同的数据库实例运行一组测试。要做到这一点，您必须覆盖测试配置集名称，这可通过在执行 Maven 时设置系统属性 quarkus.test.profile 来完成。这样，您可以控制在测试过程中活跃的哪些配置值集合。

如需了解更多有关使用 'Starting With Quarkus' 示例的标准测试的信息，请参阅 Getting Started 指南中的使用 JUnit 测试红帽构建的 Quarkus 应用程序。

先决条件

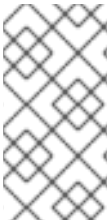
- 使用 Apache Maven 创建的 Quarkus 项目。

流程

在 Quarkus 应用上运行测试时，测试配置 配置集默认设置为 active。但是，您可以使用 quarkus.test.profile 系统属性将配置集改为自定义配置集。

1. 运行以下命令来测试应用程序：

```
mvn test -Dquarkus.test.profile=__<profile-name>__
```



注意

您不能在原生模式中使用自定义测试配置配置集。原生测试始终在 prod 配置文件下运行。

1.11. 记录红帽构建的 QUARKUS 应用程序构建类路径树

Quarkus 构建过程将应用程序中使用的扩展的部署依赖项添加到原始应用程序类路径中。您可以看到构建类路径中包含哪些依赖项和版本。quarkus-maven-plugin Maven 插件包含 dependency-tree 目标，它显示应用程序的构建依赖项树。

先决条件

- 您已创建了 Quarkus Maven 应用程序。

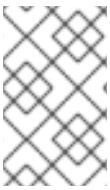
流程

- 要列出应用程序的构建依赖项树，请输入以下命令：

```
./mvnw quarkus:dependency-tree
```

示例输出。您看到的确切输出将与这个示例不同。

```
[INFO] └─ io.quarkus:quarkus-resteasy-deployment:jar:3.8.4.redhat-00002 (compile)
[INFO]    ├─ io.quarkus:quarkus-resteasy-server-common-deployment:jar:3.8.4.redhat-
00002 (compile)
[INFO]    │  ├─ io.quarkus:quarkus-resteasy-common-deployment:jar:3.8.4.redhat-
00002 (compile)
[INFO]    │  │  ├─ io.quarkus:quarkus-resteasy-common:jar:3.8.4.redhat-00002
(compile)
[INFO]    │  │  │  ├─ org.jboss.resteasy:resteasy-core:jar:6.2.4.Final-redhat-00003
(compile)
[INFO]    │  │  │  │  ├─ jakarta.xml.bind:jakarta.xml.bind-api:jar:4.0.0.redhat-00008
(compile)
[INFO]    │  │  │  │  ├─ org.jboss.resteasy:resteasy-core-spi:jar:6.2.4.Final-redhat-
00003 (compile)
[INFO]    │  │  │  │  ├─ org.reactivestreams:reactive-streams:jar:1.0.4.redhat-00003
(compile)
[INFO]    │  │  │  │  └─ com.ibm.async:asyncutil:jar:0.1.0.redhat-00010 (compile)
...
```



注意

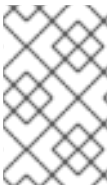
mvn dependency:tree 命令只显示应用程序的运行时依赖项。

1.12. 生成原生可执行文件

原生二进制文件是创建在特定操作系统和 CPU 架构上运行的可执行文件。

以下列表概述了原生可执行文件的一些示例：

- Linux AMD 64 位的 ELF 二进制文件
- Windows AMD 64 位的 EXE 二进制文件
- ARM 64 位的 ELF 二进制文件



注意

红帽构建的 Quarkus 仅支持 Linux AMD 64 位的 ELF 二进制文件。

构建原生可执行文件时，您的应用程序和依赖项（包括 JVM）被打包到一个文件中。应用程序的原生可执行文件包含以下项目：

- 编译的应用程序代码
- 所需的 Java 库
- 减少了虚拟机(VM)的版本，用于改进应用程序启动时间和最小磁盘和内存占用量，这也是为应用程序代码及其依赖项量身定制的

