EBook Gratis

APRENDIZAJE Apache Maven

Free unaffiliated eBook created from
Stack Overflow contributors.

#maven

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: apache-maven

It is an unofficial and free Apache Maven ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Apache Maven.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Apache Maven

Observaciones

Como lo describe su Guía de inicio oficial :

> Maven es un intento de aplicar **patrones** a la infraestructura de construcción de un proyecto para promover la comprensión y la productividad al proporcionar un camino claro en el uso de las **mejores prácticas** .

Maven es esencialmente una herramienta de gestión y comprensión de proyectos y, como tal, proporciona una manera de ayudar con la gestión:

- Construye
- Documentación
- Reportando
- Dependencias
- Control de versiones
- Lanzamientos
- Distribución

Por lo tanto, apoyar a los desarrolladores en muchas fases de todo el ciclo de vida del desarrollo de software (SDLC).

Esta filosofía es parte de Maven en su núcleo: es decir, la palabra *maven* significa *acumulador de conocimiento* (en yiddish).

Maven trata sobre la aplicación de **patrones** para lograr una infraestructura que muestre las características de visibilidad, reutilización, mantenibilidad y comprensibilidad.

- Maven nació del deseo muy práctico de hacer que varios proyectos funcionen de la misma manera, como se afirma en la declaración oficial de filosofía de Maven .
- Los desarrolladores podían moverse libremente entre proyectos, sabiendo claramente cómo funcionaban todos al comprender cómo funcionaba uno de ellos
- La misma idea se extiende a las pruebas, la generación de documentación, la generación de métricas e informes y la implementación

Versiones

Versión	Anunciar	Comentario	Fechas de lanzamiento
1.0-beta-2	anunciar	Primer lanzamiento (beta)	2002-03-30
1.0	anunciar	Primer lanzamiento oficial	2004-07-13
2.0	anunciar	Lanzamiento oficial **2.0**	2005-10-20

Versión	Anunciar	Comentario	Fechas de lanzamiento
3.0	anunciar	Lanzamiento oficial **3.0**	2010-10-08

Examples

Instalación o configuración

Las versiones binarias de Maven se pueden descargar desde el sitio web de Maven .

El binario viene como un archivo zip o como un archivo tar.gz. Después de descargarlo, se pueden seguir las instrucciones de la página de instalación :

- Asegúrese de que la variable de entorno `JAVA_HOME` esté establecida y apunte a su instalación de JDK (no a JRE). Por ejemplo, en una máquina Windows, esta carpeta de instalación puede corresponder a `C:\Program Files\Java\jdk1.8.0_51` .
- Extraiga el archivo de distribución en el directorio de su elección.
- Agregue el directorio `bin` directorio creado (llamado `apache-maven-3.3.9` para Maven 3.3.9) a la `PATH` entorno `PATH` . (Referencia para cambiarlo en Windows).
- Verifique que la configuración sea correcta ejecutando `mvn -version` en la línea de comandos.

No es necesario establecer la variable de entorno `M2_HOME` o `MAVEN_HOME` .

Instalación en Ubuntu

1. En una terminal ejecuta `sudo apt-get install maven`
2. Una vez que se haya terminado la instalación, compruebe que funciona correctamente con `mvn -v` el resultado debería ser:

```
Apache Maven 3.3.9
Maven home: /usr/share/maven
Java version: 1.8.0_121, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.8.0-parrot-amd64", arch: "amd64", family: "unix"
```

Si esto no funciona, asegúrese de tener un JDK instalado en su entorno `javac -version`

Configurando las Configuraciones Proxy

Si su conexión a Internet se proporciona a través de un proxy, Maven no podrá descargar archivos jar desde repositorios remotos, un problema común que enfrentan las empresas.

Para resolver este problema, a Maven se le deben proporcionar los detalles y las credenciales del proxy yendo a *{Ubicación de instalación de Maven}* → *conf* → `settings.xml` . Desplácese hacia abajo hasta la etiqueta `<proxies>` e ingrese los detalles aquí, usando el formato mencionado en los

comentarios.

Para usuarios de Eclipse.

Eclipse usa su propio archivo `settings.xml` para ejecutar Maven, cuya ubicación se puede encontrar en el menú *Ventana → Preferencias → Maven → Configuración de usuario → Configuración de usuario:.* Si el archivo no está disponible en la ubicación mencionada, simplemente créelo usted mismo o cree un duplicado del archivo desde la ubicación anterior *{ubicación de instalación de Maven} → conf →* `settings.xml` .

Para usuarios de IntelliJ

Abra la configuración y navegue a Maven -> Importando. (Esto puede estar anidado en Compilación, Ejecución, Despliegue -> Crear herramientas ->, dependiendo de la versión de IntelliJ que esté usando).

Establezca el campo denominado "Opciones de VM para el importador" como:

```
-DproxySet=true -DproxyHost=<HOST> -DproxyPort=<PORT>
-DproxySet=true -DproxyHost=myproxy.com -DproxyPort=8080
```

Aplicar y reiniciar IntelliJ.

