Red Hat build of Eclipse Vert.x 4.3

Eclipse Vert.x 入门

用于 Eclipse Vert.x 4.3.7

Last Updated: 2023-04-07

Red Hat build of Eclipse Vert.x 4.3 Eclipse Vert.x 入门

用于 Eclipse Vert.x 4.3.7

法律通告

摘要

本指南论述了如何使用 Apache Maven 创建简单的 Eclipse Vert.x 应用程序。

目录

第 1 章 开始的先决条件

本指南涵盖了概念以及开发人员使用 Eclipse Vert.x 运行时所需要的实际详细信息。

作为应用开发人员，您可以使用 Eclipse Vert.x 创建以 Java 编写在 OpenShift 环境中的基于微服务的应用程序。

本指南介绍了如何创建、打包、运行和测试简单的 Eclipse Vert.x 项目。

前提条件

- OpenJDK 8 或 OpenJDK 11 已安装，**JAVA_HOME** 环境变量指定 Java SDK 的位置。登录红帽客户门户，从 软件下载 下载红帽构建的 Open JDK。
- 已安装 Apache Maven 3.6.0 或更高版本。您可以从 Apache Maven Project 网站下载 Maven。

第 2 章 ECLIPSE VERT.X 概述

Eclipse Vert.x 是用于创建被动、非阻塞和在 Java 虚拟机(JVM)上运行的异步应用程序的工具箱。

Eclipse Vert.x 旨在成为云原生。它允许应用程序使用非常少的线程。这可避免在创建新线程时导致的开销。这可让 Eclipse Vert.x 应用和服务有效使用其内存以及云环境中的 CPU 配额。

在 OpenShift 中使用 Eclipse Vert.x 运行时，可以更轻松地构建被动系统。OpenShift 平台的功能也可用，如滚动更新、服务发现和 canary 部署。借助 OpenShift，您可以在您的应用中实施外部化配置、健康检查、断路器和故障转移等微服务模式。

2.1. ECLIPSE VERT.X 的主要概念

本节介绍了与 Eclipse Vert.x 运行时关联的一些关键概念。它还简要概述被动系统。

云和容器应用程序

云原生应用通常利用微服务构建。它们设计为形成分离组件的分布式系统。这些组件通常在包含大量节点的集群之上在容器内运行。这些应用程序预计会在不需要任何服务停机时间的情况下更新各个组件的故障。基于云原生应用的系统依赖于由底层云平台（如 OpenShift）提供的自动化部署、扩展和管理和维护任务。使用现成的管理和编排工具在集群级别上执行管理和配置任务，而不是在各个机器的水平执行。

被动系统

被动系统(Reactive manifesto 中定义)是一个分布式系统，它具有以下特征：

Elastic

系统在不同的工作负载下保持响应，各个组件根据需要进行扩展和负载均衡，以适应工作负载的不同部分。弹性应用程序无论它们同时收到的请求数量如何，都可提供相同的服务质量。

弹性

即使有任何独立组件失败，系统也会保持响应。在系统中，组件相互隔离。这有助于在失败时快速恢复各个组件。单个组件故障不应影响其他组件的功能。这可防止级联失败，而隔离组件的失败导致其他组件被阻断，并逐渐失败。

响应

快速响应的系统设计为始终以合理的时间内响应请求，以确保提供一致的服务质量。为了保持响应响应，应用程序间的通信通道绝对不会被阻止。

message-Driven

应用的各个组件使用异步消息传递来相互通信。如果事件发生（如鼠标点击或服务上的搜索查询），服务会发送通用频道的消息（即事件总线）。消息反过来由相应的组件发现并由相应的组件处理。

被动系统是分布式系统。它们设计为可将其异步属性用于应用程序开发。

被动编程

被动系统的概念描述了分布式系统的架构，但被动编程是指使应用程序在代码级别重新活跃的做法。被动编程是一种开发模型，用于编写异步和事件驱动的应用程序。在被动应用程序中，代码对事件或消息做出反应。

被动编程有多种实现。例如，使用回调进行简单的实施，使用 Reactive Extensions (Rx)和 coroutines 的复杂实现。

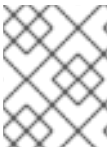
Reactive Extensions (Rx)是 Java 中最成熟的被动编程形式之一。它使用 *RxJava* 库。

