Maven Parte 1

Conceptos generales

People matter, results count.

Introducción

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 mas, como por ejemplo generar los mapas de Hibernate a partir de una base de datos, desplegar la aplicación, etc…**

"Lo mas útil de Maven en mi opinión, es el manejo de las dependencias. Aun 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 esta instalado por defecto.
Para usuarios de Windows, hay que descomprimir el fichero zip donde mejor les sea conveniente. Luego necesitamos que crear la variable de entorno M2_HOME **apuntando al directorio donde descomprimimos maven y** añadir M2_HOME al path del sistema **para poder ejecutar maven desde la consola.**
Maven puede ser usado desde la consola, aunque también existe un plugin para eclipse (m2Eclipse)
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

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**.**

Grupos y Artefactos

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**. EJ:**

```
<dependency>
        <groupid>org.springframework</groupid>
        <artifactid>spring-orm</artifactid>
        <version>3.0.5.RELEASE</version>
        <scope>runtime</scope>
</dependency>
```

Scope (alcance)

Scope (Alcance)

El *scope* sirve para indicar el 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*.**

Goals

¿Qué es un Goal?

Un Goal no es mas que un comando que recibe maven como parámetro **para que haga algo.**

La sintaxis sería:

mvn plugin:comando

Goals

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:hbm2hbmxmlEn **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 curso.

Archetypes

Archetypes

Un archetype, 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 con Maven…

Creando un proyecto con Maven…

Vamos a crear nuestro primer proyecto Maven.

Si quisieramos hacerlo desde consola:

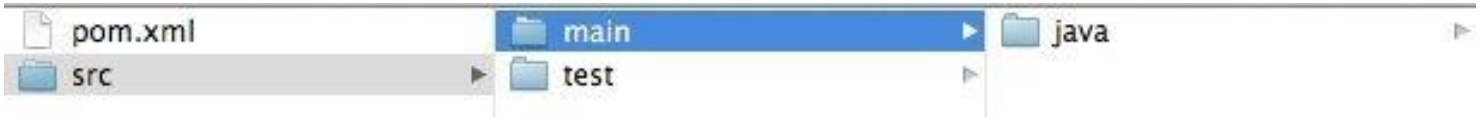
mvn archetype:generate -DgroupId=com.capgemini.escuelita -DartifactId=maven-jar-sample

En este caso vamos a ver un ejemplo desde el plugin de eclipse (m2Eclipse)

Estructura de proyectos

Estructura de proyectos 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 un proyecto…

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/capgemini/escuelita/App.java:

Compilando el proyecto…

```java
/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( “Hello World!” );
    }
}
```

Compilando el proyecto…

```java
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 );
    }
}
```

Compilando el proyecto

Como podemos ver, el código del ejemplo 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*.**

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

Instalando el Artefacto

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/afichera/workspaces/excuelita/maven-jar-sample/target/maven-jar-sample-1.0-SNAPSHOT.jar to /Users/afichera/.m2/repository/com/capgemini/escuelita/maven-jar-sample/1.0-SNAPSHOT/maven-jar-sample-1.0-SNAPSHOT.jar [INFO] Installing /Users/afichera/workspaces/escuelita/maven-jar-sample/pom.xml to /Users/afichera/.m2/repository/com/capgemini/escuelita/maven-jar-sample/1.0-SNAPSHOT/maven-jar-sample-1.0-SNAPSHOT.pom***

Instalando el artefacto

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

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

¿De que nos sirve esto?

La gran utilidad es que ahora podemos incluir este proyecto como dependencia en otro proyecto maven**, mejorando la potabilidad y modularidad de nuestros proyectos**

Artifactory… ¿que es?

Artifactory, ¿Que es?

Artifactory es una herramienta que nos permite 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.

Integración con el IDE

Integración con el IDE

Los proyectos maven, se pueden importar en los IDE mas conocidos del mercado. En eclipse 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.**