Instalación en Mac OSX con Brew

1. En una terminal de ejecución `brew install maven`
2. Una vez finalizada la instalación, compruebe que maven funciona correctamente con `mvn -v` . La salida debe verse algo como:

```
Apache Maven 3.3.9
Maven home: /usr/local/Cellar/maven/3.3.9/libexec
Java version: 1.8.0_121, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.12.4", arch: "x86_64", family: "mac"
```

Si esto no funciona, asegúrese de tener un JDK instalado en su entorno `javac -version`

Lea Empezando con Apache Maven en línea:
https://riptutorial.com/es/maven/topic/898/empezando-con-apache-maven

Capítulo 2: Acceso a las informaciones de Maven en código.

Introducción

A veces es útil obtener las propiedades de Maven, como la versión actual, en el código. Aquí hay algunas maneras de hacerlo.

Examples

Obtener el número de versión dentro de un tarro

Si empaqueta su aplicación en un `jar` utilizando el `maven-jar-plugin` o el `maven-jar-plugin` `maven-assembly-plugin` , una manera fácil de obtener la versión actual de pom es agregar una entrada en el manifiesto, que luego está disponible en Java.

El secreto es establecer el indicador `addDefaultImplementationEntries` en verdadero (y el `addDefaultSpecificationEntries` es que también necesita la identificación del artefacto).

configuración del plugin jar :

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <mainClass>...</mainClass>
            <addDefaultImplementationEntries>
              true
            </addDefaultImplementationEntries>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

configuración del complemento de montaje :

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <configuration>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
```

```
        <archive>
            <manifest>
              <addDefaultImplementationEntries>true</addDefaultImplementationEntries>
            </manifest>
        </archive>
    </configuration>
    <executions>
        <execution .../>
    </executions>
</plugin>
```

`addDefaultImplementationEntries` indica a Maven que agregue los siguientes encabezados al `MANIFEST.MF` de su `MANIFEST.MF` jar:

```
Implementation-Title: display-version
Implementation-Version: 1.0-SNAPSHOT
Implementation-Vendor-Id: test
```

Ahora puedes usar esta línea de código en cualquier lugar de tu archivo para acceder al número de versión:

```
getClass().getPackage().getImplementationVersion()
```

Más información aquí y aquí .

Mantener un archivo de propiedades sincronizado usando el mecanismo de filtrado de propiedades de maven

Como explica esta documentación ,

> A veces, un archivo de recursos deberá contener un valor que solo se puede proporcionar en el momento de la compilación. Para lograr esto en Maven, ponga una referencia a la propiedad que contendrá el valor en su archivo de recursos usando la sintaxis `${<property name>}` . La propiedad puede ser uno de los valores definidos en su `pom.xml` , un valor definido en la `settings.xml` del usuario.xml, una propiedad definida en un archivo de propiedades externo o una propiedad del sistema.

Como ejemplo, vamos a crear un simple `info.txt` en `src/main/resources` contiene la versión pom y el tiempo de compilación.

1. cree un `src/main/resources/info.txt` con el siguiente contenido:

   version = $ {pom.version} build.date = $ {timestamp}

2. Pídale a Maven que *expanda* las propiedades estableciendo el `filtering` en verdadero:

   ```
   <build>
       <resources>
           <resource>
              <directory>src/main/resources</directory>
              <filtering>true</filtering>
   ```

```
        </resource>
    </resources>
</build>
```

3. con eso, la versión se actualizará, pero desafortunadamente un error dentro de Maven evita que la propiedad `${maven.build.timestamp}` pase al mecanismo de filtrado de recursos (más información aquí). Por lo tanto, vamos a crear una propiedad de `timestamp` como una solución! Agregue lo siguiente a las propiedades del pom:

```
<properties>
    <timestamp>${maven.build.timestamp}</timestamp>
    <maven.build.timestamp.format>yyyy-MM-dd'T'HH:mm</maven.build.timestamp.format>
</properties>
```

4. ejecutar maven, debería encontrar un `info.txt` en `target/classes` con un contenido como:

```
version=0.3.2
build.date=2017-04-20T13:56
```

Leyendo un pom.xml en tiempo de ejecución usando el complemento maven-model

Los otros ejemplos pueden ser la manera mejor y más estable de obtener un número de versión en una aplicación de ***forma estática*** . Esta respuesta propone una alternativa que muestra cómo hacerlo ***dinámicamente*** durante el tiempo de ejecución, utilizando la biblioteca *maven-model maven* .

Agregue la dependencia:

```
<dependency>
  <groupId>org.apache.maven</groupId>
  <artifactId>maven-model</artifactId>
  <version>3.3.9</version>
</dependency>
```

En Java, crea un `MavenXpp3Reader` para leer tu pom. Por ejemplo:

```
package de.scrum_master.app;

import org.apache.maven.model.Model;
import org.apache.maven.model.io.xpp3.MavenXpp3Reader;
import org.codehaus.plexus.util.xml.pull.XmlPullParserException;

import java.io.FileReader;
import java.io.IOException;

public class MavenModelExample {
    public static void main(String[] args) throws IOException, XmlPullParserException {
        MavenXpp3Reader reader = new MavenXpp3Reader();
        Model model = reader.read(new FileReader("pom.xml"));
        System.out.println(model.getId());
        System.out.println(model.getGroupId());
```

