2. Despliegue de software

Aplicación java en maven

Víctor Ponz

Institut Educació Secundària
El Caminàs

Curso de especialización en Ciberseguridad

Contenidos

Introducción

Esta guía te guía a través del uso de Maven para construir un proyecto Java simple.

Lo que vas a construir

Crearás una aplicación que proporcione la hora del día y luego la construirás con Maven.

Lo que necesitarás

- Alrededor de 15 minutos
- java instalado
- Un editor de texto o IDE favorito
- JDK 8 o posterior

Configurar el proyecto

Primero necesitarás configurar un proyecto Java para que Maven lo construya. Para mantener el enfoque en Maven, haz el proyecto tan simple como sea posible por ahora. Crea esta estructura en una carpeta del proyecto de tu elección.

```
└── src
    └── main
        └── java
            └── hello
```

Dentro del directorio `src/main/java/hello`, puedes crear las clases Java que quieras. Para mantener la coherencia con el resto de esta guía, crea estas dos clases: `HelloWorld.java` y `Greeter.java`.

`-> src/main/java/hello/HelloWorld.java`

```java
package hello;

public class HelloWorld {
  public static void main(String[] args) {
    Greeter greeter = new Greeter();
    System.out.println(greeter.sayHello());
```

```
  }
}
```

-> src/main/java/hello/Greeter.java

```
package hello;

public class Greeter {
  public String sayHello() {
    return "Hello world!";
  }
}
```

Ahora que tienes un proyecto que está listo para ser construido con Maven, el siguiente paso es instalar Maven.

Maven se puede descargar como un archivo zip en https://maven.apache.org/download.cgi. Sólo se requieren los binarios, así que busca el enlace a apache-maven-{version}-bin.zip o apache-maven-{version}-bin.tar.gz. ç

Una vez descargado el archivo zip, descomprímelo en tu ordenador. A continuación, añada la carpeta *bin* a su ruta (muévelo a la carpeta `/opt` y ejecuta `export PATH=/opt/apache-maven-3.8.4/bin:$PATH`)

Para probar la instalación de Maven, ejecuta `mvn` desde la línea de comandos:

```
mvn -v
```

Si todo va bien, usted debe ser presentado con alguna información sobre la instalación de Maven. Tendrá un aspecto similar (aunque quizás ligeramente diferente) al siguiente:

```
Apache Maven 3.8.4 (9b656c72d54e5bacbed989b64718c159fe39b537)
Maven home: /opt/apache-maven-3.8.4
Java version: 11.0.13, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-amd64
Default locale: es_ES, platform encoding: UTF-8
OS name: "linux", version: "5.11.0-27-generic", arch: "amd64", family: "unix"
```

¡Enhorabuena! Ya tienes Maven instalado.

Definir una construcción simple de Maven

Ahora que Maven está instalado, necesitas crear una definición de proyecto Maven. Los proyectos Maven se definen con un archivo XML llamado `pom.xml`. Entre otras cosas, este archivo da el nombre del proyecto, la versión, y las dependencias que tiene en las bibliotecas externas.

Crea un archivo llamado *pom*.xml en la raíz del proyecto (es decir, ponlo junto a la carpeta `src`) y dale el siguiente contenido:

```
<?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
 ↪  https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.springframework</groupId>
    <artifactId>gs-maven</artifactId>
    <packaging>jar</packaging>
    <version>0.1.0</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.4</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer

 ↪  implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer"
```

```
                                    <mainClass>hello.HelloWorld</mainClass>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

Con la excepción del elemento opcional `<packaging>`, este es el archivo *pom.* xml más simple posible necesario para construir un proyecto Java. Incluye los siguientes detalles de la configuración del proyecto:

- `<packaging>`. Versión del modelo POM (siempre 4.0.0).
- `<groupId>`. Grupo u organización a la que pertenece el proyecto. A menudo se expresa como un nombre de dominio invertido.
- `<artifactId>`. Nombre que se le dará al artefacto de biblioteca del proyecto (por ejemplo, el nombre de su archivo JAR).
- `version`. Versión del proyecto que se está construyendo.
- `packaging`. Cómo debe empaquetarse el proyecto. Por defecto, «jar» para el empaquetado de archivos JAR.

En este punto tienes un proyecto Maven mínimo, pero capaz, definido.

Construir código Java

Maven está ahora listo para construir el proyecto. Ahora puedes ejecutar varios objetivos del ciclo de vida de la construcción con Maven, incluyendo objetivos para compilar el código del proyecto, crear un paquete de biblioteca (como un archivo JAR), e instalar la biblioteca en el repositorio local de dependencias de Maven.

Para probar la compilación, emita lo siguiente en la línea de comandos:

```
mvn compile
```

La salida será parecida a esta:

```
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< org.springframework:gs-maven >----------
----------
[INFO] Building gs-maven 0.1.0
[INFO] --------------------------------[ jar ]----------------------
----------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ gs-
maven ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
[INFO] skip non existing resourceDirectory /home/alumno/Documentos/maven/src/main/
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ gs-
maven ---
[INFO] Nothing to compile - all classes are up to date
[INFO] ------------------------------------------------------------------
----------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------
----------
[INFO] Total time:  0.515 s
[INFO] Finished at: 2022-01-11T19:47:50+01:00
[INFO] ------------------------------------------------------------------
----------
```

Esto ejecutará Maven, diciéndole que ejecute el objetivo de *compilación*. Cuando termine, deberías encontrar los archivos *.class* compilados en el directorio *target/classes*.

Dado que es poco probable que quiera distribuir o trabajar con los archivos . *class* directamente, probablemente querrá ejecutar el objetivo de *paquete* en su lugar:

```
mvn package
```

El objetivo de *package* compilará su código Java, ejecutará cualquier prueba, y terminará empaquetando el código en un archivo JAR dentro del directorio de *destino*. El nombre del archivo JAR se basará en el `<artifactId>` y la `<version>`del proyecto. Por ejemplo, dado el archivo *pom.xml* mínimo de antes, el archivo JAR se llamará `gs-maven-0.1.0.jar`

Para ejecutar el archivo JAR ejecute

```
java -jar target/gs-maven-0.1.0.jar
```

Maven también mantiene un repositorio de dependencias en tu máquina local (normalmente en un directorio *.m2/repository* en tu directorio personal) para acceder rápidamente a las dependencias del proyecto. Si quieres instalar el archivo JAR de tu proyecto en ese repositorio local, entonces debes invocar el objetivo de `install`

```
mvn install
```

El objetivo de *instalación* compilará, probará y empaquetará el código de su proyecto y luego lo copiará en el repositorio local de dependencias, listo para que otro proyecto lo referencie como una dependencia.

Hablando de dependencias, ahora es el momento de declarar las dependencias en la construcción de Maven.

Declarar dependencias

El sencillo ejemplo de Hola Mundo es completamente autocontenido y no depende de ninguna librería adicional. La mayoría de las aplicaciones, sin embargo, dependen de bibliotecas externas para manejar funcionalidades comunes y complejas.

Por ejemplo, suponga que además de decir «¡Hola Mundo!», quiere que la aplicación imprima la fecha y hora actuales. Aunque podría utilizar las facilidades de fecha y hora de las bibliotecas nativas de Java, puede hacer las cosas más interesantes utilizando las bibliotecas Joda Time.

En primer lugar, cambie `HelloWorld.java` para que se vea así

