MAVEN

by NAIF (SIMASOFT)

INTRODUCIÓN

- Maven es una herramienta de gestión de proyectos. yyy
- Se basa en un fichero central, pom.xml, donde se define todo lo que necesita el proyecto.
- Maven maneja las dependencias del proyecto, compila, empaqueta y ejecuta los test.
- Mediante plugins, permite hacer mucho mas, como por ejemplo desplegar la aplicación, etc.
- Lo mas útil de Maven, es el manejo de las dependencias, solo se necesita definir en el pom.xml las dependencias que se necesitan y maven las descarga y las añade al classpath.

A Maven project

A Maven project is simply a folder in your filesystem (also known as the project root folder) that contains a file called pom.xml, the XML representation of your Project Object Model (POM); this is the first and most important Maven convention.

This minimal structure allows you to run a mvn command from the project root folder. By default, the mvn command searches for a pom.xml file in the local folder and it stops immediately if it is not able to find it.

By convention, all artifacts generated by the build are delivered in a folder relative to the pom.xml location known as Build Directory (the target by default). Since the target is generated on each build, it is:

- Safe to delete it anytime
- Crucial to ignore it when sharing the project using a Version Control System software

A Maven project defines a packaging, which identifies the main objective of the build, which in turn specifies the artifact that is going to be produced by the invocation of the build.Default JAR (other values are EAR, EJB, RAR, PAR, WAR, and POM).

POM packaging is an exception, since:

- The Maven build does not produce an artifact
- The Maven build considers as the only artifact the main pom.xml file of the Maven project

A POM Maven project can be useful for the following activities:

- Aggregate dependencies (for more information, you can navigate to Lifecycle | Dependency Management)
- Parent POM (for more information, you can navigate to Lifecycle | Multi-module project)

http://maven.apache.org/ref/3.1.1/maven-model-builder/super-pom.html

Creating from an archetype

Now it's time for some action! We are creating a new Java project#containing a Maven build#starting from an archetype, using the maven-archetype-quickstart command (we will read more about archetypes in the next section). You don't need to create any file or folder in advance; Maven will take care of creating the root project folder and the initial content.

- Run the following command:

```
mvn archetype:generate -DgroupId=com.mycompany.demo -DartifactId=myapp -
DarchetypeArtifactId=maven-archetype-quickstart-DinteractiveMode=false
```

As you can see, the command specifies one property for each coordinate that identifies the archetype *http://search.maven.org/#artifactdetails_org.apache.maven.archetypes_maven-archetype-quickstart_1.1_maven-archetype*.

The preceding command will ask you some information regarding the project (groupId, artifactId, version) you're about to create. The rules for Maven artifact coordinates apply; so let's assume the following values:

```
groupId: com.mycompany.demo
artifactId: myapp
version: 1.0-SNAPSHOT
```

If everything goes fine, the end of the output should be similar to the following:

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 12.653s
[INFO] Finished at: Sun Mar 17 16:38:36 CET 2013
[INFO] Final Memory: 7M/265M
[INFO] ------------------------------------------------------------
[source, JAVA]
```

The process will download the needed dependencies into your Maven Local Repository; with a fast connection, you should be done in less than a minute.

The root project folder#with the same name as the artifactId coordinate you've specified earlier#should now be located in the same place where you ran the last mvn command.

Exploring the project structure

Let’s now explore the newly created myapp folder. At the first level, you can see POM represented by the pom.xml file; the src folder contains all other project files.

The project file’s structure will be as follows:

```
myapp
|-- pom.xml
|-- src
|   '-- main
|     '-- java
|       '-- com
|         '-- mycompany
|           '-- demo
|             '-- App.java
|   '-- test
|     '-- java
|       '-- com
|         '-- mycompany
|           '-- demo
|             '-- AppTest.java
```

The components of the structure are explained as follows:

- **pom.xml:** Defines the project’s coordinates, a profile, and some dependencies.
- **src/main/java:** Contains the project’s source code. You can see that a sample App.java file has been created, which contains a Java "Hello World" application.
- **src/test/java:** Contains a JUnit test.

4.1. Exploring POM

The following code snippet shows the reported pom.xml content:

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany.demo</groupId>
    <artifactId>myapp</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>myapp</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
```

```
        </dependency>
    </dependencies>
</project>
```

