Red Hat build of Eclipse Vert.x 4.3

Eclipse Vert.x の使用

Eclipse Vert.x 4.3.7 での使用

Last Updated: 2023-04-08

Red Hat build of Eclipse Vert.x 4.3 Eclipse Vert.x の使用

Eclipse Vert.x 4.3.7 での使用

法律上の通知

概要

本ガイドでは、Apache Maven を使用して簡単な Eclipse Vert.x アプリケーションを作成する方法を説明します。

目次

第1章 使用するための要件

本ガイドでは、概念と、開発者が Eclipse Vert.x ランタイムを使用する際に必要となる実用的な詳細情報を説明します。

アプリケーション開発者は、Eclipse Vert.x を使用して、OpenShift 環境で実行される Java で記述されたマイクロサービスベースのアプリケーションを作成できます。

本ガイドでは、簡単な Eclipse Vert.x プロジェクトを作成、パッケージ化、実行、およびテストする方法を説明します。

前提条件

- OpenJDK 8 または OpenJDK 11 がインストールされ、**JAVA_HOME** 環境変数が Java SDK の場所を指定していること。Red Hat カスタマーポータルにログインして、Software Downloads から Red Hat ビルドの Open JDK をダウンロードします。
- Apache Maven 3.6.0 以降がインストールされている。Maven は Apache Maven Project の Web サイトからダウンロードできます。

第2章 ECLIPSE VERT.X の概要

Eclipse Vert.x は、Java 仮想マシン (JVM) で実行されるリアクティブで非ブロッキングの非同期アプリケーションを作成するために使用されるツールキットです。

Eclipse Vert.x はクラウドネイティブとなるように設計されています。これにより、アプリケーションは非常に少ないスレッドを使用できます。これにより、新規スレッドの作成時に生じるオーバーヘッドを回避します。これにより、Eclipse Vert.x アプリケーションおよびサービスで、クラウド環境でメモリーと CPU クォータが効果的に使用されます。

OpenShift で Eclipse Vert.x ランタイムを使用すると、リアクティブなシステムのビルドが簡単になります。ローリング更新、サービス検出、カナリアデプロイメントなどの OpenShift プラットフォーム機能も利用できます。OpenShift では、外部化の設定、ヘルスチェック、サーキットブレーカー、フェイルオーバーなどのマイクロサービスパターンをアプリケーションに実装できます。

2.1. ECLIPSE VERT.X の主な概念

本セクションでは、Eclipse Vert.x ランタイムに関連する主な概念を説明します。また、リアクティブなシステムの概要も説明します。

クラウドネイティブおよびコンテナーネイティブアプリケーション

クラウドネイティブアプリケーションは、通常マイクロサービスを使用してビルドされます。分離コンポーネントの分散システムを形成するように設計されています。これらのコンポーネントは、通常、多数のノードを含むクラスターでコンテナー内で実行されます。これらのアプリケーションは、個々のコンポーネントの障害に対して耐性があることが期待されており、サービスのダウンタイムなく更新できます。クラウドネイティブアプリケーションに基づくシステムは、基盤となるクラウドプラットフォーム (OpenShift など) によって提供される自動デプロイメント、スケーリング、管理タスク、メンテナーンスタスクに依存します。管理および管理タスクは、個々のマシンレベルではなく、既成の管理およびオーケストレーションツールを使用してクラスターレベルで実行されます。

リアクティブシステム

reactive manifesto に定義されているリアクティブなシステムは、以下の特徴を持つ分散システムです。

柔軟性

システムは、さまざまなワークロードで応答性を維持し、必要に応じて個々のコンポーネントをスケーリングして負荷分散することで、ワークロードの違いに対応します。Elastic アプリケーションは、同時に受け取る要求の数に関係なく、同じ品質のサービスを提供します。

耐久性

システムが各コンポーネントで障害が発生した場合でも応答し続けます。システムでは、コンポーネントは相互に分離されます。これにより、個々のコンポーネントに障害が発生した場合に迅速に復元するのに役立ちます。1 つのコンポーネントの障害は、他のコンポーネントの機能に影響を与えることはありません。これにより、分離されたコンポーネントの障害により他のコンポーネントがブロックされ、徐々に障害が発生するようなカスケード障害が防止されます。

