Maven

Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a **Apache Ant** (y en menor medida a PEAR de PHP y CPAN de Perl), pero tiene un modelo de configuración de construcción más simple, basado en un formato XML.

Maven utiliza un **Project Object Model (POM)** para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Las partes del **ciclo de vida** principal del proyecto Maven son:

1. `compile` : Genera los ficheros .class compilando los fuentes .java
2. `test` : Ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.
3. `package` : Genera el fichero .jar con los .class compilados
4. `install` : Copia el fichero .jar a un directorio de nuestro ordenador donde maven deja todos los .jar. De esta forma esos .jar pueden utilizarse en otros proyectos maven en el mismo ordenador.
5. `deploy` : Copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a ese servidor remoto.

Cuando se ejecuta cualquiera de los comandos maven, por ejemplo, si ejecutamos `mvn install`, maven irá verificando todas las fases del ciclo de vida desde la primera hasta la del comando, ejecutando solo aquellas que no se hayan ejecutado previamente.

También existen algunas metas que están fuera del ciclo de vida que pueden ser llamadas, pero Maven asume que estas metas no son parte del ciclo de vida por defecto (no tienen que ser siempre realizadas). Estas metas son:

1. `clean` → Elimina todos los .class y .jar generados. Después de este comando se puede comenzar un compilado desde cero.
2. `assembly` → Genera un fichero .zip con todo lo necesario para instalar nuestro programa java. Se debe configurar previamente en un fichero xml qué se debe incluir en ese zip.
3. `site` → Genera un sitio web con la información de nuestro proyecto. Dicha información debe escribirse en el fichero pom.xml y ficheros .apt separados.
4. `site-deploy` → Sube el sitio web al servidor que hayamos configurado.

Pero estas metas pueden ser añadidas al ciclo de vida a través del **Project Object Model (POM)**.

Más información sobre el **Project Object Model (POM)**.

Sintaxis

```
# Sintaxis de Maven
$ mvn [options] [<goal(s)>] [<phase(s)>]
```

Archetypes

> ⚠ Si se usa una terminal PowerShell, hay que usar las **comillas dobles** en los parámetros.

Maven tiene definidos una serie de *archetypes* para generar proyectos de distinta índole con una estructura de ficheros concreta.

Podemos ejecutar el comando de forma completa indicando toda la información necesaria para que Maven genere el proyecto o de forma interactiva para que Maven solicite la información directamente en la consola:

```
# Crear un proyecto Java(jar)
$ mvn archetype:generate "-DgroupId=org.yourcompany.project" "-DartifactId=some-project" "-Dversion=1.0-SNAPSHOT"

# Crear un proyecto web(war)
$ mvn archetype:generate "-DgroupId=org.yourcompany.project" "-DartifactId=application" "-Dversion=1.0-SNAPSHOT" "-Darch

# Crear un archetype a partir de un proyecto existente
$ mvn archetype:create-from-project
```

Fases del ciclo de vida de construcción de un proyecto en Maven

En Maven podemos ejecutar un *goal* como por ejemplo `mvn archetype:generate` o una fase del ciclo de vida de construcción de un proyecto como puede ser `mvn package`.

Los ciclos de vida de construcción y sus fases en orden son:

1. `Clean lifecycle` - pre-clean, clean, post-clean
2. `Default lifecycle` - validate, initialize, generate-sources, process-sources, generate-resources, process-resources, compile, process-classes, generate-test-sources, process-test-sources, generate-test-resources, process-test-resources, test-compile, process-test-classes, test, prepare-package, package, pre-integration-test, integration-test, post-integration-test, verify, install, deploy
3. `Site lifecycle` - pre-site, site, post-site, site-deploy

Cuando indicamos a Maven que ejecute una fase, como por ejemplo `mvn compile`, Maven ejecutará cada fase previa en orden de forma secuencial incluida la fase usada en el comando.

Por tanto, si miramos la lista veremos que al indicarle `mvn compile` Maven ejecutará en este orden las fases de *'validate'*, *'initialize'*, '*generate-sources*', '*process-sources*', '*generate-resources*', '*process-resources*' y '*compile*'.

Maven **no ejecutará fases de otros ciclos de vida si no se indica explícitamente**. Es decir, si ejecutamos el comando `mvn compile` del ejemplo anterior, Maven no ejecutará ninguna fase del ciclo de vida de *'clean'*. Por tanto, para que Maven ejecute las fases del grupo *'clean'* y las fases del grupo *'default'* debemos indicarlo con el comando `mvn clean compile`.

Las fases con nombres con prefijos como *pre-, post-* o *process-** son fases cuya función no es ser llamada directamente ya que son fases que producen resultados intermedios para la fase siguiente. Por tanto no son útiles como fase en sí misma y deben evitarse.