The pom.xml file that is defined is simple and easy to understand:

- It defines a **<packaging>jar</packaging>** element, therefore we expect to produce a JAR file from this build
- It specifies a dependency to the JUnit library, limiting it to test the scope (scopes are explained in the next section)
- It defines all the coordinates to identify it as an artifact (**groupId**, **artifactId**, and **version**), which are the same that are typed when running mvn archetype:generate:
  - **groupId:** Represents the group, company, team, organization, and/or project
  - **artifactId:** Represents the name of the project (or project module)
  - **version:** Represents the number of the current release; since our version ends with SNAPSHOT, it means that this is an unreleased project (development is in progress)

4.2. Compile, install, and run

From the root project folder, run **mvn install** and check the target folder. Maven generates the following files and folders in the target folder:

- **classes:** Contains the compiled Java sources (.class files) found in src/main/java and also the contents of src/main/resources (in our case it is empty). Used by the resources:resources andcompiler:compile goals.
- **test-classes:** This is the same as classes, but it is applied to src/test/java and src/test/resources. Used by the resources:testResources and compiler:testCompile goals.
- **maven-archiver:** Contains a manifest file of the Maven project; you can read more on the Maven official documentation available at *http://maven.apache.org/shared/maven-archiver/index.html*.
- **surefire-reports:** Contains the Surefire Plugin reports; the folder is empty, since no tests are defined in this project (src/main/test is empty). Created by the surefire:test goal.
- **myapp-1.0-SNAPSHOT.jar:** Contains the contents of the classes folder compressed in a JAR. Created by the jar:jar goal.

The same JAR file has been copied into your Maven Local Repository by the install:install goal, and it is located in **~/repository/com/mycompany/demo/myapp/1.0-SNAPSHOT/myapp-1.0-SNAPSHOT.war.**

- To run a console Java application packaged as a JAR file, the command is as follows:

```
java -jar target/myapp-1.0-SNAPSHOT.jar
```

Unfortunately, we forgot to insert into the JAR archive any information about what the main executable class is, so we get an error message as follows:

```
Failed to load Main-Class manifest attribute from
myapp-1.0-SNAPSHOT.jar
```

To solve this issue, you can use maven-jar-plugin (more information is available at *http://maven.apache.org/shared/maven-archiver/examples/classpath.html* as follows:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>2.4</version>
    <configuration>
        <archive>
            <manifest>
                <mainClass>com.mycompany.demo.App</mainClass>
            </manifest>
        </archive>
    </configuration>
</plugin>
```

Now you can run mvn clean install to (re)create the JAR file containing a complete Java manifest file, with the following line:

```
Main-Class: com.mycompany.demo.App
```

You can now confidently run the application, using the following command:

```
java -jar target/myapp-1.0-SNAPSHOT.jar
```

4.3. Changing POM

By default, Maven uses Java 5, but it is possible to add the Compiler Plugin to tell Maven which JDK version to use for compiling your project.

Insert the following XML snippet just before the **<dependencies>** tag:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.1</version>
```

```
    <configuration>
        <source>1.7</source>
        <target>1.7</target>
    </configuration>
</plugin>
```

This additionally allows you to choose the source and target JDK versions (in this case, it means that the compiler accepts Java Version 1.7 statements).

It could be a good idea to set JUnit to a more recent version, which at this time is 4.11. Just edit the version number within the <dependency> element:

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
</dependency>
```

We don't need to think about the transitive dependencies, as Maven will handle them for us; we just need to change the <version> value and re-run the build.

4.4. Analyzing the console and fixing warnings

The Maven console output is the most authoritative source (along with the effective POM that will be discussed later) to know what a Maven build is doing; it is a very good exercise to read the console output and see what operations are performed by the Maven build.

You should also see one (type of) warning:

```
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered
 resources, i.e. build is platform dependent!
```

This warning is triggered by the resources:resources goal execution and it's informing us that#since we haven't specified an encoding in the build#the plugin will use one of the current platforms, which is a common build portability issue.

To fix the issue, we just need to define a property:

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

Well done! Your final pom.xml file should look like this:

```
<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/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.mycompany.demo</groupId>
    <artifactId>myapp</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>