```
        System.out.println(model.getArtifactId());
        System.out.println(model.getVersion());
    }
}
```

El registro de la consola es el siguiente:

```
de.scrum-master.stackoverflow:my-artifact:jar:1.0-SNAPSHOT
de.scrum-master.stackoverflow
my-artifact
1.0-SNAPSHOT
```

Lea Acceso a las informaciones de Maven en código. en línea: https://riptutorial.com/es/maven/topic/9773/acceso-a-las-informaciones-de-maven-en-codigo-

Capítulo 3: Ciclo de construcción de Maven

Introducción

A continuación se incluye una lista completa de las fases del ciclo de vida de compilación predeterminado de Maven. Cada una de estas fases se invoca agregándola al comando `mvn` , por ejemplo, `mvn install` .

Examples

Maven construir fases de ciclo de vida

```
validate
```

Valida si el proyecto es correcto y toda la información requerida está disponible para la compilación.

```
initialize
```

Inicializa el entorno de compilación, por ejemplo, establece propiedades o crea directorios.

```
generate-sources
```

Genera código fuente para ser procesado en la fase de 'compilación'.

```
process-sources
```

Procesa el código fuente en caso de que sea necesario aplicar algún filtro.

```
generate-resources
```

Genera recursos para ser incluidos en el artefacto.

```
process-resources
```

Procesa y copia recursos en el directorio de salida (`${basedir}/target/classes`).

```
compile
```

Compila el código fuente del proyecto en el directorio de origen (`${basedir}/src/main/[java|groovy|...]`) en el directorio de salida (`${basedir}/target/classes`).

```
process-classes
```

Procesa los archivos `.class` generados en la fase de `compile` , por ejemplo, para realizar mejoras de código de bytes.

```
generate-test-sources
```

Genera código fuente de prueba para ser procesado en la fase de `test-compile` .

```
process-test-sources
```

Procesa el código fuente de la prueba en caso de que sea necesario aplicar algún filtro.

```
generate-test-resources
```

Genera recursos para pruebas.

```
process-test-resources
```

Procesa y copia recursos de prueba en el directorio de recursos (`${basedir}/src/main/resources`) en el directorio de salida de prueba (`${basedir}/target/test-classes`).

```
test-compile
```

Compila el código fuente en el directorio de origen de prueba ('$ {basedir} / src / test / [java | groovy | ...]') en el directorio de salida de prueba (`${basedir}/target/test-classes`).

```
process-test-classes
```

Los procesos prueban los archivos `.class` generados en la fase de `test-compile` , por ejemplo, para realizar mejoras de código de bytes (Maven 2.0.5 y superior).

```
test
```

Ejecuta pruebas utilizando algún marco de prueba adecuado. Nota: estos casos de prueba no se consideran para el empaquetado y la implementación.

```
prepare-package
```

Realiza los cambios finales y las validaciones antes de que finalmente se cree el paquete.

```
package
```

Empaqueta el código compilado y probado con éxito en algún formato distribuible como JAR, WAR, EAR en el directorio de destino (`${basedir}/target`).

```
pre-integration-test
```

Realiza acciones antes de que se ejecuten las pruebas de integración si requieren aplicar algunos cambios en el entorno para la aplicación.

```
integration-test
```

Procesa y posiblemente implementa la aplicación en un entorno donde se pueden ejecutar pruebas de integración.

```
post-integration-test
```

Realiza acciones después de las pruebas de integración, como la limpieza del entorno creado en la fase de `pre-integration-test` .

```
verify
```

Comprueba si un paquete es válido y cumple con los criterios de calidad requeridos.

```
install
```

Instala el artefacto en el repositorio local. Cualquier otro proyecto local puede usar este artefacto como una de sus dependencias después de eso (si su IDE no admite la *resolución de dependencias del área de trabajo de* todos modos).

```
deploy
```

Copia el paquete en un repositorio remoto para que esté disponible para otros desarrolladores.

Lea Ciclo de construcción de Maven en línea: https://riptutorial.com/es/maven/topic/9679/ciclo-de-construccion-de-maven

Capítulo 4: Complemento de montaje de Maven

Examples

Creando archivo .jar con todas las dependencias del proyecto.

Para crear un JAR que contenga todas sus dependencias, es posible utilizar el formato de descriptor integrado `jar-with-dependencies` . El siguiente ejemplo configura una ejecución del Complemento de ensamblaje vinculado a la fase del `package` , utilizando este descriptor incorporado y declarando una clase principal de `com.example` :

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>2.6</version>
  <executions>
    <execution>
      <id>make-assembly</id>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
      <configuration>
        <archive>
          <manifest>
              <mainClass>com.example</mainClass>
          </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Corriendo:

```
mvn clean package
```

en la línea de comandos se generará y se adjuntará al proyecto el jar-with-dependencies.

Si se necesita más control sobre este uber-jar , recurra al Complemento de sombra de Maven .

Lea Complemento de montaje de Maven en línea:
https://riptutorial.com/es/maven/topic/2308/complemento-de-montaje-de-maven

Capítulo 5: Crear un complemento de Maven

Introducción

Maven te permite implementar y usar complementos personalizados. Estos complementos permiten vincular el comportamiento adicional a cualquier fase del ciclo de vida de Maven.

Cada objetivo de Maven se crea al implementar un MOJO (Maven Ordinary Java Object): una clase de Java implementada con anotaciones que describe cómo invocarlo.

El prefijo de objetivo de un complemento se deriva de su nombre de artefacto. Un artefacto `hello-world-plugin` crea un prefijo de objetivo `hello-world` . El objetivo de `hello` se puede ejecutar con `mvn hello-world:hello` .

Observaciones

Un complemento de Maven es un JAR que contiene un `maven/plugins.xml` que describe los metadatos del complemento. Este archivo es generado por el `maven-plugin-plugin` .

Examples

Declarar un artefacto Maven como un complemento de Maven

Un artefacto construido por Maven puede ser declarado como un plugin Maven especificando el envase como `maven-plugin` en el `pom.xml` .

```
<packaging>maven-plugin</packaging>
```

Debe declarar una dependencia en la API del complemento y las anotaciones.

```
<dependency>
    <groupId>org.apache.maven</groupId>
    <artifactId>maven-plugin-api</artifactId>
    <version>3.3.9</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.maven.plugin-tools</groupId>
    <artifactId>maven-plugin-annotations</artifactId>
    <version>3.5</version>
    <scope>provided</scope>
</dependency>
```

Es necesario agregar un complemento para generar los metadatos.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
```

```
    <artifactId>maven-plugin-plugin</artifactId>
    <version>3.5</version>
</plugin>
```

Creando un objetivo

Las metas se implementan creando un MOJO. Este es un archivo de clase anotado con anotaciones de `maven-plugin-annotations` .

```
@Mojo(name = "hello")
public final class HelloWorldMojo extends AbstractMojo {

    public void execute() throws MojoExecutionException, MojoFailureException {
         getLog().info("Hello world");
    }
}
```

Usando la configuración del plugin

Los complementos se pueden configurar anotando los campos con `@Parameter` . El MOJO luego se inyecta con la configuración.

```
@Mojo(name = "greet")
public final class GreetMojo extends AbstractMojo {

    @Parameter(required = true)
    public String name;

    public void execute() throws MojoExecutionException, MojoFailureException {
        getLog().info("Hello " + name);
    }
}
```

El parámetro `name` se puede configurar en el POM:

```
<plugin>
    <groupId>com.mattunderscore</groupId>
    <artifactId>hello-world-plugin</artifactId>
    <version>1.0-SNAPSHOT</version>
    <configuration>
        <name>Matt</name>
    </configuration>
</plugin>
```

Si el objetivo de `greet` se ejecuta como un objetivo independiente, el parámetro de `name` se puede definir como propiedad en la línea de comando:

```
mvn <plugin name>:greet -Dname=Geri
```

Accediendo a la información del proyecto.

El complemento puede, entre otros, acceder a la información sobre el proyecto Maven actual que

se está construyendo.

```
@Mojo(name = "project")
public final class ProjectNameMojo extends AbstractMojo {

    @Parameter(defaultValue = "${project}", readonly = true, required = true)
    private MavenProject project;

    public void execute() throws MojoExecutionException, MojoFailureException {
        getLog().info("Hello, this is " + project.getName());
    }
}
```

El ejemplo anterior imprimirá en la consola el nombre del proyecto Maven en el que se ejecuta, que se especifica en el elemento `<project>/<name>` de su POM.

La clase `MavenProject` utilizada en el complemento requiere una dependencia de `maven-core` con un alcance de `compile` (predeterminado) en la POM del complemento:

```
<dependency>
    <groupId>org.apache.maven</groupId>
    <artifactId>maven-core</artifactId>
    <version>3.3.9</version>
</dependency>
```

Además, el uso de anotaciones requiere la siguiente dependencia en el POM del complemento:

```
<dependency>
  <groupId>org.apache.maven.plugin-tools</groupId>
  <artifactId>maven-plugin-annotations</artifactId>
  <version>3.5</version>
  <scope>provided</scope> <!-- annotations are needed only to build the plugin -->
</dependency>
```

Declara una fase por defecto para un gol.

```
@Mojo(name = "hi", defaultPhase = LifecyclePhase.COMPILE)
```

Obtener el directorio de compilación como un archivo

```
@Parameter(defaultValue = "${project.build.directory}")
private File buildDirectory;
```

Lea Crear un complemento de Maven en línea: https://riptutorial.com/es/maven/topic/8635/crear-un-complemento-de-maven

Capítulo 6: Genere informes FIXME / TODO utilizando el complemento taglist-maven-plugin

Introducción

Este es un pequeño fragmento de código (xml) para resaltar cómo usar el taglist-maven-plugin para generar informes personalizados (de TODO, FIXME work ...)