Las **fases en orden más importantes** del ciclo de vida de construcción de la mayoría de proyectos son:

```
# Eliminar el directorio `target` que contiene las clases compiladas
$ mvn clean

# Validar si la definición y la estructura del proyecto es correcta
$ mvn validate

# Compilar el código fuente del proyecto
$ mvn compile
```

```
# Ejecutar los tests del correspondiente framework de test unitarios
$ mvn test

# Empaquetar el código compilado en un formato distribuible como JAR o WAR
mvn package

# Ejecutar las validaciones para verificar si el empaquetado es válido y cumple los criterios de calidad
$ mvn verify

# Instalar el empaquetado en el repositorio local (`user_dir\.m2\`) para ser usado como dependencia en otros proyectos l
$ mvn install

# Copia el empaquetado final en un repositorio remoto para ser usado en otros proyectos
$ mvn deploy
```

Para evitar que se ejecutan los tests previos al invocar fases posteriores, podemos usar la opción `-DskipTests=true` como por ejemplo `mvn install -DskipTests=true`.

Si queremos evitar que se compilen y que se ejecuten, usaremos la opción `-Dmaven.test.skip=true`.

Con la opción `--offline` le indicamos a Maven que trabaje de forma offline, sin descargar ningún artefacto.

Con la opción `--update-snapshots` forzamos a Maven para que actualice las dependencias de un repositorio remoto.

Podemos indicar el número de hilos que Maven debe utilizar para realizar el trabajo con la opción `-T` como por ejemplo `-T 4` para indicar 4 hilos de ejecución o `-T 2C` para indicar 2 hilos por CPU.

Fases adicionales

```
# Generar un _site_ del proyecto sin el informe de los test
$ mvn site

# Generar un _site_ del proyecto con el informe de los tests
$ mvn test site

# Generar un _site_ del proyecto con el informe de los tests de integración
$ mvn verify site
```

Otros plugins

De forma interna, para Maven los ciclos de vida corresponden a un *'plugin'*. Por ejemplo la fase *'clean'* corresponde al plugin *'maven-clean-plugin'*. Por tanto, se podría invocar como `mvn clean` o usando el *goal* `mvn clean:clean`.

Maven incluye una serie de *'plugins'* que forman parte de su núcleo y que corresponden a las fases vistas anteriormente.

La lista de *plugins* oficiales y no oficiales soportados por Maven se puede consultar aquí. Hay más listas de *plugins* de Maven como por ejemplo MojoHaus.

Calidad del código

```
# Analizar la calidad del código con _Sonar_
$ mvn clean install -DskipTests=true sonar:sonar

# Visualizar la ayuda y los _goals_ disponibles del plugin
$ mvn sonar:help
```

- Más información sobre el plugin

Informes sobre los tests

```
# Ejecutar los tests unitarios y generar un informe sobre la cobertura del código de los tests
$ mvn clean cobertura:cobertura

# Visualizar la ayuda y los _goals_ disponibles del plugin
$ mvn cobertura:help
```

- Más información sobre el plugin

Administrar las versiones de los artefactos del proyecto

```
# Escanea las dependencias del proyecto y genera un informe con las actualizaciones disponibles
$ mvn versions:display-dependency-updates

# Escanea los _plugins_ del proyecto y genera un informe con las actualizaciones disponibles
$ mvn versions:display-plugin-updates
```

- Más información sobre el plugin

Administrar las dependencias del proyecto

```
# Visualizar en formato árbol las dependencias del proyecto
$ mvn dependency:tree

# Analizar y validar las dependencias del proyecto
$ mvn dependency:analyze
```

- Más información sobre el plugin

Ayuda

```
# Visualiza los ajustes de Maven en formato XML
$ mvn help:effective-settings

# Visualiza el fichero POM
$ mvn help:effective-pom "-Dverbose=true"

# Visualiza todos los perfiles
$ mvn help:active-profiles

# Visualiza la información relativa de un _plugin_
$ mvn help:describe "-Dcmd=install"

# Visualiza información relativa del _goal_ de un _plugin_
$ mvn help:describe "-Dcmd=install:install"

# Visuliza la ayuda de un _plugin_
$ mvn compiler:help
$ mvn dependency:help
$ mvn versions:help`

# Visualiza la ayuda en detalle de un _goal_ de un _plugin
$ mvn compiler:help -Dgoal=compile -Ddetail
```

Referencias

- https://maven.apache.org/index.html
- https://maven.apache.org/guides/MavenQuickReferenceCard.pdf
- https://www.baeldung.com/maven-guide
- https://www.sonatype.com/maven-complete-reference

Licencia

Esta obra está bajo una licencia de Creative Commons Reconocimiento-Compartir Igual 4.0 Internacional.