    <name>myapp JAR application</name>
    <url>http://maven.apache.org</url>

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

    <build>
        <plugins>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>2.4</version>
                <configuration>
                    <archive>
                     <manifest>
                    <mainClass>com.mycompany.demo.App</mainClass>
                     </manifest>
                    </archive>
                </configuration>
            </plugin>

        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

INSTALACION

- Descargar *http://maven.apache.org/download.html*
- Configurar la variable de Entorno: M2_HOME. Ejemplo: M2_HOME=D:\javajdk\maven\3.0.2
- Agregar el Maven al Path del windows. Ejemplos: …; %M2_HOME%\bin
- Probar la instalación abriendo una ventana de D.O.S con cmd y digitar: mvn --version

Installation

Maven is a Java application, and it is distributed for the most adopted platforms; in this section we will cover the installation using the pre-built, binary package downloadable directly from the Apache project page, which is the most advised way to install Apache Maven, as opposed to Rpm, Deb, HomeBrew, MacPorts, and any other operating system-specific installation manager.

6.1. Step 1: Prerequisites

Maven needs minimal resources to run. Just be sure to have at least Java 1.5 installed:

- **Java:** JDK 1.5 or above
- **Memory:** No minimum memory limit, but at least 32 MB for the Maven processes are needed
- **Disk space:**
- No formal minimum limit, but Maven actually needs a few hundred available megabytes of disk

6.2. Step 2: Downloading Maven

Open the browser to *http://maven.apache.org/download.cgi* and follow the instructions; download the ZIP archive for Windows or the tar.gz archive for Unix-like operating systems containing the Apache Maven binaries

6.3. Step 3: Installing Maven

The installation process is almost exactly the same across different platforms, but the command-line syntax differs a bit.

6.3.1. Microsoft Windows

1. Unzip the distribution archive, that is, apache-maven-3.0.5-bin.zip, to (for example) C:\Program Files\Apache. The subdirectory apache-maven-3.0.5 will be created in that folder while uncompressing the archive.
2. Open up the system properties window, select the Advanced tab and the Environment Variables button, then add the M2_HOME variable as a user variable with the value C:\Program Files \Apache\apache-maven-3.0.5.
3. In the same dialog, append%M2_HOME%\bin to the PATH variable value.
4. Add MAVEN_OPTS as a user variable to specify JVM properties. A value of -Xms256m -Xmx512m could be a good start.

6.3.2. Linux and Mac OS X

Installing on Linux and Mac OS X is practically the same process:

1. Download and extract the apache-maven-3.0.5-bin.tar.gz file to the directory you wish to install Maven, for example /usr/local/apache-maven-3.0.5.
2. Define M2_HOME using export M2_HOME=/usr/local/apache-maven-3.0.5. You can optionally set it in ~/.bash_profile.
3. Add Maven binaries to the classpath export PATH=$PATH:$M2_HOME/bin. You can optionally set it in ~/.bash_profile.
4. Add the MAVEN_OPTS environment variable to the same .profile file, for example export MAVEN_OPTS="-Xms256m -Xmx512m".

6.4. Step 4: Verifying the installation

As Maven is a Java tool, first of all we need to be sure Java is installed properly. The quickest way to test it is by opening a terminal:

OS	Task	Command
Windows	Open Command Line Console	c:\> java -version
Linux	Open Terminal	$ java -version
Mac OS X	Open Terminal	$ java -version

In the console, run the following command, to check that Java is properly installed. For example, in a Microsoft Windows environment, you should see something similar to the following:

```
c:\>java -version
java version "1.7.0_10-ea"
Java(TM) SE Runtime Environment (build 1.7.0_10-ea-b13)
Java HotSpot(TM) 64-Bit Server VM (build 23.6-b04, mixed mode)
```

If everything is fine, you can now run the mvn command to check that Maven is installed. You should see something similar to the following output:

```
c:\>mvn -version
Apache Maven 3.0.5 (r01de14724cdef164cd33c7c8c2fe155faf9602da; 2013-02-19 14:51:28+0100)
Maven home: C:\Program Files\Apache\apache-maven-3.0.5
Java version: 1.7.0_10-ea, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.7.0_03\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```

Now you are ready to work with Maven!

ARCHETYPE

An archetype is an Apache Maven template (or a blueprint, if you prefer); it can contain files and folders, any content is allowed; the only mandatory element is the root pom.xml.

The Maven Archetype Plugin available at *http://maven.apache.org/archetype/maven-archetype-plugin/* delivers all goals that you need to handle this type of artifact:

- generate: It creates a Maven project, given the archetype's coordinates that are passed via the command line. If coordinates are not specified, Maven will show a list of archetypes available on Maven Central and will interactively let the user choose which archetype to use.
- create-from-project: It can be executed from the root of any Maven project; it generates an artifact (in target/generated-source/archetype) containing the archetype with the structure and contents of the current Maven project.

Every Maven Repository can be requested to deliver the complete list of archetypes hosted just by running the following command:

```
mvn archetype:generate \
 -DarchetypeCatalog=http://artifacts.mysite.com/repository/archetype-
catalog.xml
```

If the property archetypeCatalog is not set, the Maven Central Repository will be used, showing a list of more than 700 archetypes (you may want to search "archetype" on *http://search.maven.org* first.

POM

El POM (Project Object Model): [http://books.sonatype.com/mvnref-book/reference/pom-relationships.html]

El POM file es el archivo XML de configuración de cualquier proyecto Maven, en él podemos encontrar información sobre la versión y las dependencias del proyecto así como cualquier otra configuración personalizada para el proyecto.

8.1. Minimal pom.xml

An example of a minimal pom.xml file is as follows:

```
<project>
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.mycompany.app</groupId>
   <artifactId>app-web</artifactId>
   <version>1.0-SNAPSHOT</version>
   <type>war</type>
</project>
```

The pom.xml file is the central configuration file for a Maven project, and it is fundamental to understand it deeply; it contains information about the project structure, metadata, and configuration related with the plugin's executions.

The modelVersion element is in every pom.xml; it represents the pom.xml XML schema version and is set to 4.0.0 for all Maven 2.x and 3.x-based builds.

The SNAPSHOT suffix in a pom.xml file specifies that the artifacts produced by this build are unreleased, and therefore they considered nightly builds.

In this section we will introduce the most important elements of a POM; some will not be mentioned, while some others will be briefly introduced.

8.2. Fig. The Project Object Model

Figure 8.1. The Project Object Model

8.2.1. Super POM

Every Maven POM implicitly inherits from Super POM (more information is available at *http://maven.apache.org/ref/3.0.5/maven-model-builder/super-pom.html*, which contains all the default values that are needed to perform built-in Maven features, as we will see later in this book. Super POM is provided by the Maven installation.

It is not intended to be changed as it would cause build portability issues (more information is available at *http://www.devx.com/Java/Article/32386* but it is definitely interesting to read and investigate it further (more information is available at *http://maven.apache.org/guides/introduction/introduction-to-the-pom.html* in order to be more confident when using/overriding values in your pom.xml file.

8.3. Parent (also known as POM Inheritance)

A pom.xml file can define a parent as a pointer to a POM artifact. As a result, all parent's Maven configurations will be inherited.

```
<parent>
   <groupId>com.mycompany.myproject</groupId>
   <artifactId>my-parent-pom</artifactId>
   <version>1.0-SNAPSHOT</version>
</parent>
```

8.4. Effective POM

Each execution of a Maven build will first of all merge the current pom.xml definition with the following:

- Parent POM (and any parent's parent POM, if present)
- Super POM
- Enabled profiles

The result is called the effective POM, and it's the definition that is taken into consideration when executing Maven builds.

When writing a pom.xml definition or debugging a Maven build it is crucial to investigate the effective POM in order to deeply understand what Maven is doing.

In order to visualize it, simply run the command mvn help:effective-pom, or use an IDE that supports effective POM visualization.

The Maven Help Plugin available at *http://maven.apache.org/plugins/maven-help-plugin* provides more useful goals to debug your Maven logic.

PROPERTIES

- The definition of properties in Maven is as follows:

```
<properties>
   <dbDriver>postgres</dbDriver>
</properties>
```

In our example, postgres is the default value.

- Maven properties are key/value pairs defined in POM that allow defining constants in your build, keeping the syntax clean and readable. You can use properties to (also partially) define the values of all POM elements: except the POM coordinates (groupId, artifactId, type, version) and a few others. For example, in multi module projects, you cannot use properties for defining parent coordinates or module name exceptions.
- Property values can be passed by the mvn command by appending #D<property name>=<property value> at the end of the line; this feature allows you to provide different build options that are easy to use. For example, if your application supports different JDBC drivers, you may want to provide the option to run the build appending #DdbDriver=mysql.
-
- The Super POM defines a list (available at *http://www.sonatype.com/books/mvnref-book/reference/resource-filtering-sect-properties.html*) of properties that you can use in your POM; the most used can be divided into the following:
  - POM information: All the coordinates of the current project can be accessed via property, including all the Maven Plugin configurations
  - Environment properties: Many environment variables, such as the hostname or the JAVA_HOME
- Our example will be as follows:

```
<dependencies>
    <dependency>
        <groupId>book.simasoft.co</groupId>
        <version>4.0.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

9.1. Maven Properties:

- *Maven Properties* [http://books.sonatype.com/mvnref-book/reference/resource-filtering-sect-properties.html]
- *MavenPropertiesGuide* [http://docs.codehaus.org/display/MAVENUSER/MavenPropertiesGuide]

ARTIFACTS

An artifact in a Maven context is a file that is (or has been) produced by a build execution and represents an application binary (of a specific version) that is subject to a lifecycle.

An artifact can have different purposes, listed as follows:

- **A project's library:** JAR, WAR, EAR, ZIP, or any file extension you may want to integrate in your build.
- **Maven plugin:** A JAR application containing the logic to execute build executions.
- Maven archetype:* A JAR application containing the logic to create a Maven project with a pre-defined file and folder content. Archetypes will be introduced in the next section.
- Project descriptor:* As we'll see shortly, a POM is itself an artifact.

The following coordinates uniquely identify an artifact:

- **groupId:** This coordinate represents the group, company, team, organization, and/or project related with the given artifact. The convention is the same as that of the Java packages (more information is available at *http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html*. For example, projects from Apache Software Foundation would have a groupId coordinate that starts with org.apache. A good rule of thumb for deciding the g ranularity of groupId is by following the project's structure. For example, com.mycompany.myproject.persistence, com.mycompany.myproject.api, and so on.
- **artifactId:** This coordinate represents the name of the project (or module) related with the given artifact. artifactId must not contain any version-related information; if the artifact is a module, it is advised to join the project name with the module one (that is, commons-logging). Using only lowercase letters and the dash (-) as a separator is a clear and consolidated strategy. Good examples for artifactID are maven-core,commons-math, and log4j-over-sl4j.
- **type:** The extension (and filetype) of the artifact; the default type is JAR, but it can have any extension, such as WAR, EAR, or any other.
- **version:** This coordinate is a specific release of a project. It consists of a group of literals separated by dots; for example, 1.0, 2.0.1-RC1, and 2.0.0.1-alpha-2. If the version ends with the SNAPSHOT literal, it means that the artifact is a nightly build and therefore not released yet. In order to take full advantage of Maven's version management, every work in progress project should have a SNAPSHOT version; we will discuss this later.
- classifier:* This is an additional coordinate to handle two (or more) artifacts having the same coordinates but containing a different content; empty by default. For example, Maven identifies binary and source (source artifacts are archives containing the source code of your application, for more on this you can visit *http://maven.apache.org/plugin-developers/cookbook/attach-source-javadoc-artifacts.html* artifacts using the same coordinates, but with different classifiers.

For more information about naming conventions, check the official Maven documentation available at *http://maven.apache.org/guides/mini/guide-naming-conventions.html*

10.1. Coordinates uniquely identify an artifact

- <groupId> El id del grupo al que pertenece el proyecto.
- <artifactId> El id del artifact o proyecto (en la mayor#a de los casos el nombre del proyecto). Identificador particular de una librer#a en particular
- <version> La versión del artifact en el grupo especificado.
- <dependency> Aqu# se colocar#n todas las dependencias del proyecto.

El POM requiere que estos valores estén definidos ya que esta es la sintaxis en que Maven los identifica dentro de su repositorio.

DEPENDENCIAS

- Una dependencia es una referencia en el POM de la librería que se desea incluir en el proyecto.
- A Maven dependency is a pointerdefined in a pom.xml file to an artifact that needs to be included as a project's library. It must of course define the artifact's coordinates in order to uniquely identify it:

```
<dependencies>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>3.2.2.RELEASE</version>
   </dependency>
</dependencies>
```

- Dependencies are transitive; if A depends on B and B depends on C, then A depends on C. In order to exclude a transitive dependency from your build, you can define exclusions as follows:

```
<dependencies>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>3.2.2.RELEASE</version>
      <exclusions>
        <exclusion>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
        <exclusion>
      </exclusions>
   </dependency>
</dependencies>
```

Additionally, a dependency can define a <scope> element, but they will be introduced in the Maven lifecycle section.