Examples

pom.xml para generar un informe FIXME

```
<build>
    <plugins>
        <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>project-info-reports</artifactId>
                <version>2.9</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>index</goal>
                        </goals>
                        <phase>site</phase>
                    </execution>
                </executions>
            </plugin>
    </plugins>
</build>

<reporting>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-project-info-reports-plugin</artifactId>
                <version>2.9</version>
                <reportSets>
                    <reportSet>
                    <reports>
                        <report>index</report>
                        <report>issue-tracking</report>
                    </reports>
                    </reportSet>
                </reportSets>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>taglist-maven-plugin</artifactId>
                <version>2.4</version>
                <configuration>
```

```
                <tagListOptions>
                    <tagClasse>
                        <displayName>FIXME Work</displayName>
                        <tags>
                            <tag>
                                <matchString>FIXME</matchString>
                                <matchType>ignoreCase</matchType>
                            </tag>
                            <tag>
                                <matchString>@fixme</matchString>
                                <matchType>ignoreCase</matchType>
                            </tag>
                        </tags>
                    </tagClasse>
                    <tagClasse>
                        <displayName>TODO Work</displayName>
                        <tags>
                            <tag>
                                <matchString>TODO</matchString>
                                <matchType>ignoreCase</matchType>
                            </tag>
                            <tag>
                                <matchString>@todo</matchString>
                                <matchType>ignoreCase</matchType>
                            </tag>
                        </tags>
                    </tagClasse>
                </tagListOptions>
            </configuration>
        </plugin>
    </plugins>
</reporting>
```

Entonces corre

```
mvn clean site:site
```

Lea Genere informes FIXME / TODO utilizando el complemento taglist-maven-plugin en línea: https://riptutorial.com/es/maven/topic/10110/genere-informes-fixme---todo-utilizando-el-complemento-taglist-maven-plugin

Capítulo 7: Integración de eclipse

Examples

Instalar Maven en Eclipse

Puede aprovechar las potentes funciones de Apache Maven en Eclipse instalando la función M2Eclipse . Siga estos pasos para instalar Maven en Eclipse:

1. Abra Eclipse y seleccione *Ayuda* → *Instalar nuevo software ...*
2. En el cuadro de diálogo abierto, seleccione el botón `Agregar ...` para agregar un nuevo repositorio.
3. Rellene el formulario con la siguiente información y confirme con `OK` :

   Nombre: `M2Eclipse`

   Ubicación: `http://download.eclipse.org/technology/m2e/releases`
4. Una vez que finalice la *opción Pendiente ...* , `seleccione Todo` y seleccione `Siguiente` .
5. ⊙ `acepte los términos del acuerdo de licencia y seleccione Finalizar`.
6. Al final de la instalación, se le pedirá que reinicie Eclipse. Seleccione `Sí` para realizar el reinicio.

Compruebe si Eclipse ya tiene instalado M2Eclipse Maven.

Vaya a *Ayuda* → *Acerca de Eclipse* → Compruebe si la *función m2e* está allí: .

Configurar una instalación personalizada de Maven en Eclipse.

Eclipse proporcionaría su propio entorno Maven integrado fuera de la caja, que no se recomienda cuando se debe usar una determinada versión de Maven o se debe realizar una configuración adicional (proxy, espejos, etc.): es decir, para tener un control total sobre qué entorno Maven sería utilizado por el IDE.

- Seleccione *Ventana* → *Preferencias* → *Maven* → *Instalaciones*
- Seleccione `Agregar ..` para agregar una instalación Maven personalizada / local
- Suministre la información necesaria y seleccione `Finalizar` :

  Instalación en casa: `... your Maven home ... Directorio ...`

  Nombre de la instalación: `Apache Maven xyz`

- ☑ selecciónelo como predeterminado (en lugar de la versión *EMBEDDED* predeterminada) y confirme con `OK` .

Lea Integración de eclipse en línea: https://riptutorial.com/es/maven/topic/2315/integracion-de-eclipse

Capítulo 8: Maven instalar en ventana

Introducción

Cómo instalar Maven en la ventana 7.

Observaciones

Cómo instalar Maven en la ventana 7 pasos:

1. descargue el formulario de Maven https://maven.apache.org/download.cgi (sitio web de poffice) 2.unzip la carpeta binaria de Maven y guárdelo en cualquier floder (bueno: guárdelo en los archivos de programa en la unidad c)
2. Verifique el indicador de valor del valor de la variable de entorno y escriba echo% java_home% para que muestre la ruta del jdk como: C: \ Archivos de programa \ Java \ jdk1.8.0_102 si no se muestra, establezca la variable de entorno java_home

Establezca las variables de entorno usando el sistema m2_home: establezca la ruta de la carpeta donde está almacenado maven_home: igual que arriba configure la ruta Agregando a PATH: Agregue el directorio bin de la distribución desempaquetada a la ruta de la variable de entorno PATH del usuario:% m2_home% \ bin

Para verificar, abra el símbolo del sistema y escriba mvn -version debería mostrar este mensaje Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T04: 57: 37-07: 00) Maven home: /opt/apache-maven-3.3.3 Versión de Java: 1.8.0_45, proveedor: Oracle Corporation Página de inicio de Java: /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/jre Configuración regional predeterminada: en_US, codificación de la plataforma: UTF-8 Nombre del SO: mac os x ", versión:" 10.8.5 ", arco:" x86_64 ", familia:" mac

en algún momento no se mostrará porque la carpeta mvn no se ejecuta con el acceso de administrador hace que se ejecute como administrador

Examples

instalando

Compruebe el valor de la variable de entorno, por ejemplo, echo% JAVA_HOME% C: \ Archivos de programa \ Java \ jdk1.7.0_51

Lea Maven instalar en ventana en línea: https://riptutorial.com/es/maven/topic/10813/maven-instalar-en-ventana

Capítulo 9: Maven Surefire Plugin

Sintaxis

- prueba mvn
- mvn -Dtest = com.example.package.ExampleTest test

Examples

Probando una clase Java con JUnit y el complemento Maven Surefire

El complemento Maven Surefire se ejecuta durante la fase de prueba del proceso de construcción de Maven o cuando la `test` se especifica como un objetivo de Maven. La siguiente estructura de directorio y el archivo `pom.xml` mínimo configurarán a Maven para ejecutar una prueba.