要从 Quarkus 应用生成原生可执行文件，您可以选择容器内构建或 local-host 构建。下表解释您可以使用的不同构建选项：

表 1.1. 构建生成原生可执行文件的选项

构建选项	Requires	使用	结果	优点
in-container build - Supported	容器运行时，如 Podman 或 Docker	默认 **registry.access.redhat.com/quarkus/mandrel-for-jdk-21-rhel8:23.1** 构建器镜像	使用主机的 CPU 架构进行 Linux 64 位可执行文件	GraalVM 不需要在本地设置，从而使 CI 管道 更有效地运行
local-host build - 仅支持上游	GraalVM 或 Mandrel 的本地安装	其本地安装作为 **quarkus.native.builder-image** 属性的默认设置	具有与执行构建的机器相同的操作系统和 CPU 架构的可执行文件	不允许或不想使用 Docker 或 Podman 等工具的开发人员的替代方案。总体而言，它比容器内构建方法更快。

重要

- **红帽构建的 Quarkus 3.8 仅支持使用基于 Java 21 的 红帽构建的 Quarkus 原生构建器镜像构建原生 Linux 可执行文件，该镜像是 Mandrel 的产品化分发。虽然其他镜像在社区中可用，但产品不支持它们，因此您不应该将其用于您希望红帽提供支持的生产构建。**
- **其源基于 17 （不使用 Java 18 - 21 的功能）编写的应用程序仍然可以使用基于 Java 21 的 Mandrel 23.1 基础镜像编译应用程序的原生可执行文件。**
- **使用红帽构建的 Quarkus 不支持使用 Oracle GraalVM 社区版(CE)、Mael 社区版本或任何其他 GraalVM 发行版构建原生可执行文件。**

1.12.1. 使用 in-container 构建生成原生可执行文件

要创建原生可执行文件并运行原生镜像测试，请使用由红帽构建的 Quarkus 提供的 原生 配置集进行容器内构建。

先决条件

- **podman 或 Docker 已安装。**
- **容器可以访问至少 8GB 内存。**

流程

1. **打开 Getting Started project pom.xml 文件，并验证项目是否包含 native 配置集：**

```
<profiles>
 <profile>
  <id>native</id>
  <activation>
   <property>
    <name>native</name>
   </property>
  </activation>
  <properties>
   <skipITs>false</skipITs>
   <quarkus.package.type>native</quarkus.package.type>
  </properties>
 </profile>
</profiles>
```

2. **使用以下方法之一构建原生可执行文件：**

- **使用 Maven：**
  - **对于 Docker：**

```
./mvnw package -Dnative -Dquarkus.native.container-build=true
```

  - **对于 Podman：**

```
./mvnw package -Dnative -Dquarkus.native.container-build=true -
Dquarkus.native.container-runtime=podman
```

- **使用 Quarkus CLI：**
  - **对于 Docker：**

```
quarkus build --native -Dquarkus.native.container-build=true
```

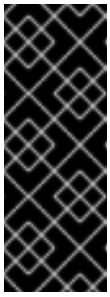
  - **对于 Podman：**

```
quarkus build --native -Dquarkus.native.container-build=true -
Dquarkus.native.container-runtime=podman
```

步骤结果

这些命令在目标目录中创建一个 *-runner 二进制文件，其中适用以下内容：

- *-runner 文件是由 Quarkus 生成的构建原生二进制文件。
- 目标目录 是一个目录，Maven 会在构建 Maven 应用程序时创建该目录。



重要

将 Quarkus 应用程序编译到原生可执行文件会在分析和优化过程中消耗大量内存。您可以通过设置 quarkus.native.native-image-xmx 配置属性来限制原生编译过程中使用的内存量。设置低内存限值可能会增加构建时间。

3. 要运行原生可执行文件，请输入以下命令：

```
./target/*-runner
```

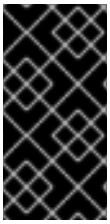
其他资源

- "将红帽构建的 Quarkus 应用程序构建到原生可执行文件"指南中的原生 可执行配置属性。

1.12.2. 使用 local-host 构建生成原生可执行文件

如果您不使用 Docker 或 Podman，请使用 Quarkus local-host build 选项来创建和运行原生可执行文件。

使用本地主机构建方法比使用容器更快，并适用于使用 Linux 操作系统的机器。



重要

红帽构建的 Quarkus 不支持在生产环境中使用以下步骤。只有在 Docker 或 Podman 不可用时，才使用这个方法测试或作为备份方法。

先决条件

- Mandrel 或 GraalVm 的本地安装，根据 构建原生可执行文件 指南进行了正确配置。
  - 另外，对于 GraalVM 安装，还必须安装 native-image。