11.1. Ámbito de las dependencias:

- **Compile:** Es el ámbito por defecto. Las dependencias están disponibles en el proyecto y en sus proyectos dependientes.
- **Provided:** Se espera que el JDK, la aplicación o el contenedor provea la dependencia.
- **Runtime:** La dependencia no es requerida en tiempo de compilación pero sí para la ejecución.
- **Test:** Son dependencias que son requeridas solo cuando se compila y ejecuta los test.

- **System:** Similar a provided pero se le debe indicar el jar que contiene la dependencia
- **Import:** Solo es usado en una dependencia del tipo POM en la sección . Indica que el POM utilizado debe ser remplazado con las dependencias que éste tenga en su sección

```
<dependency>
   <groupId>org.jboss.seam</groupId>
   <artifactId>seam-bom</artifactId>
   <version>${seam.version}</version>
   <scope>import</scope> Ambito
   <type>pom</type>
</dependency>
```

Los objetos en el repositorio se identifican mediante 3 etiquetas: groupId,artifactId,version

PLUGINS

A Maven plugin is a JAR Maven artifact containing Java classes that implement one or more goals using the Maven Plugin API (more information is available at *http://maven.apache.org/ref/3.0.5/maven-plugin-api*), and declares a public short name (that is, tomcat7x).

```
<build>
   <plugins>
      <plugin>
         <groupId>org.apache.tomcat.maven</groupId>
         <artifactId>tomcat7-maven-plugin</artifactId>
         <version>2.1</version>
         <configuration>
            <url>http://127.0.0.1:8080/manager</url>
            <server>Tomcat</server>
            <path>/app-web</path>
         </configuration>
      </plugin>
   </plugins>
</build>
```

Guide to Configuring Plug-ins [http://maven.apache.org/guides/mini/guide-configuring-plugins.html]

12.1. Goals

The goal identifies a build task; it has a unique name (that is, run) within the same plugin. It can access and change the POM and provide a wide range of operations, from zipping a folder to performing a remote deployment on a Tomcat server.

- A goal can be executed by the mvn command using the following syntax:

  ```
  <plugin_shortname>:<goal_name>, for example mvn tomcat7x:run
  ```

- A goal can be invoked by the lifecycle phase; you will read more about it in the Maven lifecycle section.
- All plugins having an artifactId coordinate starting with maven are directly supported by Maven projects (more information is available at *http://maven.apache.org/plugins/*.
- In order to understand how to use a Maven Plugin, search for its official documentation page; the Usage page available at *http://tomcat.apache.org/maven-plugin-2.0/tomcat7-maven-plugin/usage.html* explains how to add the plugin to your build;
- The Goals page available at *http://tomcat.apache.org/maven-plugin-2.0/tomcat7-maven-plugin/plugin-info.html* lists all goals and parameters that you can set.

PROFILES

A Maven profile is a POM element that contains specific POM configurations for specific cases. A profile can for example be used to define environment specific configurations and build behaviors, as suggested by the following code snippet:

Although profiles may be considered an advanced topic, they are widely used by the Maven community; it is crucial to know at least what they are in order to fully understand a Maven build.

```
<profiles>
    <profile>
        <id>postgres</id>
        <dependencies>
            <dependency>
              <groupId>book.simasoft.co</groupId>
              <version>4.0.0-SNAPSHOT</version>
            </dependency>
        </dependencies>
    </profile>
</profiles>
```

Profiles are activated by invocation, appending -P<profile name> to your mvn command; additionally, profiles can declare their own **<activation>** rules.

When activated, profiles are one by one, following the definition ordering merged with the main pom.xml definition into the effective POM.

The effective POM will be the result of the sum of the main pom.xml features and the profile ones.

There's much more available at *http://books.sonatype.com/mvnref-book/reference/profiles.html* to know about profiles in order to exploit all their potentials.