Estructura de directorio dentro del directorio raíz del proyecto:

```
— project_root
  ├— pom.xml
  ├— src
  |  ├— main
  |  |  └— java
  |  └— test
  |     └— java
  └— target
     └— ...
```

Contenido de `pom.xml` :

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>company-app</artifactId>
  <version>0.0.1</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Cree un archivo llamado `PlusTenTest.java` con el siguiente contenido en el directorio `src/test/java/com/example/app` :

```
package com.example.app;

import static org.junit.Assert.assertEquals;
```

```
import org.junit.Test;

public class PlusTenTest {

    @Test
    public void incrementTest() {
        int result = PlusTen.increment(10);
        assertEquals("PlusTen.increment(10) result", 20, result);
    }
}
```

La anotación `@Test` le dice a JUnit que debe ejecutar `incrementTest()` como prueba durante la fase de `test` del proceso de construcción de Maven. Ahora cree `PlusTen.java` en `src/main/java/com/example/app` :

```
package com.example.app;

public class PlusTen {
    public static int increment(int value) {
        return value;
    }
}
```

Ejecute la prueba abriendo un indicador de comando, navegando al directorio raíz del proyecto e invocando el siguiente comando:

```
mvn -Dtest=com.example.app.PlusTenTest test
```

Maven compilará el programa y ejecutará el método de prueba `incrementTest()` en `PlusTenTest` . La prueba fallará con el siguiente error:

```
...
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.005 sec <<< FAILURE! - in
com.example.app.PlusTenTest
incrementTest(com.example.app.PlusTenTest)  Time elapsed: 0.004 sec  <<< FAILURE!
java.lang.AssertionError: PlusTen.increment(10) result expected:<20> but was:<10>
at org.junit.Assert.fail(Assert.java:88)
at org.junit.Assert.failNotEquals(Assert.java:743)
at org.junit.Assert.assertEquals(Assert.java:118)
at org.junit.Assert.assertEquals(Assert.java:555)
at com.example.app.PlusTenTest.incrementTest(PlusTenTest.java:12)


Results :

Failed tests:
  PlusTenTest.incrementTest:12 PlusTen.increment(10) result expected:<20> but was:<10>

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

[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.749 s
[INFO] Finished at: 2016-09-02T20:50:42-05:00
[INFO] Final Memory: 14M/209M
```

```
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.19.1:test
(default-test) on project app: There are test failures.
...
```

El complemento Maven Surefire crea un directorio `/target/surefire-reports/` en el directorio de su proyecto que contiene los archivos `com.example.app.PlusTenTest.txt` y `TEST-com.example.app.PlusTenTest.xml` que contienen los detalles de error del principio de la salida anterior.

Siguiendo el patrón de desarrollo basado en pruebas, modifique `PlusTen.java` para que el método `increments()` funcione correctamente:

```
package com.example.app;

public class PlusTen {
    public static int increment(int value) {
        return value + 10;
    }
}
```

Invoque el comando de nuevo:

```
mvn -Dtest=com.example.app.PlusTenTest test
```

La prueba pasa:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.example.app.PlusTenTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.028 sec

Results :

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

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.753 s
[INFO] Finished at: 2016-09-02T20:55:42-05:00
[INFO] Final Memory: 17M/322M
[INFO] ------------------------------------------------------------------------
```

¡Felicidades! Ha probado una clase Java utilizando JUnit y el complemento Maven Surefire.

Lea Maven Surefire Plugin en línea: https://riptutorial.com/es/maven/topic/5876/maven-surefire-plugin

Capítulo 10: Plugin EAR Maven

Introducción

A continuación se muestra una configuración de ejemplo para un complemento básico de Ear Maven para empaquetar artefactos .war y .jar

Examples

Una configuración básica de EAR

