Introducción a Maven

Maven es una herramienta de gestión de proyectos. Se basa en un fichero central, **pom.xml**, donde se define todo lo que necesita tu proyecto. **Maven maneja las dependencias del proyecto, compila, empaqueta y ejecuta los test**. Mediante plugins, permite hacer mucho más, como por ejemplo generar los mapas de Hibernate a partir de una base de datos, desplegar la aplicación, etc...

Lo más útil de Maven en mi opinión, es el manejo de las dependencias. Aún recuerdo cuando empezaba a trabajar en Java. **Tenías que ir bajándote manualmente los jar que necesitabas en tu proyecto y copiarlos manualmente en el classpath**. Era muy tedioso. Con Maven esto se acabó. Solo necesitas definir en tu pom.xml las dependencias que necesitas y **maven las descarga y las añade al classpath**.

Instalando Maven

Maven normalmente ya viene instalado en cualquier distribución popular de linux o al menos, está disponible en los repositorios oficiales de la distribución. En otros sistemas, como MacOSX, tambien está instalado por defecto. Para los que empleáis windows, teneis que descomprimir el fichero zip donde mejor os venga. Luego, teneis que crear la variable de entorno **M2_HOME** apuntando al directorio donde descomprimais maven y **añadir M2_HOME al path del sistema** para poder ejecutar maven desde la consola.

Yo soy amante de la consola, pero si quereis olvidaros de la linea de comandos, podeis instalar el Eclipse y el plugin m2eclipse desde Eclipse Market Place. Este plugin ya contiene Maven, por lo que no necesitáis instalarlo.

La primera vez que ejecutemos maven, **creará un repositorio local en tu disco duro**. En concreto, creará la carpeta **.m2** en la carpeta **home** del usuario. En ella se guardarán todos los artefactos que maneje maven.

Grupos y artefactos

Un artefacto es un componente de software que podemos incluir en un proyecto como dependencia. Normalmente será un **jar**, pero podría ser de otro tipo, como un **war** por ejemplo. Los artefactos pueden tener dependencias entre sí, por lo tanto, si incluimos un artefacto en un proyecto, también obtendremos sus dependencias.

Un grupo es un conjunto de artefactos. Es una manera de organizarlos. Así por ejemplo todos los artefactos de Spring Framewok se encuentran en el grupo **org.springframework**.

Veamos un ejemplo:

```
<dependency>

    <groupid>org.springframework</groupid>

    <artifactid>spring-orm</artifactid>

    <version>3.0.5.RELEASE</version>

    <scope>runtime</scope>

</dependency>
```

Esta es la manera de declarar una dependencia de nuestro proyecto con un artefacto. **Se indica el identificador de grupo, el identificador del artefacto y la versión**.

Scope (alcance)

El *scope* sirve para indicar el **ámbito o alcance de nuestra dependencia** y su transitividad. Hay 6 tipos:

- **compile**: es la que tenemos por defecto sino especificamos scope. Indica que la dependencia es necesaria para compilar. La dependencia además se propaga en los proyectos dependientes.
- **provided**: Es como la anterior, pero esperas que el contenedor ya tenga esa libreria. Un claro ejemplo es cuando desplegamos en un servidor de aplicaciones, que por defecto, tiene bastantes librerías que utilizaremos en el proyecto, así que no necesitamos desplegar la dependencia.
- **runtime**: La dependencia es necesaria en tiempo de ejecución pero no es necesaria para compilar.
- **test**: La dependencia es solo para testing que es una de las fases de compilación con maven. JUnit es un claro ejemplo de esto.
- **system**: Es como provided pero tienes que incluir la dependencia explicitamente. Maven no buscará este artefacto en tu repositorio local. Habrá que especificar la ruta de la dependencia mediante la etiqueta <*systemPath*>

- **import**: este solo se usa en la sección *dependencyManagement*. Lo explicaré en otro artículo sobre transitividad de dependencias.

¿Qué es un Goal?