REPOSITORIOS

- A Maven Repository is a folder with a specific layout that can optionally be located remotely:
- En el repositorio encontramos los JAR, WAR, etc en una jerarqu#a de carpetas que siguen el mismo concepto: groupId, artifactId, version.
- El repositorio es un mecanismo espacial de Maven para organizar los ficheros Jar y otras dependencias que utilizan tus aplicaciones. (Maven tambi#n utiliza el t#rmino artefacto para referirse a las dependencias).
- El repositorio esencialmente es una carpeta con un estructura espec#fica para Maven, y soluciona dos problemas:
  - **Primero**, proporciona una localizaci#n centralizada para todos los ficheros Jar y otras dependencias que necesita tu proceso de construcci#n.
  - **Segundo**, es un ayuda con los problemas de versiones proponiendo una convenci#n de nombrado.

```
<repositories>
   <repository>
      <id>my-custom-repo</id>
      <url>http://artifacts.mysite.com/repository</url>
   </repository>
</repositories>
```

- The Repository layout is a key convention in Maven that allows you to uniquely locate an artifact:
  - <repository_url>/<groupId>/<artifactId>/<version>/<artifactId>-<classifier>-<version>.<type>
- For example, you can have the following coordinates for the preceding artifact:

```
groupId: org.apache.solr
artifactId: solr
version: 4.3.0
type: .war
```

- For a remote URL, the Repository URL can be:
  - *http://repo.maven.apache.org/maven2*
- The layout key will hence be:

  - *http://repo.maven.apache.org/maven2/org/apache/solr/solr/4.3.0/solr-4.3.0.war*
- For a local URL, the Repository URL can be:
  - file:///Users/mau/.m2/repository
- The layout key will hence be:
  - file:///Users/mau/.m2/repository/org/apache/solr/solr/4.3.0/solr-4.3.0.war
- A Maven Repository is the source and the destination of artifacts in the following scenarios:
  - **Source:** When a Maven build depends on one or more artifacts, the Maven Repository is the place where these files are resolved and downloaded from
  - **Destination:** When a Maven build produces one or more artifacts, it may be optionally deployed on a Maven Repository
- A Maven Repository can restrict the download/upload artifact operations depending whether the artifact's version is a SNAPSHOT literal or not. This way, you can easily define nightly builds, repositories, and define tailored maintenance operations (that is, remove SNAPSHOT artifacts after 30 days).
- SNAPSHOT artifacts are a special case for Maven Repositories:
  - When uploaded, the SNAPSHOT literal of the artifact name will be replaced with the current timestamp (more information is available at *http://docs.oracle.com/javase/6/docs/api/java/sql/Timestamp.html*
  - When downloaded, the revolved artifact will be the one with the highest timestamp (most recently uploaded amongst all other SNAPSHOT artifacts having the same coordinates)
  - The Super POM defines two very special repositories:

14.1. Tipos de Repositorios de Maven

Maven maneja 3 niveles de repositorio:

- **Repositorio Local:**
  - El repositorio local normalmente conocido como .m2 se guarda localmente en nuestra pc
  - This repository is a local folder located in /.m2/repository (means user home in Linux, Unix, and OS X environments). The Local Repository works as a cache for all remotely-fetched artifacts: every time Maven downloads an artifact for you, it will do it only once. This rule does not apply to -SNAPSHOT artifacts, since these versions are supposed to change frequently; in this case, the build will ask the Maven Repository whether the SNAPSHOT artifact was updated since the last fetch.

- **Repositorio Empresarial:**
  - Maven Central Repository: This is a remote Maven repository containing the official releases of Maven core plugins which deliver all built in functionalities of Apache Maven and the biggest collection of Java artifacts in the world (Java.net and Oracle are hosted here, and many other companies, projects, and communities). Maven Central is open (more information is available at *http://www.sonatype.org/central/participate*) to the contribution of anyone who wants to share their artifacts with the rest of the world. You can browse Maven Central using *http://search.maven.org* (shown in the following screenshot). Maven Central is hosted by Sonatype (more information is available at *http://www.sonatype.org/*.
  - El repositorio Empresarial tiene la misma estructura que el repositorio local con la diferencia que almacena librerías de uso compartido a nivel de red. Los repositorios empresariales normalmente se utilizan cuando se maneja un equipo de desarrolladores que trabaja sobre un conjunto variado de proyectos, los cuales normalmente tienen interdependencia entre ellos y sus librerías.
- **Repositorio Global:**
  - Los repositorios globales son repositorios públicos que almacenan librerías de uso compartido, como por ejemplo el repositorio mismo de Maven o el repositorio público de JBoss o maven central.