応答の早さ

応答するシステムは、一貫性のあるサービス品質を確保するために、合理的な時間内に要求に常に応答するように設計されています。応答を維持するには、アプリケーション間の通信チャネルをブロックすることはできません。

メッセージ駆動型

アプリケーションの個々のコンポーネントは、非同期のメッセージパスを使用して相互に通信します。マウスクリックやサービスの検索クエリーなど、イベントが発生した場合、サービスは一般的なチャネル (イベントバス) にメッセージを送信します。メッセージはそれぞれのコンポーネントに

よってキャッチされ、処理されます。

リアクティブシステムは分散システムです。これらは、アプリケーション開発に非同期プロパティーを使用できるように設計されています。

リアクティブプログラミング

リアクティブシステムの概念は、分散システムのアーキテクチャーを記述しますが、リアクティブプログラミングは、アプリケーションをコードレベルでリアクティブにする手法を指します。リアクティブプログラミングは、非同期およびイベント駆動型のアプリケーションを記述する開発モデルです。リアクティブアプリケーションでは、コードはイベントまたはメッセージに反応します。

リアクティブプログラミングにはいくつかの実装があります。たとえば、コールバックを使用した簡単な実装、Reactive Extensions (Rx) を使用した複雑な実装、およびコルーチンを使用した複雑な実装などです。

Reactive Extensions (Rx) は、Java におけるリアクティブプログラミングの最も成熟した形式の 1 つです。これは **RxJava** ライブラリーを使用します。

第3章 POM ファイルを使用した ECLIPSE VERT.X プロジェクトの作成

基本的な Eclipse Vert.x アプリケーションの開発時に、以下のアーティファクトを作成する必要があります。これらのアーティファクトは、最初の **getting-started** Eclipse Vert.x プロジェクトで作成します。

- Eclipse Vert.x メソッドを含む Java クラス。
- アプリケーションをビルドするために Maven が必要とする情報が含まれる **pom.xml** ファイル。

次の手順では、応答として **Greetings!** を返す簡単な **Greeting** アプリケーションを作成します。

注記

Eclipse Vert.x は、アプリケーションを OpenShift にビルドおよびデプロイするために OpenJDK 8 および OpenJDK 11 をベースとしたビルダーイメージをサポートします。Oracle JDK および OpenJDK 9 のビルダーイメージはサポートされていません。

前提条件

- OpenJDK 8 または OpenJDK 11 がインストールされている。
- Maven がインストールされている。

手順

1. 新しいディレクトリー **getting-started** を作成し、そこに移動します。

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

これは、アプリケーションのルートディレクトリーです。

2. ルートディレクトリーにディレクトリー構造 **src/main/java/com/example/** を作成し、そこに移動します。

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

3. アプリケーションコードを含む Java クラスファイル **MyApp.java** を作成します。

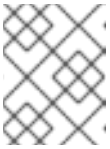
```
package com.example;

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());
                }
            });
    }
}
```

アプリケーションは、ポート 8080 で HTTP サーバーを起動します。リクエストを送信すると、**Greetings!** メッセージが返されます。

4. アプリケーションルートディレクトリー **getting-started** に、以下の内容で **pom.xml** ファイルを作成します。

- **<dependencyManagement>** セクションに、**io.vertx:vertx-dependencies** アーティファクトを追加します。
- **type** を **pom** として指定し、**scope** を **import** として指定します。
- **<project>** セクションの **<properties>** で、EclipseVert.x と EclipseVert.xMaven Plugin のバージョンを指定します。



注記

プロパティーを使用して、リリースごとに変更する値を設定できます。たとえば、製品またはプラグインのバージョンです。

- **<project>** セクションの **<plugin>** で、**vertx-maven-plugin** を指定します。Eclipse Vert.x Maven Plugin は、アプリケーションのパッケージ化に使用されます。
- **repositories** および **pluginRepositories** を追加して、アプリケーションのビルド用のアーティファクトおよびプラグインが含まれるリポジトリーを指定します。
pom.xml には、以下のアーティファクトが含まれます。

```
<?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. アプリケーションのルートディレクトリーから Maven を使用してアプリケーションをビルドします。