流程

1. 对于 GraalVM 或 Mandrel，使用以下方法之一构建原生可执行文件：
   - 使用 Maven：

```
./mvnw package -Dnative
```

   - 使用 Quarkus CLI：

```
quarkus build --native
```

步骤结果

这些命令在目标目录中创建一个 *-runner 二进制文件，其中适用以下内容：

- *-runner 文件是 Quarkus 生成的内置原生二进制文件。
- 目标目录 是一个目录，Maven 会在构建 Maven 应用程序时创建该目录。

注意

构建原生可执行文件时，会启用 prod 配置集，除非在 quarkus.profile 属性中修改了。

2. 运行原生可执行文件：

```
./target/*-runner
```

其他资源

如需更多信息，请参阅 Quarkus "Building a native executable" 指南中的 Producing a native executable 部分。

1.12.3. 手动创建容器

您可以使用应用程序为 Linux AMD64 手动创建容器镜像。当您使用 Quarkus Native 容器生成原生镜像时，原生镜像会创建一个以 Linux AMD64 为目标的可执行文件。如果您的主机操作系统与 Linux AMD64 不同，则无法直接运行二进制文件，您需要手动创建容器。

您的 Quarkus Getting Started 项目在 src/main/docker 目录中包含一个 Dockerfile.native，其内容如下：

```
FROM registry.access.redhat.com/ubi8/ubi-minimal:8.9
WORKDIR /work/
RUN chown 1001 /work \
    && chmod "g+rwX" /work \
    && chown 1001:root /work
COPY --chown=1001:root target/*-runner /work/application

EXPOSE 8080
USER 1001

ENTRYPOINT ["./application", "-Dquarkus.http.host=0.0.0.0"]
```

注意

通用基础镜像(UBI)

以下列表显示了可用于 Dockerfile 的合适镜像。

- **Red Hat Universal Base Image 8 (UBI8)。此基础镜像旨在设计并设计成为所有容器化应用程序、中间件和实用程序的基础层。**

```
registry.access.redhat.com/ubi8/ubi:8.9
```

- **Red Hat Universal Base Image 8 Minimal (UBI8-minimal)。使用 microdnf 作为软件包管理器的精简版 UBI8 镜像。**

```
registry.access.redhat.com/ubi8/ubi-minimal:8.9
```

- **所有红帽基础镜像都可在容器镜像目录站点中找到。https://catalog.redhat.com/software/containers/search?q=UBI&p=1**

流程

1. **使用以下方法之一构建原生 Linux 可执行文件：**

- **docker:**

```
./mvnw package -Dnative -Dquarkus.native.container-build=true
```

- **Podman:**

```
./mvnw package -Dnative -Dquarkus.native.container-build=true -
Dquarkus.native.container-runtime=podman
```

2. **使用以下方法之一构建容器镜像：**

- **docker:**

```
docker build -f src/main/docker/Dockerfile.native -t quarkus-quickstart/getting-started
```

- Podman

```
podman build -f src/main/docker/Dockerfile.native -t quarkus-quickstart/getting-started
```

3. 使用以下方法之一运行容器：

- docker:

```
docker run -i --rm -p 8080:8080 quarkus-quickstart/getting-started
```

- Podman:

```
podman run -i --rm -p 8080:8080 quarkus-quickstart/getting-started
```

1.13. 测试原生可执行文件

以原生模式测试应用，以测试原生可执行文件的功能。使用 @QuarkusIntegrationTest 注释来构建原生可执行文件，并根据 HTTP 端点运行测试。

重要

以下示例演示了如何使用本地安装 GraalVM 或 Mandrel 测试原生可执行文件。开始之前，请考虑以下点：

- Red Hat build of Quarkus 不支持这种情况，如 Producing a native executable 所述。
- 您在此处测试的原生可执行文件必须与主机的操作系统和架构匹配。因此，这个过程不适用于 macOS 或 in-container 构建。

流程

1. 打开 pom.xml 文件，并验证 build 部分是否具有以下元素：

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-failsafe-plugin</artifactId>
  <version>${surefire-plugin.version}</version>
  <executions>
    <execution>
      <goals>
        <goal>integration-test</goal>
        <goal>verify</goal>
      </goals>
      <configuration>
        <systemPropertyVariables>
          <native.image.path>${project.build.directory}/${project.build.finalName}-runner</native.image.path>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
          <maven.home>${maven.home}</maven.home>
        </systemPropertyVariables>
      </configuration>
    </execution>
  </executions>
</plugin>
```

- Maven Failsafe 插件(maven-failsafe-plugin)运行集成测试，并指示生成的原生可执行文件的位置。

2. 打开 src/test/java/org/acme/GreetingResourceIT.java 文件，并验证该文件是否包含以下内容：