第 3 章 使用 POM 文件创建 ECLIPSE VERT.X 项目

当您开发基本的 Eclipse Vert.x 应用程序时，您应该创建以下工件。我们将在我们首先 **启动的** Eclipse Vert.x 项目中创建这些工件。

- 包含 Eclipse Vert.x 方法的 Java 类。
- 包含 Maven 构建应用程序所需的信息的 **pom.xml** 文件。

以下流程创建了一个简单的 **Greeting** 应用程序，它返回 **Greetings!** 作为响应。



注意

Eclipse Vert.x 支持基于 OpenJDK 8 和 OpenJDK 11 的构建器镜像，用于构建和部署应用程序到 OpenShift。不支持 Oracle JDK 和 OpenJDK 9 构建器镜像。

前提条件

- 安装了 OpenJDK 8 或 OpenJDK 11。
- 已安装 Maven。

流程

1. 创建新目录 **getting-started** 并浏览到其中。

```
$ mkdir getting-started
$ cd getting-started
```

这是应用的根目录。

2. 在根目录中创建目录结构 **src/main/java/com/example/**，然后导航到.

```
$ mkdir -p src/main/java/com/example/
$ cd src/main/java/com/example/
```

3. 创建包含应用代码的 Java 类文件 **MyApp.java**。

```
package com.example;

import io.vertx.core.AbstractVerticle;
import io.vertx.core.Promise;

public class MyApp extends AbstractVerticle {

  @Override
  public void start(Promise<Void> promise) {
    vertx
      .createHttpServer()
      .requestHandler(r ->
        r.response().end("Greetings!"))
      .listen(8080, result -> {
        if (result.succeeded()) {
          promise.complete();
```

```
            } else {
              promise.fail(result.cause());
            }
          });
    }
}
```

该应用在端口 8080 上启动 HTTP 服务器。向您发送不动时，它又会得到 **回报！** 我很好。

4. 在应用程序根目录中创建一个 **pom.xml** 文件，其内容如下：

- **在 & lt;dependencyManagement& gt; 部分中，添加 io.vertx:vertx-dependencies 构件。**

- **将 类型指定为 pom，范围 指定为 导入。**

- **在 & lt;project& gt; 部分的 <properties > 下，指定 Eclipse Vert.x 和 Eclipse Vert.x Maven 插件的版本。**

注意

属性可用于设置在每个版本中更改的值。例如，产品或插件的版本。

- **在 &lt ;project> 部分中，在 & lt;plugin& gt; 下，指定 vertx-maven-plugin。Eclipse Vert.x Maven 插件用于打包您的应用程序。**

- **包含 软件仓库 和插件，以指定 包含要构建应用程序的工件和插件的存储库。**

pom.xml 包含以下工件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.example</groupId>
 <artifactId>my-app</artifactId>
 <version>1.0.0-SNAPSHOT</version>
 <packaging>jar</packaging>
```

```
<name>My Application</name>
<description>Example application using Vert.x</description>

<properties>
  <vertx.version>4.3.7.redhat-00002</vertx.version>
  <vertx-maven-plugin.version>1.0.24</vertx-maven-plugin.version>
  <vertx.verticle>com.example.MyApp</vertx.verticle>

  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
</properties>

<!-- Import dependencies from the Vert.x BOM. -->
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.vertx</groupId>
      <artifactId>vertx-dependencies</artifactId>
      <version>${vertx.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<!-- Specify the Vert.x artifacts that your application depends on. -->
<dependencies>
  <dependency>
    <groupId>io.vertx</groupId>
    <artifactId>vertx-core</artifactId>
  </dependency>
  <dependency>
    <groupId>io.vertx</groupId>
    <artifactId>vertx-web</artifactId>
  </dependency>

  <!-- Test dependencies -->
  <dependency>
    <groupId>io.vertx</groupId>
    <artifactId>vertx-junit5</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.4.0</version>
    <scope>test</scope>
  </dependency>

</dependencies>

<!-- Specify the repositories containing Vert.x artifacts. -->
<repositories>
```

```
    <repository>
      <id>redhat-ga</id>
      <name>Red Hat GA Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
    </repository>
  </repositories>