```
mvn vertx:run
```

6. アプリケーションが実行していることを確認します。
curl またはブラウザーを使用して、アプリケーションが **http://localhost:8080** で実行されているかどうかを確認し、**"Greetings!"** がレスポンスとして返されるかを確認します。

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

第4章 JUNIT を使用した ECLIPSE VERT.X アプリケーションのテスト

getting-started プロジェクトで Eclipse Vert.x アプリケーションをビルドした後に、アプリケーションを JUnit 5 フレームワークでテストし、想定どおりに実行されるようにします。JUnit 5 のテストでは、Eclipse Vert.x **pom.xml** ファイルの次の 2 つの依存関係が使用されます。

```
<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>
```

- テストには **vertx-junit5** 依存関係が必要です。JUnit 5 は、**@Test**, **@BeforeEach**, **@DisplayName** などのさまざまなアノテーションを提供します。**Vertx** および **VertxTestContext** インスタンスの非同期注入を要求するために使用されます。
- **junit-jupiter-engine** 依存関係は、ランタイム時にテストを実行するために必要です。

前提条件

- **pom.xml** ファイルを使用して Eclipse Vert.x **getting-started** プロジェクトを構築している。

手順

1. 生成された **pom.xml** ファイルを開き、Surefire Maven プラグインのバージョンを設定します。

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

2. ルートディレクトリーにディレクトリー構造 **src/test/java/com/example/** を作成し、そこに移動します。

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

3. アプリケーションコードを含む Java クラスファイル **MyTestApp.java** を作成します。

```
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. Maven を使用してアプリケーションで JUnit テストを実行するには、アプリケーションのルートディレクトリーから以下のコマンドを実行します。

```
mvn clean verify
```

テスト結果は、**target/surefire-reports** で確認できます。**com.example.MyAppTest.txt** ファイルにはテスト結果が含まれます。

第5章 ECLIPSE VERT.X プロジェクトを作成するための他の方法

このセクションでは、Eclipse Vert.x プロジェクトを作成するさまざまな方法について説明します。

5.1. コマンドラインでの ECLIPSE VERT.X プロジェクトの作成

コマンドラインで Eclipse Vert.x Maven プラグインを使用して、Eclipse Vert.x プロジェクトを作成できます。コマンドラインで属性と値を指定できます。

前提条件

- OpenJDK 8 または OpenJDK 11 がインストールされている。
- Maven 3 以降がインストールされている。
- テキストエディターまたは IDE が利用できる。
- HTTP リクエストを実行する Curl または HTTPie、またはブラウザーが利用できる。

手順

1. コマンドターミナルで以下のコマンドを入力して、Maven が OpenJDK 8 または OpenJDK 11 を使用し、Maven のバージョンが 3.6.0 以降であることを確認します。

```
mvn --version
```

2. 上記のコマンドで OpenJDK 8 または OpenJDK 11 が返されない場合は、OpenJDK 8 または OpenJDK 11 へのパスを PATH 環境変数に追加し、コマンドを再度入力します。
3. ディレクトリーを作成し、そのディレクトリーの場所に移動します。

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

4. Eclipse Vert.x Maven プラグインを使用して新しいプロジェクトを作成するには、以下のコマンドを使用します。

```
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}
```

以下の例は、説明したコマンドを使用して Eclipse Vert.x アプリケーションを作成する方法を示しています。

```
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
```

次の表に、**setup** コマンドで定義できる属性を示します。

属性	デフォルト値	説明
vertx_version	Eclipse Vert.x のバージョン。	プロジェクトで使用する Eclipse Vert.x のバージョン。
project_group_id	**io.vertx.example**	プロジェクトの一意識別子。
project_artifact_id	**my-vertx-project**	プロジェクトおよびプロジェクトディレクトリーの名前。**project_artifact_id** を指定しないと、Maven プラグインはインタラクティブモードを起動します。ディレクトリーがすでに存在する場合、生成は失敗します。
project-version	**1.0-SNAPSHOT**	プロジェクトのバージョン。
verticle_class	**io.vertx.example.MainVerticle**	Verticle パラメーターによって作成される新しい verticle クラスファイル。