```
package org.acme;

import io.quarkus.test.junit.QuarkusIntegrationTest;

@QuarkusIntegrationTest (1)
public class GreetingResourceIT extends GreetingResourceTest { (2)

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

(1) 在测试之前，使用另一个测试运行程序从原生文件启动应用。可执行文件通过使用 Maven Failsafe 插件中配置的 native.image.path 系统属性来检索。

(2)

本例扩展了 GreetingResourceTest，但您也可以创建新的测试。

3. 运行测试：

```
./mvnw verify -Dnative
```

以下示例显示了这个命令的输出：

```
./mvnw verify -Dnative
....

GraalVM Native Image: Generating 'getting-started-1.0.0-SNAPSHOT-runner'
(executable)...
========================================================================
========================================
[1/8] Initializing...                                                (6.6s @ 0.22GB)
 Java version: 17.0.7+7, vendor version: Mandrel-23.1.0.0-Final
 Graal compiler: optimization level: 2, target machine: x86-64-v3
 C compiler: gcc (redhat, x86_64, 13.2.1)
 Garbage collector: Serial GC (max heap size: 80% of RAM)
 2 user-specific feature(s)
 - io.quarkus.runner.Feature: Auto-generated class by {ProductLongName} from the
existing extensions
 - io.quarkus.runtime.graal.DisableLoggingFeature: Disables INFO logging during the
analysis phase
[2/8] Performing analysis...  [******]                                (40.0s @
2.05GB)
  10,318 (86.40%) of 11,942 types reachable
  15,064 (57.36%) of 26,260 fields reachable
  52,128 (55.75%) of 93,501 methods reachable
   3,298 types,   109 fields, and 2,698 methods registered for reflection
      63 types,    68 fields, and    55 methods registered for JNI access
       4 native libraries: dl, pthread, rt, z
[3/8] Building universe...                                           (5.9s @ 1.31GB)
[4/8] Parsing methods...      [**]                                    (3.7s @
2.08GB)
[5/8] Inlining methods...     [***]                                   (2.0s @ 1.92GB)
[6/8] Compiling methods...    [******]                                (34.4s @
3.25GB)
[7/8] Layouting methods...    [[7/8] Layouting methods...    [**]
(4.1s @ 1.78GB)
[8/8] Creating image...       [**]                                    (4.5s @ 2.31GB)
  20.93MB (48.43%) for code area:    33,233 compilation units
  21.95MB (50.80%) for image heap:  285,664 objects and 8 resources
 337.06kB ( 0.76%) for other data
  43.20MB in total

....
```

```
[INFO]
[INFO] --- maven-failsafe-plugin:3.0.0-M7:integration-test (default) @ getting-started ---
[INFO] Using auto detected provider
org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running org.acme.GreetingResourceIT
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2024-02-21 14:04:52,681 INFO  [io.quarkus] (main) getting-started 1.0.0-SNAPSHOT
native (powered by Red Hat build of Quarkus 3.8.0.Final) started in 0.038s. Listening
on: http://0.0.0.0:8081
2024-02-21 14:04:52,682 INFO  [io.quarkus] (main) Profile prod activated.
2024-02-21 14:04:52,682 INFO  [io.quarkus] (main) Installed features: [cdi, resteasy-
reactive, smallrye-context-propagation, vertx]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.696 s - in
org.acme.GreetingResourceIT
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-failsafe-plugin:3.0.0-M7:verify (default) @ getting-started ---
```

注意

Quarkus 在自动失败原生测试前等待 60 秒启动原生镜像。您可以通过配置 quarkus.test.wait-time 系统属性来更改此持续时间。

您可以使用以下命令扩展等待时间，其中 < *duration*> 是等待时间（以秒为单位）：

```
./mvnw verify -Dnative -Dquarkus.test.wait-time=<duration>
```

注意

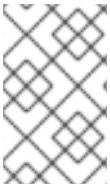
- 默认情况下，使用 prod 配置集运行原生测试，除非在 quarkus.test.native-image-profile 属性中进行了修改。