<!-- Specify the version of the Maven Surefire plugin. -->
    <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>3.0.0-M5</version>
      </plugin>
      <plugin>

<!-- Configure your application to be packaged using the Vert.x Maven Plugin. -->
        <groupId>io.reactiverse</groupId>
        <artifactId>vertx-maven-plugin</artifactId>
        <version>${vertx-maven-plugin.version}</version>
        <executions>
          <execution>
            <id>vmp</id>
            <goals>
              <goal>initialize</goal>
              <goal>package</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

5. 从应用的根目录使用 Maven 构建应用。

```
mvn vertx:run
```

6. 验证应用是否正在运行。

使用 curl 或浏览器验证您的应用程序是否在 http://localhost:8080 中运行，并返回 "Greetings!" 作为回答。

```
$ curl http://localhost:8080
Greetings!
```

第 4 章 使用 JUNIT 测试 ECLIPSE VERT.X 应用程序

在 getting-started 项目中构建了 Eclipse Vert.x 应用后，使用 JUnit 5 框架测试您的应用程序以确保它按预期运行。Eclipse Vert.x pom.xml 文件中的以下两个依赖项用于 JUnit 5 测试：

```
<dependency>
    <groupId>io.vertx</groupId>
    <artifactId>vertx-junit5</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.4.0</version>
    <scope>test</scope>
  </dependency>
```

- 测试需要 vertx-junit5 依赖项。JUnit 5 提供了各种注释，如 @Test、@Before 每个、@DisplayName 等等，它们可用于请求对 Vertx 和 VertxTestContext 实例进行异步注入。

- 运行时执行测试需要 junit-jupiter-engine 依赖项。

前提条件

- 您已使用 pom.xml 文件构建了 Eclipse Vert.x getting-started 项目。

流程

1. 打开生成的 pom.xml 文件并设置 Surefire Maven 插件的版本：

```
<plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>3.0.0-M5</version>
</plugin>
```

2. 在根目录中创建目录结构 src/test/java/com/example/，然后导航到.

```
$ mkdir -p src/test/java/com/example/
$ cd src/test/java/com/example/
```

3. 创建包含应用代码的 Java 类文件 MyTestApp.java。

```
package com.example;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;

import io.vertx.core.Vertx;
import io.vertx.core.http.HttpMethod;
import io.vertx.junit5.VertxExtension;
import io.vertx.junit5.VertxTestContext;

@ExtendWith(VertxExtension.class)
class MyAppTest {

 @BeforeEach
 void prepare(Vertx vertx, VertxTestContext testContext) {
  // Deploy the verticle
  vertx.deployVerticle(new MyApp())
    .onSuccess(ok -> testContext.completeNow())
    .onFailure(failure -> testContext.failNow(failure));
 }

 @Test
 @DisplayName("Smoke test: check that the HTTP server responds")
 void smokeTest(Vertx vertx, VertxTestContext testContext) {
  // Issue an HTTP request
  vertx.createHttpClient()
    .request(HttpMethod.GET, 8080, "127.0.0.1", "/")
    .compose(request -> request.send())
    .compose(response -> response.body())
    .onSuccess(body -> testContext.verify(() -> {
      // Check the response
      assertEquals("Greetings!", body.toString());
      testContext.completeNow();
    }))
    .onFailure(failure -> testContext.failNow(failure));
 }
}
```

4. 使用 Maven 在我的应用上运行 JUnit 测试，可从应用的根目录运行以下命令：

```
mvn clean verify
```

您可以检查 目标/surefire-reports 的测试结果。com.example.MyAppTest.txt 文件包含测试结果。

第 5 章 创建 ECLIPSE VERT.X 项目的其它方法

本节显示了创建 Eclipse Vert.x 项目的不同方法。

5.1. 在命令行中创建 ECLIPSE VERT.X 项目

您可以在命令行中使用 Eclipse Vert.x Maven 插件来创建 Eclipse Vert.x 项目。您可以在命令行中指定属性和值。

前提条件

- 安装了 OpenJDK 8 或 OpenJDK 11。
- 已安装 Maven 3 或更高版本。
- 提供文本 ior 或 IDE。
- 提供 curl 或 HTTPie 或浏览器以保证 HTTP 可重复使用。