属性	デフォルト値	説明
dependency_names	オプションのパラメーター	プロジェクトに追加する依存関係のコンマ区切りリスト。以下の構文を使用して依存関係を設定することもできます。 **groupId:artifactId:version:classifier** 以下に例を示します。 - BOM からバージョンを継承する場合は、以下の構文を使用します。 **io.vertx:vertxcodetrans** - 依存関係を指定するには、以下の構文を使用します。 **commons-io:commons-io:2.5** - 分類子で依存関係を指定するには、以下の構文を使用します。 **io.vertx:vertx-template-engines:3.4.1:shaded**

このコマンドは、**getting-started** ディレクトリーで以下のアーティファクトを使用して空の Eclipse Vert.x プロジェクトを作成します。

- アプリケーションをビルドおよび実行するように設定された Maven ビルド記述子 **pom.xml**
- **src/main/java** フォルダーの verticle の例

5. **pom.xml** ファイルで、アプリケーションをビルドするための Eclipse Vert.x アーティファクトを含むリポジトリーを指定します。

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

または、**settings.xml** ファイルでビルドアーティファクトを指定するように Maven リポジトリーを設定することもできます。詳細は、Eclipse Vert.x プロジェクトの Apache Maven リポジトリーの設定 セクションを参照してください。

6. Eclipse Vert.x プロジェクトをテンプレートとして使用し、独自のアプリケーションを作成します。
7. アプリケーションのルートディレクトリーから Maven を使用してアプリケーションをビルドします。

```
mvn package
```

8. アプリケーションのルートディレクトリーから Maven を使用してアプリケーションを実行します。

```
mvn vertx:run
```

5.2. コミュニティー VERT.X スターターを使用した ECLIPSE VERT.X プロジェクトの作成

コミュニティー Vert.x スターターを使用して Eclipse Vert.x プロジェクトを作成できます。スターターはコミュニティープロジェクトを作成します。コミュニティープロジェクトを Red Hat ビルドの Eclipse Vert.x プロジェクトに変換する必要があります。

前提条件

- OpenJDK 8 または OpenJDK 11 がインストールされている。
- Maven 3 以降がインストールされている。
- テキストエディターまたは IDE が利用できる。
- HTTP リクエストを実行する Curl または HTTPie、またはブラウザーが利用できる。

手順

1. コマンドターミナルで以下のコマンドを入力して、Maven が OpenJDK 8 または OpenJDK 11 を使用し、Maven のバージョンが 3.6.0 以降であることを確認します。

```
mvn --version
```

2. 上記のコマンドで OpenJDK 8 または OpenJDK 11 が返されない場合は、OpenJDK 8 または OpenJDK 11 へのパスを PATH 環境変数に追加し、コマンドを再度入力します。
3. Vert.x Starter に移動します。
4. Eclipse Vert.x の **Version** を選択します。
5. 言語として **Java** を選択します。
6. ビルドツールとして **Maven** を選択します。
7. プロジェクトの一意識別子である **Group Id** を入力します。この手順では、デフォルトの **com.example** のままにしておきます。
8. プロジェクトおよびプロジェクトディレクトリーの名前である **Artifact Id** を入力します。この手順では、デフォルトの **starter** のままにしておきます。

9. プロジェクトに追加する依存関係を指定します。この手順では、 **Dependencies**テキストボックスに入力するか、**Dependencies**のリストから選択して、**Vert.x Web** 依存関係を追加します。

10. **Advanced options** をクリックして OpenJDK バージョンを選択します。この手順では、デフォルトの **JDK 11** のままにしておきます。

11. **Generate Project** をクリックします。Eclipse Vert.x プロジェクトのアーティファクトが含まれる **starter.zip** ファイルがダウンロードされます。