RECURSOS

15.1. GENERALIDADES

- Generar el proyecto:
  - mvn archetype:generate -DgroupId=com.mycompany.demo -DartifactId=myapp -DarchetypeArtifactId=maven-archetype-quickstart-DinteractiveMode=false
  - Plugin maven-shade-plugin
    - Este plugin permite configurar Maven para que al empaquetar el proyecto JAR con sus dependencias y asi poderlo ejecutar con la orden: java -jar target/myapp-1.0-SNAPSHOT.jar
    - Para lo anteror debemos agregar la secci#n build con el respectivo plugin en el archivo: pom.xml

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>1.3.1</version>
      <executions>
        <execution>
        <phase>package</phase>
        <goals>
          <goal>shade</goal>
        </goals>
        <configuration>
          <minimizeJar>true</minimizeJar>
          <transformers>

  <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
             <manifestEntries>
               <Main-Class>com.mycompany.demo.App</Main-Class>
             </manifestEntries>
           </transformer>

<transformimplementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"/
  >
          </transformers>
        </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

- Para ejecutar el programa java sin utilizar maven:

- javac -cp mongo.jar;. App.java // Driver de mongo.jar debe estar en el mismo directorio.
- java -cp mongo.jar;. App

- Ejemplos para compilar y ejecutar el programa
  - mvn compile exec:java -Dexec.mainClass=com.tengen.Week1Homework3
  - mvn compile exec:java -Dexec.mainClass=com.tengen.Week1Homework4
  - java -jar target/myapp-1.0-SNAPSHOT.jar

15.2. LINKS:

- Referencias
  - *Maven: The Complete Reference* [http://books.sonatype.com/mvnref-book/reference/]
  - *Introducción a Maven* [http://www.genbetadev.com/java-j2ee/introduccion-a-maven]
  - *Maven, nunca antes resultó tan fácil compilar, empaquetar* [http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=maven]
  - *Maven Getting Started - Users* [https://community.jboss.org/wiki/MavenGettingStarted-Users]
  - *introduction* [http://maven.apache.org/guides/introduction/]
- Maven Archetypes
  - *Maven Archetypes* [http://maven.apache.org/archetype/maven-archetype-bundles/]
  - *Guide to Creating Archetypes* [http://maven.apache.org/guides/mini/guide-creating-archetypes.html]
- Repositorios
  - *The JBoss Community public repository is a composite repository* [https://repository.jboss.org/nexus/content/groups/public/]
  - *Maven Repository Magic* [http://vimeo.com/12620367]
- Pluins
  - *Maven Resources Plugin* [http://maven.apache.org/plugins/maven-resources-plugin/]
  - *Hacer un plugin para Maven* [http://chuwiki.chuidiang.org/index.php?title=Hacer_un_plugin_para_Maven]
- Properties

- *MavenPropertiesGuide* [http://docs.codehaus.org/display/MAVENUSER/MavenPropertiesGuide]
- *Maven Properties* [http://books.sonatype.com/mvnref-book/reference/resource-filtering-sect-properties.html]

- Dependencias
  - *Introduction to the Dependency Mechanism* [http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html]

- POM
  - *Introduction to the POM* [http://maven.apache.org/guides/introduction/introduction-to-the-pom.html]

- Libros
  - *Safari Books Online* [https://ssl.safaribooksonline.com/trial?iid=anon-home-redirect]
  - *Amazon* [http://www.amazon.com/books-used-books-textbooks/b/ref=sa_menu_bo?ie=UTF8&node=283155]
  - *Instant Apache Maven Starter - ISBN: 978-1-78216-760-0* [http://my.safaribooksonline.com/book/operating-systems-and-server-administration/apache/9781782167600]

- Varios
  - *Introduction to the Standard Directory Layout* [http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html]

GLOSARIO MAVEN

Maven
: Es una herramienta para la gesti#n de proyectos de software, que se basa en el concepto de POM (Proyect Object Model).

Arquetipo
: Es una plantilla,capaz de generar una estructura de directorios y archivos.