1.14. 使用红帽构建的 QUARKUS 开发模式

开发模式使用后台编译启用热部署，这意味着当您修改 Java 或资源文件时，刷新浏览器会自动生效。这也适用于配置属性文件等资源文件。您可以使用 Maven 或 Quarkus 命令行界面(CLI)在开发模式中启动 Quarkus。

先决条件

- 您已创建了 Quarkus Maven 应用程序。
- 已安装 Quarkus CLI，这是您可以在开发模式下启动 Quarkus 的方法之一。如需更多信息，请参阅安装 Quarkus CLI。



注意

Quarkus CLI 仅用于 dev 模式。红帽不支持在生产环境中使用 Quarkus CLI。

流程

1. 切换到包含 Quarkus 应用 pom.xml 文件的目录。
2. 要以开发模式编译并启动 Quarkus 应用程序，请使用以下方法之一：
   - 使用 Maven：

```
./mvnw quarkus:dev
```

   - 使用 Quarkus CLI：

```
quarkus dev
```

3. 更改您的应用程序并保存文件。
4. 刷新浏览器，以触发工作区扫描。

如果检测到任何更改，则会重新编译 Java 文件，并重新部署应用程序。然后，重新部署的应用程序会为您的请求提供服务。如果编译或部署存在问题，则会出现错误页面。

在开发模式中，调试器已被激活，并侦听端口 5005。

5. 可选： 要在运行应用前等待调试器附加，请包括 -Dsuspend ：

```
./mvnw quarkus:dev -Dsuspend
```

6. 可选： 要防止调试器运行，请包含 -Ddebug=false ：

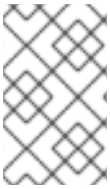
```
./mvnw quarkus:dev -Ddebug=false
```

1.15. 调试红帽构建的 QUARKUS 项目

当红帽构建的 Quarkus 以开发模式启动时，会默认启用调试，调试器会监听端口 5005，而不挂起 JVM。您可以从命令行或配置系统属性来启用和配置 Quarkus 的调试功能。您还可以使用 Quarkus CLI 调试项目。

先决条件

- 您已创建了红帽构建的 Quarkus Maven 项目。
- 已安装 Quarkus 命令行界面(CLI)，这是可用于编译和调试项目的方法之一。如需更多信息，请参阅安装 Quarkus CLI。



注意

Quarkus CLI 仅用于 dev 模式。红帽不支持在生产环境中使用 Quarkus CLI。

流程

使用以下方法之一控制调试：

通过配置系统属性来控制调试器

1. 更改 debug 系统属性的以下值，其中 *PORT* 是调试器侦听的端口：

- 错误 ：JVM 在禁用调试模式时启动。
- true ：JVM 以调试模式启动，并侦听端口 5005。
- 客户端 ：JVM 在客户端模式中启动，并尝试连接到 localhost:5005。
- *PORT*: JVM 以调试模式启动并侦听 *PORT*。

2. 要在以 debug 模式运行时挂起 JVM，请将 suspend 系统属性的值设置为以下值之一：
   - y 或 true ：调试模式 JVM 启动挂起。
   - n 或 false ：调试模式 JVM 启动而不暂停。

从命令行控制调试器

- 要使用暂停的 JVM 以调试模式编译并启动 Quarkus 应用程序，请使用以下方法之一：
  - 使用 Maven：

```
./mvnw quarkus:dev -Dsuspend
```

  - 使用 Quarkus CLI：

```
quarkus dev --suspend
```

为特定主机网络接口启用调试器

在开发模式中，为了安全起见，Quarkus 将 debug 主机接口设置为 localhost。

要为特定主机网络接口启用调试器，您可以使用以下方法之一使用 -DdebugHost 选项：

- **使用 Maven：**

```
./mvnw quarkus:dev -DdebugHost=<host-ip-address>
```

- **使用 Quarkus CLI：**

```
quarkus dev --debug-host=<host-ip-address>
```

其中 < host-ip-address > 是您要启用调试的主机网络接口的 IP 地址。

注意

要在所有主机接口上启用调试，请将 < host-ip-address& gt; 替换为以下值：

```
0.0.0.0
```

1.16. 其他资源

- 红帽构建的 Quarkus 入门
- Apache Maven 项目

更新于 2024-05-10