```
<dependencies>
    <dependency>
        <groupId>{ejbModuleGroupId}</groupId>
        <artifactId>{ejbModuleArtifactId}</artifactId>
        <version>{ejbModuleVersion}</version>
        <type>ejb</type>
    </dependency>
    <dependency>
        <groupId>{webModuleGroupId}</groupId>
        <artifactId>{webModuleArtifactId}</artifactId>
        <version>{webModuleVersion}</version>
        <type>war</type>
    </dependency>
</depencencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-ear-plugin</artifactId>
                <version>2.9.1</version>
                <configuration>
                    <version>1.4</version><!-- application.xml verion -->
                    <modules>
                        <ejbModule>
                            <groupId>{ejbModuleGroupId}</groupId>
                            <artifactId>{ejbModuleArtifactId}</artifactId>
                        </ejbModule>
                        <webModule>
                            <groupId>{webModuleGroupId}</groupId>
                            <artifactId>{webModuleArtifactId}</artifactId>
                            <contextRoot>/custom-context-root</contextRoot>
                        </webModule>
                    </modules>
                </configuration>
        </plugin>
    </plugins>
</build>
```

Una vez compilada, `mvn clean install` , genera un artefacto *.ear* en el directorio de *destino que* contiene tanto el ejb (.jar) como el módulo web, junto con el archivo de descripción JEE de *application.xml* .

Lea Plugin EAR Maven en línea: https://riptutorial.com/es/maven/topic/10111/plugin-ear-maven

Capítulo 11: Plugin Maven Tomcat

Examples

Iniciar Tomcat utilizando el complemento de Maven.

En el ejemplo, iniciaremos Tomcat 7 utilizando el complemento Maven, opcionalmente agregaremos la protección de usuario / contraseña para el punto final REST. También añadiendo característica de la construcción de la guerra.

Agregue la siguiente sección en la sección de plugin de pom para tomcat

```
             <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                    <url>http://localhost:8090/manager</url>
                    <server>localhost</server>
                    <port>8191</port>
                    <path>/${project.build.finalName}</path>
                    <tomcatUsers>src/main/tomcatconf/tomcat-users.xml</tomcatUsers>
                </configuration>
            </plugin>
```

Asegúrese de que se haya agregado el complemento de guerra maven y web.xml esté presente en la ubicación /src/main/webapp/WEB-INF/web.xml. A continuación se muestra un ejemplo de plugin de guerra.

```
<plugin>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.3</version>
            </plugin>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                    <webResources>
                        <resource>
                            <!-- this is relative to the pom.xml directory -->
                            <directory>/src/main/webapp/WEB-INF/web.xml</directory>
                        </resource>
                    </webResources>
                </configuration>
            </plugin>
```

Opcionalmente, agregue tomcat-users.xml a la ubicación src / main / tomcatconf. Se copiará automáticamente cuando se inicie Tomcat.

```
<tomcat-users>
```

```
    <user name="user" password="password" roles="admin" />
</tomcat-users>
```

Opcionalmente, agregue la siguiente entrada en web.xml para proteger la URL de REST.

```
    <!-- tomcat user -->
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Wildcard means whole app requires authentication</web-resource-
name>
            <url-pattern>/helloworld/*</url-pattern>
            <http-method>GET</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>admin</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <login-config>
        <auth-method>BASIC</auth-method>
    </login-config>
```

Crear nueva construcción maven de eclipse. Seleccione el proyecto de guerra y en la sección de Objetivos agregue el siguiente comando.

```
tomcat7:run
```

Verás el mensaje.

[INFO] --- tomcat7-maven-plugin: 2.2: run (default-cli) @ web-service-ldap2 --- [INFO] Ejecutando war en http: // localhost: 8191 /

Lea Plugin Maven Tomcat en línea: https://riptutorial.com/es/maven/topic/6292/plugin-maven-tomcat

Capítulo 12: POM - Modelo de objeto de proyecto

Examples

Estructura de pom

El modelo de objetos del proyecto es la unidad básica de Maven y define la estructura del proyecto, las dependencias, etc.

Los siguientes son muy mínimos para crear un POM:

- raíz del `project`
- `modelVersion` - se debe establecer en `4.0.0`
- `groupId` - el ID del grupo del proyecto
- `artifactId` - el ID del artefacto (proyecto)
- `version` : la versión del artefacto en el grupo especificado

`groupId` , `artifactId` y `version` se llaman *coordenadas de Maven* y a veces se abrevian con *GAV* . Identifican de forma única el artefacto resultante de un proyecto en un repositorio de Maven (y *deberían* hacerlo en todo el universo).

Una muestra mínima de POM se ve así:

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.sample</groupId>
  <artifactId>sample-app</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

Herencia POM

La herencia es el activo más importante de la POM, donde lo siguiente se puede administrar desde la súper POM a la POM secundaria.

- dependencias
- desarrolladores y colaboradores
- listas de complementos (incluyendo informes)
- ejecuciones de plugin con identificadores coincidentes
- configuración de plugin

Lo siguiente habilita la herencia.

```
  <parent>
```

```
    <groupId>com.sample</groupId>
    <artifactId>sample-app-parent</artifactId>
    <version>1.0.0</version>
  </parent>
```

La estructura de POM se parece a

```
<project>
  <parent>
    <groupId>com.sample</groupId>
    <artifactId>sample-app-parent</artifactId>
    <version>1.0.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.sample</groupId>
  <artifactId>sample-app</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

Agregación POM

Los módulos de un proyecto de múltiples módulos se agregan desde una estructura jerárquica.