12. **getting-started** ディレクトリーを作成します。

13. ZIP ファイルの内容を **getting-started** フォルダーに展開します。Vert.x Starter は、以下のアーティファクトで Eclipse Vert.x プロジェクトを作成します。

    - Maven ビルド記述子 **pom.xml** ファイル。このファイルには、アプリケーションをビルドおよび実行するための設定が含まれています。
    - **src/main/java** フォルダーの verticle の例。
    - **src/test/java** フォルダーで JUnit 5 を使用したサンプルテスト。
    - コードスタイルを適用するためのエディター設定。
    - ファイルを無視するための Git 設定。

14. コミュニティープロジェクトを Red Hat ビルドの Eclipse Vert.x プロジェクトに変換するには、**pom.xml** ファイルの以下の値を置き換えます。

    - **vertx.version**: 使用する Eclipse Vert.x バージョンを指定します。たとえば、Eclipse Vert.x 4.3.7 バージョンを使用する場合は、バージョンを 4.3.7.redhat-00002 として指定します。
    - **vertx-stack-depchain**: この依存関係を **vertx-dependencies** に置き換えます。

15. **pom.xml** ファイルで、アプリケーションをビルドするための Eclipse Vert.x アーティファクトを含むリポジトリーを指定します。

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

    または、**settings.xml** ファイルでビルドアーティファクトを指定するように Maven リポジトリーを設定することもできます。詳細は、Eclipse Vert.x プロジェクトの Apache Maven リポジトリーの設定 セクションを参照してください。

16. Eclipse Vert.x プロジェクトをテンプレートとして使用し、独自のアプリケーションを作成します。

17. アプリケーションのルートディレクトリーから Maven を使用してアプリケーションをビルドします。

```
mvn package
```

18. アプリケーションのルートディレクトリーから Maven を使用してアプリケーションを実行します。

```
mvn exec:java
```

19. アプリケーションが実行していることを確認します。
curl またはブラウザーを使用して、アプリケーションが **http://localhost:8888** で実行されているかどうかを確認し、"Hello from Vert.x!" がレスポンスとして返されるかを確認します。

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

第6章 ECLIPSE VERT.X プロジェクトの APACHE MAVEN リポジトリーの設定

Apache Maven は分散型構築自動化ツールで、ソフトウェアプロジェクトの作成、ビルド、および管理を行うために Java アプリケーション開発で使用されます。Maven は Project Object Model (POM) ファイルと呼ばれる標準の設定ファイルを使用して、プロジェクトの定義や構築プロセスの管理を行います。POM ファイルは、モジュールおよびコンポーネントの依存関係、ビルドの順番、結果となるプロジェクトパッケージのターゲットを記述し、XML ファイルを使用して出力します。これにより、プロジェクトが適切かつ統一された状態でビルドされるようになります。

Maven プラグイン

Maven プラグインは、POM ファイルの定義済みの部分で 1 つ以上のゴールを達成します。Eclipse Vert.x アプリケーションは以下の Maven プラグインを使用します。

- Eclipse Vert.x Maven プラグイン (**vertx-maven-plugin**): Maven による Eclipse Vert.x プロジェクトの作成を実現し、uber-JAR ファイルの生成をサポートし、開発モードを提供します。
- Maven Surefire プラグイン (**maven-surefire-plugin**): ビルドライフサイクルのテストフェーズで使用され、アプリケーションでユニットテストを実行します。プラグインは、テストレポートが含まれるテキストファイルと XML ファイルを生成します。

Maven リポジトリー

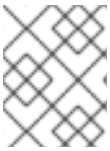
Maven リポジトリーには、Java ライブラリー、プラグイン、およびその他のビルドアーティファクトが格納されます。デフォルトのパブリックリポジトリーは Maven 2 Central Repository ですが、複数の開発チームの間で共通のアーティファクトを共有する目的で、社内のプライベートおよび内部リポジトリーとすることが可能です。また、サードパーティーのリポジトリーも利用できます。

Eclipse Vert.x プロジェクトでは、以下を使用できます。

- オンライン Maven リポジトリー
- Eclipse Vert.x Maven リポジトリーのダウンロード

6.1. オンラインリポジトリーの MAVEN の SETTINGS.XML ファイルの設定

ユーザーの **settings.xml** ファイルを設定すると、Eclipse Vert.x Maven プロジェクトでオンライン Eclipse Vert.x リポジトリーを使用できます。これは、推奨の手法です。リポジトリーマネージャーまたは共有サーバー上のリポジトリーと使用する Maven 設定は、プロジェクトの制御および管理性を向上させます。