Un Goal, es lo que hizo Cristiano Ronaldo 53 veces el año pasado. No, hablando en serio, un **Goal no es más que un comando que recibe maven como parámetro** para que haga algo. La sintaxis sería:

```
mvn plugin:comando
```

Maven tiene una arquitectura de plugins, para poder ampliar su funcionalidad, aparte de los que ya trae por defecto.

Ejemplos de goals serían:

- **mvn clean:clean (o mvn clean)**: limpia todas las clases compiladas del proyecto.
- **mvn compile**: compila el proyecto
- **mvn package**: empaqueta el proyecto (si es un proyecto java simple, genera un jar, si es un proyecto web, un war, etc...)
- **mvn install**: instala el artefacto en el repositorio local (/Users/home/.m2)

Un ejemplo de un goal de un plugin externo, sería el plugin de hibernate 3.X:

```
mvn hibernate3:hbm2hbmxml
```

En este caso el goal es **hbm2hbmxml**, que genera los mapas de hibernate a partir de una base de datos dada.

Los plugins son artefactos, y se incluyen en el pom como <plugin>, pero esto es matéria para otro artículo, donde explicaré como configurar el pom.xml.

¿Qué es un arquetype?

Un arquetype, traducido arquetipo, es, **una plantilla**. Cuando creamos un proyecto, como ahora veremos, tenemos que especificar uno. **Un arquetipo crea la estructura del proyecto, el contenido del pom.xml, la estructura de carpetas y los ficheros que incluye por defecto.**

Creando un proyecto

Vamos a crear nuestro primer proyecto Maven. Yo voy a usar eclipse para este ejemplo, pero todos los IDE del mercado lo soportan. Pero como os he comentado, soy amante de la consola, así que me abro un terminal dentro del workspace de eclipse y ejecuto el siguiente comando:

```
mvn archetype:create -DgroupId=com.genbetadev.maven -DartifactId=maven-jar-sample
```

Este comando nos genera la siguiente salida:

```
[INFO] Scanning for projects…

[INFO]

[INFO] ————————————————————————————

[INFO] Building Maven Stub Project (No POM) 1

[INFO] ————————————————————————————

[INFO]

[INFO] -- maven-archetype-plugin:2.0:create (default-cli) @ standalone-pom --

[WARNING] This goal is deprecated. Please use mvn archetype:generate instead

[INFO] Defaulting package to group ID: com.genbetadev.maven

Downloading:             http://repo1.maven.org/maven2/org/apache/maven/archetypes/maven-archetype-
quickstart/maven-metadata.xml

Downloaded:              http://repo1.maven.org/maven2/org/apache/maven/archetypes/maven-archetype-
quickstart/maven-metadata.xml (531 B at 0.9 KB/sec)

[INFO] ————————————————————————————

[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-
quickstart:RELEASE

[INFO] ————————————————————————————

[INFO] Parameter: groupId, Value: com.genbetadev.maven

[INFO] Parameter: packageName, Value: com.genbetadev.maven

[INFO] Parameter: package, Value: com.genbetadev.maven
```

```
[INFO] Parameter: artifactId, Value: maven-jar-sample

[INFO] Parameter: basedir, Value: /Users/mangrar/workspaces/Eclipse Indigo

[INFO] Parameter: version, Value: 1.0-SNAPSHOT

[INFO]   ********************   End   of   debug   info   from   resources   from   generated   POM
************************

[INFO]  project  created  from  Old  (1.x)  Archetype  in  dir:  /Users/mangrar/workspaces/Eclipse
Indigo/maven-jar-sample

[INFO] ------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------

[INFO] Total time: 2.787s

[INFO] Finished at: Mon Jul 25 19:50:09 IST 2011

[INFO] Final Memory: 7M/81M

[INFO] ------------------------------------------
```

Si veis veis la salida, nos dice que *arquetype:create* está en desuso, y en próximas versiones de maven, caerá en el olvido. Aunque aún funciona, **es recomendable usar el nuevo goal generate**:

```
mvn archetype:generate -DgroupId=com.genbetadev.maven -DartifactId=maven-jar-sample
```