El empaque de la raíz `pom` debe verse como:

```
<packaging>pom</packaging>
```

La siguiente será la estructura de directorios del proyecto:

```
 |-- sample-app
  \   `-- pom.xml
   |-- sample-module-1
   |   `-- pom.xml
   |-- sample-module-2
   |   `-- pom.xml
```

POM raíz:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.sample</groupId>
  <artifactId>sample-app</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>sample-module-1</module>
    <module>sample-module-2</module>
  </modules>
  <dependencyManagement>
    ...
  </dependencyManagement>
</project>
```

Lea POM - Modelo de objeto de proyecto en línea:
https://riptutorial.com/es/maven/topic/2310/pom---modelo-de-objeto-de-proyecto

Capítulo 13: Realizar un lanzamiento

Introducción

El complemento estándar de Maven utilizado por un Proceso de publicación es el complemento de liberación de maven; la configuración de este complemento es mínima:

SCM en el Maven pom: El proceso de lanzamiento interactuará con el control de origen del proyecto; esto significa que debemos definir el elemento "scm" en nuestro pom.xml. El elemento "scm" para una versión de lanzamiento debe contener suficiente información para Echa un vistazo a la etiqueta que se creó para esta versión.

Observaciones

Nota: asegúrese de usar el complemento de versión 2.5 o posterior de Maven para evitar problemas relacionados con Maven. El proceso de lanzamiento

```
mvn release:clean
```

El comando anterior realizará lo siguiente: eliminar el descriptor de la versión (release.properties) eliminar cualquier archivo POM de respaldo

```
mvn release:prepare
```

La siguiente parte del proceso de lanzamiento es Preparar el lanzamiento; esto hará: realizar algunas comprobaciones: no debe haber cambios no confirmados y el proyecto no debe depender de que las dependencias de SNAPSHOT cambien la versión del proyecto en el archivo pom a un número de versión completo (elimine el sufijo SNAPSHOT), en nuestro ejemplo - 0.0.1 ejecute el banco de pruebas del proyecto confirme y presione los cambios cree la etiqueta fuera de este código versionado no SNAPSHOT aumente la versión del proyecto en el pom - en nuestro ejemplo - 0.0.2-SNAPSHOT commit y empuje los cambios

```
mvn release:perform
```

La última parte del proceso de lanzamiento es realizar el lanzamiento; esto será: desproteger la etiqueta de lanzamiento de SCM compilar e implementar el código liberado Este segundo paso del proceso se basa en el resultado del paso Preparar: el release.properties.

Examples

POM.xml para realizar el lanzamiento al repositorio Nexus

```
<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>org.codezarvis.artifactory</groupId>
<artifactId>nexusrelease</artifactId>
<version>0.0.5-SNAPSHOT</version>
<packaging>jar</packaging>

<name>nexusrelease</name>
<url>http://maven.apache.org</url>

<scm>
<connection>scm:git:git@github.com:isudarshan/nexuspractice.git</connection>
<url>scm:git:git@github.com:isudarshan/nexuspractice.git</url>
<developerConnection>scm:git:git@github.com:isudarshan/nexuspractice.git</developerConnection>
<tag>HEAD</tag>
</scm>

<distributionManagement>
<!-- Publish the versioned snapshot here -->
<repository>
<id>codezarvis</id>
<name>codezarvis-nexus</name>
<url>http://localhost:8080/nexus/content/repositories/releases</url>
</repository>

<!-- Publish the versioned releases here -->
<snapshotRepository>
<id>codezarvis</id>
<name>codezarvis-nexus</name>
<url>http://localhost:8080/nexus/content/repositories/snapshots</url>
</snapshotRepository>
</distributionManagement>

<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>

<build>
<pluginManagement>
<plugins>
<plugin>
<artifactId>maven-release-plugin</artifactId>
<version>2.5.2</version>
<executions>
<execution>
<id>default</id>
<goals>
<goal>perform</goal>
</goals>
<configuration>
```

```
<pomFileName>${project.name}/pom.xml</pomFileName>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</pluginManagement>
</build>
</project>
```

Lea Realizar un lanzamiento en línea: https://riptutorial.com/es/maven/topic/9680/realizar-un-lanzamiento

Creditos

S. No	Capítulos	Contributors
1	Empezando con Apache Maven	A_Di-Matteo, Adonis, Community, darkend, Nate Vaughan, ngreen, Ray, Stephen Leppik, tobybot11, Tunaki
2	Acceso a las informaciones de Maven en código.	Derlin
3	Ciclo de construcción de Maven	Gerold Broser, isudarsan
4	Complemento de montaje de Maven	Jean-Rémy Revy, Tunaki, wallenborn, zygimantus
5	Crear un complemento de Maven	Gerold Broser, Matt Champion, Tunaki, Vince
6	Genere informes FIXME / TODO utilizando el complemento taglist-maven-plugin	Mahieddine M. Ichir
7	Integración de eclipse	A_Di-Matteo, Gerold Broser, karel, kartik, Radouane ROUFID
8	Maven instalar en ventana	shashigura
9	Maven Surefire Plugin	Gerold Broser, Nate Vaughan
10	Plugin EAR Maven	Mahieddine M. Ichir
11	Plugin Maven Tomcat	kartik
12	POM - Modelo de objeto de proyecto	Gerold Broser, JF Meier, Radouane ROUFID, VinayVeluri, Vince
13	Realizar un	isudarsan

	lanzamiento	