流程

1. 在命令终端中，输入以下命令验证 Maven 是否使用 OpenJDK 8 或 OpenJDK 11，并且 Maven 版本为 3.6.0 或更高版本：

```
mvn --version
```

2. 如果前面的命令没有返回 OpenJDK 8 或 OpenJDK 11，请将到 OpenJDK 8 或 OpenJDK 11 的路径添加到 PATH 环境变量中，然后再次输入该命令。

3. 创建目录并前往目录位置。

```
mkdir getting-started && cd getting-started
```

4. 使用以下命令，使用 Eclipse Vert.x Maven 插件创建新项目。

```
mvn io.reactiverse:vertx-maven-plugin:${vertx-maven-plugin-version}:setup -
DvertxBom=vertx-dependencies \
-DvertxVersion=${vertx_version} \
-DprojectGroupId= ${project_group_id} \
-DprojectArtifactId= ${project_artifact_id} \
-DprojectVersion=${project-version} \
-Dverticle=${verticle_class} \
-Ddependencies=${dependency_names}
```

以下示例演示了如何使用 命令创建 Eclipse Vert.x 应用。

```
mvn io.reactiverse:vertx-maven-plugin:1.0.24:setup -DvertxBom=vertx-dependencies \
-DvertxVersion=4.3.7.redhat-00002 \
-DprojectGroupId=io.vertx.myapp \
-DprojectArtifactId=my-new-project \
-DprojectVersion=1.0-SNAPSHOT \
-DvertxVersion=4.3.7.redhat-00002 \
-Dverticle=io.vertx.myapp.MainVerticle \
-Ddependencies=web
```

下表列出了您可以使用 setup 命令定义的属性：

属性	默认值	Description
vertx_version	Eclipse Vert.x 的版本。	要在项目中使用的 Eclipse Vert.x 版本。
project_group_id	**io.vertx.example**	项目的唯一标识符。
project_artifact_id	**my-vertx-project**	项目和项目目录的名称。如果没有指定 **project_artifact_id**，则 Maven 插件会启动交互模式。如果 目录已存在，生成会失败。
project-version	**1.0-SNAPSHOT**	项目的版本。
verticle_class	**io.vertx.example.MainVerticle**	由 verticle 参数创建的新 verticle 类文件。

属性	默认值	Description
dependency_names	可选参数	要添加到项目中以逗号分开的依赖关系列表。您还可以使用以下语法来配置依赖项： **groupId:artifactId:version:classifier** 例如： - 从 BOM 继承版本，使用以下语法： **io.vertx:vertxcodetrans** - 指定依赖项使用以下语法： **commons-io:commons-io:2.5** - 使用类符指定依赖项，使用以下语法： **io.vertx:vertx-template-engines:3.4.1:shaded**

该命令会创建一个空 Eclipse Vert.x 项目，其中包含 getting-started 目录中的以下工件：

- Maven build desiptor pom.xml conured to build and run aplication
- src/main/java 文件夹中的示例

5. 在 pom.xml 文件中，指定包含 Eclipse Vert.x 工件的存储库以构建应用程序。

```
<repositories>
   <repository>
     <id>redhat-ga</id>
     <name>Red Hat GA Repository</name>
     <url>https://maven.repository.redhat.com/ga/</url>
   </repository>
 </repositories>
```

或者，您也可以配置 Maven 存储库，以在 settings.xml 文件中指定构建工件。如需更多信息，请参阅为 Eclipse Vert.x 项目配置 Apache Maven 存储库。

6. 使用 Eclipse Vert.x 项目作为模板来创建自己的应用程序。

7. 从应用的根目录使用 Maven 构建应用。

```
mvn package
```

8. 从应用的根目录使用 Maven 运行应用。

```
mvn vertx:run
```

5.2. 使用 COMMUNITY VERT.X STARTER 创建 ECLIPSE VERT.X 项目

您可以使用 community Vert.x starter 创建 Eclipse Vert.x 项目。初学者会创建一个社区项目。您必须将社区项目转换为红帽构建的 Eclipse Vert.x 项目。