La gran diferencia es que es interactivo. Nos mostrará una lista de arquetipos disponibles y nos preguntará cuál de ellos queremos utilizar. Por defecto, usa *maven-archetype-quickstart*. Pulsando enter elegiremos este mismo:

```
112: remote -> maven-archetype-quickstart (An archetype which contains a sample Maven project.)

Choose a number: 112: 112

Choose version:

1: 1.0-alpha-1

2: 1.0-alpha-2

3: 1.0-alpha-3
```

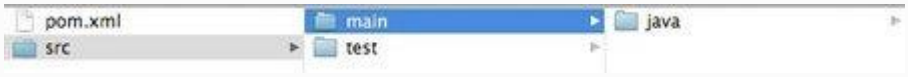
```
4: 1.0-alpha-4

5: 1.0

6: 1.1

Choose a number: 6: 6

[INFO] Using property: groupId = com.genbetadev.maven

[INFO] Using property: artifactId = maven-jar-sample

Define value for property 'version':  1.0-SNAPSHOT: :

[INFO] Using property: package = com.genbetadev.maven

Confirm properties configuration:

groupId: com.genbetadev.maven

artifactId: maven-jar-sample

version: 1.0-SNAPSHOT

package: com.genbetadev.maven

 Y: : Y

[INFO] ----------------------------------------

[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-
quickstart:1.1

[INFO] ----------------------------------------

[INFO] Parameter: groupId, Value: com.genbetadev.maven

[INFO] Parameter: packageName, Value: com.genbetadev.maven

[INFO] Parameter: package, Value: com.genbetadev.maven

[INFO] Parameter: artifactId, Value: maven-jar-sample

[INFO] Parameter: basedir, Value: /Users/mangrar/workspaces/Eclipse Indigo

[INFO] Parameter: version, Value: 1.0-SNAPSHOT

[INFO]   ********************   End   of   debug   info   from   resources   from   generated   POM
**********************
```

```
[INFO] project created from Old (1.x) Archetype in dir: /Users/mangrar/workspaces/Eclipse Indigo/maven-jar-sample

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 1:20.676s

[INFO] Finished at: Mon Jul 25 19:53:35 IST 2011

[INFO] Final Memory: 7M/81M

[INFO] ------------------------------------------------------------------------
```

Ya tenemos nuestro proyecto maven creado. Entremos ahora dentro de la carpeta y observemos su estructura.

Estructura de un proyecto maven

Dependerá del tipo de proyecto con el que trabajemos, pero tienen cosas en común. Echémosle un vistazo:



Tenemos nuestro **pom.xml**, donde se configura el proyecto maven y una carpeta **src** donde se colocan los fuentes. Dentro de src, tenemos dos carpetas: main, donde va todo el código de nuestro proyecto, y test, donde va el código de test. En este caso, como es un proyecto java simple, dentro de main solo tenemos la carpeta java. En ella ya podemos colocar nuestros paquetes y clases java.

En caso de utilizar otros arquetipos, como **maven-archetype-webapp**, tendríamos otras carpetas además de la de java, la de **resources** y **web**.

Compilando el proyecto

Bueno, ya tenemos el proyecto, ahora nos toca compilar. El archetype *maven-archetype-quickstart*, por defecto incluye un paquete y una clase, tanto en el código del proyecto como en el de test. Para el paquete **utiliza la unión del groupId más artifactId que hemos especificado al crear el proyecto**:

src/main/java/com/genbetadev/maven/App.java:

```
package com.genbetadev.maven;

/**

 * Hello world!

 *

 */

public class App

{

    public static void main( String[] args )

    {

        System.out.println( “Hello World!” );

    }

}
```

src/test/java/com/genbetadev/maven/AppTest.java:

```
package com.genbetadev.maven;

import junit.framework.Test;

import junit.framework.TestCase;

import junit.framework.TestSuite;

/**

 * Unit test for simple App.

 */

public class AppTest extends TestCase

{
```

```
    /**
     * Create the test case
     *
     * param testName name of the test case
     */
    public AppTest( String testName )
    {
        super( testName );
    }
    /**
     * return the suite of tests being tested
     */
    public static Test suite()
    {
        return new TestSuite( AppTest.class );
    }
    /**
     * Rigourous Test :-)
     */
    public void testApp()
    {
        assertTrue( true );
    }
}
```