```
-> src/main/java/hello/HelloWorld.java
```

```java
package hello;

import org.joda.time.LocalTime;

public class HelloWorld {
  public static void main(String[] args) {
    LocalTime currentTime = new LocalTime();
    System.out.println("The current local time is: " + currentTime);
    Greeter greeter = new Greeter();
    System.out.println(greeter.sayHello());
  }
}
```

Aquí `HelloWorld` utiliza la clase `LocalTime` de Joda Time para obtener e imprimir la hora actual.

Si ejecutara `mvn compile` para construir el proyecto ahora, la construcción fallaría porque no ha declarado Joda Time como una dependencia de compilación en la construcción. Puede arreglar esto añadiendo las siguientes líneas al *pom.xml* (dentro del elemento `<project>`):

```
<dependencies>
        <dependency>
            <groupId>joda-time</groupId>
            <artifactId>joda-time</artifactId>
            <version>2.9.2</version>
        </dependency>
</dependencies>
```

Este bloque de XML declara una lista de dependencias para el proyecto. En concreto, declara una única dependencia para la biblioteca Joda Time. Dentro del elemento `<dependency>`, las coordenadas de la dependencia están definidas por tres subelementos:

- `<groupId>`- El grupo u organización al que pertenece la dependencia.
- `<artifactId>`- La biblioteca que se requiere.
- `<version>`- La versión específica de la biblioteca que se requiere.

Por defecto, todas las dependencias tienen un alcance de `compile`. Es decir, deben estar disponibles en tiempo de compilación. Además, puede especificar un elemento `scope` para especificar uno de los siguientes ámbitos:

- `provided` - Dependencias que son necesarias para compilar el código del proyecto, pero que serán proporcionadas en tiempo de ejecución por un contenedor que ejecute el código (por ejemplo, la API de Java Servlet).
- `test` - Dependencias que se utilizan para compilar y ejecutar pruebas, pero que no son necesarias para construir o ejecutar el código del proyecto en tiempo de ejecución.

Ahora si ejecutas `mvn compile` o `mvn package`, Maven debería resolver la dependencia de Joda Time desde el repositorio de Maven Central y la construcción será exitosa.

Escribir una prueba

Primero añade JUnit como una dependencia a tu pom.xml, en el ámbito de la prueba:

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

A continuación, cree un caso de prueba como el siguiente

`-> src/test/java/hello/GreeterTest.java`

```
package hello;

import static org.hamcrest.CoreMatchers.containsString;
import static org.junit.Assert.*;

import org.junit.Test;

public class GreeterTest {

  private Greeter greeter = new Greeter();

  @Test
  public void greeterSaysHello() {
    assertThat(greeter.sayHello(), containsString("Hello"));
  }

}
```

Añadimos también la dependencia `hamcrest`

```
<dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-all</artifactId>
    <version>1.3</version>
    <scope>test</scope>
</dependency>
```

Maven utiliza un plugin llamado «surefire» para ejecutar las pruebas unitarias. La configuración por defecto de este plugin compila y ejecuta todas las clases en `src/test/java` con un nombre que coincida con `*Test`. Puedes ejecutar las pruebas en la línea de comandos así

```
mvn test
```

o simplemente usar el paso `mvn install` como ya mostramos arriba (hay una definición del ciclo de vida donde «test» se incluye como una etapa en «install»).

Si todo va bien esta será la salida:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running hello.GreeterTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.053 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

Que indica que ha corrido un test y no se ha producido ningún error (`Failures: 0`)

Aquí está el archivo `pom.xml` completado:

```
<?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
 ↪  https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.springframework</groupId>
    <artifactId>gs-maven</artifactId>
    <packaging>jar</packaging>
    <version>0.1.0</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>joda-time</groupId>
            <artifactId>joda-time</artifactId>
```

```
            <version>2.9.2</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest-all</artifactId>
            <version>1.3</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.4</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer
↪   implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer"
                                    <mainClass>hello.HelloWorld</mainClass>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

> NOTA: El archivo **pom**.xml completado está usando el Maven Shade Plugin por la simple conveniencia de hacer el archivo JAR ejecutable. El enfoque de esta guía es empezar con Maven, no usar este plugin en particular.

Referencias

Traducido del original