前提条件

- 安装了 OpenJDK 8 或 OpenJDK 11。
- 已安装 Maven 3 或更高版本。
- 提供文本 ior 或 IDE。
- 提供 curl 或 HTTPie 或浏览器以保证 HTTP 可重复使用。

流程

1. 在命令终端中，输入以下命令验证 Maven 是否使用 OpenJDK 8 或 OpenJDK 11，并且 Maven 版本为 3.6.0 或更高版本：

```
mvn --version
```

2. 如果前面的命令没有返回 OpenJDK 8 或 OpenJDK 11，请将到 OpenJDK 8 或 OpenJDK 11 的路径添加到 PATH 环境变量中，然后再次输入该命令。

3. 进入 Vert.x Starter。

4. 选择 Eclipse Vert.x 的 Version。

5. 选择 Java 作为语言。

6. 选择 Maven 作为构建工具。

7. 输入 组 Id，它是项目的唯一标识符。对于此过程，请保留默认的 com.example。

8. 输入 Artifact Id，这是您的项目和项目目录的名称。对于此过程，请保留默认的 starter。

9. 指定您要添加到项目中的依赖项。对于此过程，在 Depencies 文本框中键入 Vert.x Web 未指定值，也可以从 Depencies 列表中选择它。

10. 点 Advanced options 选择 OpenJDK 版本。对于此过程，请保留默认的 JDK 11。

11. 单击 Generate 项目。下载包含 Eclipse Vert.x 项目的构件的 starter.zip 文件。

12. 创建 已启动的目录。

13. 将 ZIP 文件的内容提取到 getting-started 文件夹。Vert.x Starter 创建一个具有以下工件的 Eclipse Vert.x 项目：

    - Maven 构建削减 pom.xml 文件.该文件有配置来构建和运行您的责任。
    - src/main/java 文件夹中的示例.
    - Sample 测试使用 src/test/java 文件夹中的 JUnit 5。

- ifigution to enforce Code风格.
- git configtion to ignore 文件.

14. 要将社区项目转换为红帽构建的 Eclipse Vert.x 项目，请在 pom.xml 文件中替换以下值：
    - vertx.version - 指定您要使用的 Eclipse Vert.x 版本。例如，如果您想要使用 Eclipse Vert.x 4.3.7 版本，请将版本指定为 4.3.7.redhat-00002。
    - vertx-stack-depchain - 将这个依赖项替换为 vertx-dependencies。
15. 指定要在 pom.xml 文件中构建应用程序的 Eclipse Vert.x 工件的软件仓库。

```
<repositories>
  <repository>
    <id>redhat-ga</id>
    <name>Red Hat GA Repository</name>
    <url>https://maven.repository.redhat.com/ga/</url>
  </repository>
 </repositories>
```

或者，您也可以配置 Maven 存储库，以在 settings.xml 文件中指定构建工件。如需更多信息，请参阅为 Eclipse Vert.x 项目配置 Apache Maven 存储库。

16. 使用 Eclipse Vert.x 项目作为模板来创建自己的应用程序。
17. 从应用的根目录使用 Maven 构建应用。

```
mvn package
```

18. 从应用的根目录使用 Maven 运行应用。

```
mvn exec:java
```

19. 验证应用是否正在运行。

使用 curl 或您的浏览器，验证您的应用是否在运行 http://localhost:8888，并返回 "Hello from Vert.x!" 作为回答。

```
$ curl http://localhost:8888
Hello from Vert.x!
```

第 6 章 为 ECLIPSE VERT.X 项目配置 APACHE MAVEN 存储库

Apache Maven 是 Java 应用程序开发中使用的分布式构建自动化工具，用于创建、管理和构建软件项目。Maven 使用名为 Project Object Model(POM)文件的标准配置文件来定义项目并管理构建流程。POM 文件描述了使用 XML 文件生成项目打包和输出的模块和组件依赖项、构建顺序和目标。这可确保以正确、一致的方式构建项目。

Maven 插件

Maven 插件是 POM 文件定义的部分，可实现一个或多个目标。Eclipse Vert.x 应用程序使用以下 Maven 插件：

- Eclipse Vert.x Maven 插件(vertx-maven-plugin)：启用 Maven 来创建 Eclipse Vert.x 项目，支持生成 uber-JAR 文件，并提供开发模式。
- Maven Surefire 插件(maven-surefire-plugin):在构建生命周期测试期间使用，用于在应用程序上执行单元测试。该插件生成包含测试报告的文本和 XML 文件。