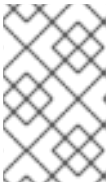


注記

Maven の **settings.xml** ファイルを変更してリポジトリーを設定する場合、変更はすべての Maven プロジェクトに適用されます。

手順

1. テキストエディターまたは統合開発環境 (IDE) で Maven の **~/.m2/settings.xml** ファイルを開きます。



注記

~/**.m2**/ ディレクトリーに **settings.xml** ファイルがない場合は、**$MAVEN_HOME/.m2/conf**/ ディレクトリーから ~/**.m2**/ ディレクトリーに **settings.xml** ファイルをコピーします。

2. 以下の行を Maven の **settings.xml** ファイルの **<profiles>** 要素に追加します。

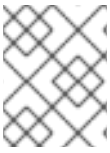
```
<!-- Configure the Maven repository -->
<profile>
 <id>red-hat-enterprise-maven-repository</id>
 <repositories>
  <repository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>https://maven.repository.redhat.com/ga/</url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>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

3. 以下の行を **settings.xml** ファイルの **<activeProfiles>** 要素に追加し、ファイルを保存します。

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

6.2. ECLIPSE VERT.X MAVEN リポジトリーのダウンロードおよび設定

オンライン Maven リポジトリーを使用しない場合は、Eclipse Vert.x Maven リポジトリーをダウンロードおよび設定し、Maven で Eclipse Vert.x アプリケーションを作成できます。Eclipse Vert.x Maven リポジトリーには、Java 開発者がアプリケーションの構築に通常使用する多くの要件が含まれています。この手順では、**settings.xml** ファイルを編集して Eclipse Vert.x Maven リポジトリーを設定する方法を説明します。



注記

Maven の **settings.xml** ファイルを変更してリポジトリーを設定する場合、変更はすべての Maven プロジェクトに適用されます。

手順

1. Red Hat カスタマーポータルの Software Downloads ページから Eclipse Vert.x Maven リポジトリーの ZIP ファイルをダウンロードします。ソフトウェアをダウンロードするには、ポータルにログインする必要があります。

2. ダウンロードしたアーカイブを展開します。

3. ~/**.m2**/ ディレクトリーに移動し、テキストエディターまたは統合開発環境 (IDE) で Maven の **settings.xml** ファイルを開きます。

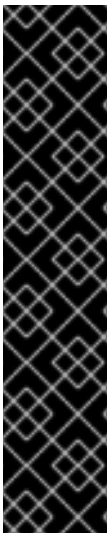
4. 以下の行を **settings.xml** ファイルの **<profiles>** 要素に追加します。**MAVEN_REPOSITORY**

はダウンロードした Eclipse Vert.x Maven リポジトリーのパスです。**MAVEN_REPOSITORY** の形式は **file://$PATH** である必要があります (例: **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. 以下の行を **settings.xml** ファイルの **<activeProfiles>** 要素に追加し、ファイルを保存します。

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



重要

Maven リポジトリーに古いアーティファクトが含まれる場合は、プロジェクトをビルドまたはデプロイしたときに以下のいずれかの Maven エラーメッセージが表示されることがあります。ここで、**ARTIFACT_NAME** は不明なアーティファクトの名前で、**PROJECT_NAME** はビルドを試みているプロジェクトの名前になります。

- **Missing artifact PROJECT_NAME**
- **[ERROR] Failed to execute goal on project ARTIFACT_NAME; Could not resolve dependencies for PROJECT_NAME**

この問題を解決するには、**~/.m2/repository** ディレクトリーにあるローカルリポジトリーのキャッシュバージョンを削除し、最新の Maven アーティファクトを強制的にダウンロードします。

第7章 関連情報

- Maven Surefire プラグインの詳細は、Apache Maven Project の Web サイトを参照してください。
- JUnit 5 テストフレームワークの詳細については、JUnit 5 の Web サイトを参照してください。