Como podéis ver, el código es bastante trivial. Ahora vamos a compilar:

```
mvn package
```

Este goal hace dos cosas, compila y empaqueta. En este caso, crea un jar. Si solo quisiéramos compilar usaríamos *compile* en vez de *package*.

```
</p><p>Este goal lo que ha hecho es compilar, ejecutar el test de junit y empaquetar como jar. En caso de que el test fallara, no compilaría (se puede evitar con el parámetro -DskipTest).</p>
```

Si ahora navegáis dentro del proyecto, veréis que tenemos una nueva carpeta llamada **target**. En ella es donde maven genera los .class y en este caso, el jar.

Instalando el artefacto

Por último nos queda instalar nuestro artefacto en el repositorio local de maven:

```
mvn install
```

Veamos la salida:

```
[INFO] -- maven-install-plugin:2.3.1:install (default-install) @ maven-jar-sample --

[INFO] Installing /Users/mangrar/workspaces/Eclipse Indigo/maven-jar-sample/target/maven-jar-sample-1.0-SNAPSHOT.jar to /Users/mangrar/.m2/repository/com/genbetadev/maven/maven-jar-sample/1.0-SNAPSHOT/maven-jar-sample-1.0-SNAPSHOT.jar

[INFO] Installing /Users/mangrar/workspaces/Eclipse Indigo/maven-jar-sample/pom.xml to /Users/mangrar/.m2/repository/com/genbetadev/maven/maven-jar-sample/1.0-SNAPSHOT/maven-jar-sample-1.0-SNAPSHOT.pom
```

Este goal lo que hace es copiar tanto el jar como el pom a nuestro repositorio:

/Users/mangrar/.m2/repository/com/genbetadev/maven/maven-jar-sample/1.0-SNAPSHOT/maven-jar-sample-1.0-SNAPSHOT.jar
/Users/mangrar/.m2/repository/com/genbetadev/maven/maven-jar-sample/1.0-SNAPSHOT/maven-jar-sample-1.0-SNAPSHOT.pom

La ruta coincide con el grupo que especificamos al crear el proyecto, concatenando el nombre del artefacto y la versión:

/com/genbetadev/maven/maven-jar-sample/1.0-SNAPSHOT/

¿De que nos sirve esto? pues la gran utilidad es que ahora **podemos incluir este proyecto como dependencia en otro proyecto maven**, mejorando la potabilidad y modularidad de nuestros proyectos. Si estáis siguiendo mis artículos de introducción a Spring Framework, lo comprenderéis mejor. Ayer, en el segundo capítulo, creábamos la base de la capa de acceso a datos de una aplicación mediante hibernate. Es una buena idea, tenerlo como un proyecto y poder importarlo desde cualquier otro con solo incluir la dependencia en el pom de cualquier otro proyecto.

Existen además herramientas como Artifactory que nos permiten crear y gestionar repositorios maven en la red. Muy útil para cualquier equipo de desarrollo, ya que distintos programadores pueden acceder a todos los artefactos creados en una empresa de desarrollo e incluir lo que necesiten en sus proyectos.

Por último señalar que el Goal install depende de la compilación y empaquetado. Por lo tanto, si vamos a ejecutar este goal, no hace falta ejecutar los otros dos.

Integración en el IDE

Los proyectos maven, se pueden importar en los IDE más conocidos del mercado. Yo uso eclipse y para hacerlo hay que ejecutar el siguiente comando dentro de la carpeta del proyecto:

```
mvn eclipse:eclipse
```

Este Goal crea los ficheros necesarios para que Eclipse pueda importar el proyecto. Lo único que tenemos que hacer es **File -> import -> Existing projects into workspace** y seleccionar la carpeta correspondiente a nuestro proyecto. Si tenemos instalado el plugin m2eclipse, no hace falta ejecutar ningún Goal de maven. Simplemente abrimos **File -> import -> Existing maven projects** y ya lo tenemos en nuestro IDE:

Para otros IDE, el procedimiento es similar. Por ejemplo para Intellij IDEA es:

```
mvn idea:idea
```