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存储库，可在开发团队之间共享通用构件。也可从第三方获取存储库。

在 Eclipse Vert.x 项目中，您可以使用：

- 在线 Maven 软件仓库
- 下载 Eclipse Vert.x Maven 存储库

6.1. 为在线存储库配置 MAVEN SETTINGS.XML 文件

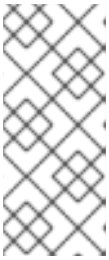
您可以通过配置用户 settings.xml 文件，将在线 Eclipse Vert.x 存储库与 Eclipse Vert.x Maven 项目搭配使用。这是推荐的方法。与共享服务器上的存储库管理器或存储库一起使用的 Maven 设置可以更好地控制项目和管理。

注意

当您通过修改 Maven settings.xml 文件配置存储库时，更改将应用到所有 Maven 项目。

流程

1. 在文本编辑器中打开 Maven ~/.m2/settings.xml 文件，或者集成开发环境(IDE)。



注意

如果 ~/.m2/ 目录中没有 settings.xml 文件，请将 $MAVEN_HOME/.m2/conf/ 目录中的 settings.xml 文件复制到 ~/.m2/ conf/ 目录中。

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

```
<!-- Configure the 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>
 </profile>
```

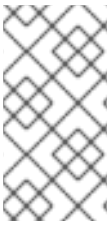
3. 在 settings.xml 文件的 <activeProfiles > 元素中添加以下行并保存该文件。

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

6.2. 下载并配置 ECLIPSE VERT.X MAVEN 存储库

如果您不想使用在线 Maven 存储库，您可以下载并配置 Eclipse Vert.x Maven 存储库，以使用 Maven 创建 Eclipse Vert.x 应用程序。Eclipse Vert.x Maven 存储库包含 Java 开发人员用来构建应用

程序的许多要求。这个步骤描述了如何编辑 settings.xml 文件来配置 Eclipse Vert.x Maven 存储库。



注意

当您通过修改 Maven settings.xml 文件配置存储库时，更改将应用到所有 Maven 项目。

流程

1. 从红帽客户门户网站的 软件下载 页面下载 Eclipse Vert.x Maven 存储库 ZIP 文件。要下载软件，您必须登录门户。

2. 展开下载的存档。

3. 将目录更改为 ~/.m2/ 目录，并在文本编辑器或集成开发环境(IDE)中打开 Maven settings.xml 文件。

4. 将以下行添加到 settings.xml 文件的 & lt;profiles > 元素，其中 *MAVEN_REPOSITORY* 是您下载的 Eclipse Vert.x Maven 存储库的路径。*MAVEN_REPOSITORY* 的格式必须是 file://$PATH，例如 file:///home/userX/rhb-vertx-4.1.5.SP1-maven-repository/maven-repository。

```
<!-- Configure the Maven repository -->
<profile>
 <id>red-hat-enterprise-maven-repository</id>
 <repositories>
  <repository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>MAVEN_REPOSITORY</url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>
    <enabled>false</enabled>
   </snapshots>
  </repository>
 </repositories>
</profile>
```

5. 在 settings.xml 文件的 <activeProfiles > 元素中添加以下行并保存该文件。

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

重要

如果您的 Maven 存储库包含过时的工件，您可能会在构建或部署项目时得到以下 Maven 错误消息之一，其中 *ARTIFACT_NAME* 是缺少工件的名称，*PROJECT_NAME* 是您要构建的项目的名称：

- 缺少工件 *PROJECT_NAME*
- [ERROR] 在项目 *ARTIFACT_NAME* 上执行目标失败；Could 不会解析 *PROJECT_NAME* 的依赖项

要解决这个问题，请删除 ~/.m2/repository 目录中的本地存储库的缓存版本，以强制下载最新的 Maven 工件。

第 7 章 其他资源

- 有关 Maven Surefire 插件的更多信息，请参阅 Apache Maven Project 网站。
- 有关 JUnit 5 测试框架的信息，请参见 JUnit 5 网站。