Consejería de Innovación, Ciencia y Empresa

JUNTA DE ANDALUCIA

Curso de Introducción a Maven 2

la nueva idea de Andalucía

Innovar, nuestra forma de avanzar y competir en el entorno global

El Origen de MAVEN

- Maven originalmente empezó como un intento de simplificar la el proceso de construcción en el proyecto **Jakarta Turbine**.
- Había varios proyectos cada uno con sus propios ficheros build de Ant, que eran levemente diferentes, y JARs subidos al CVS.
- Se quería una manera **estándar de construir** los proyectos, una definición clara de que consistía el proyecto, un medio fácil de publicar información del proyecto y una forma de **compartir** jars entre varios proyectos.

¿Que es MAVEN?

- Maven es un framework de gestión de proyectos de software.
- Es un proyecto de código abierto de la fundación de software Apache.
- Maven esta Basado en POM (**project object model**). Cada proyecto tiene la información para su ciclo de vida en el descriptor xml (por defecto el fichero pom.xml)
- Maven Proporciona funcionalidades desde la compilación hasta la distribución, despliegue y documentación de los proyectos.
- Maven posee las abstracciones necesarias que animan la reútilización y que ayudan a la estructuración del proyecto.

¿Que es MAVEN?

- Maven es un Framework de gestión de proyectos
- NO ES SÓLO un software para Gestionar proyectos software
  - *"Maven es un sistema de estándares, un repositorio y un software usado para manejar y describir proyectos. Define un ciclo estándar para la construcción, prueba y despliegue de componentes del proyecto. Proporciona un marco que permite la reutilización fácil de la lógica común de la estructura para todos los proyectos que siguen los estándares Maven"*

Instalación

Para instalar maven solo tenemos que seguir estos paso:

1) Desempaquetamos el archivo maven-2.0.x-bin.tar.gz
http://maven.apache.org/download.html

- **tar zxvf maven-2.0.x-bin.tar.gz "o"**
- **unzip maven-2.0.x.zip**

2) Añadir el directorio a el PATH:

- **export PATH=.:/home/prueba/maven-2.0.x /bin:$PATH "o"**
- **set PATH="c:\program files\maven-2.0.x\bin";%PATH%**

3) Establece la variable JAVA_HOME a la carpeta del jdk.

- **export JAVA_HOME=/home/prueba/jdk1.5.0_09**

Creación de un Proyecto

Un Proyecto en Maven es básicamente una carpeta en la que tenemos un pom.xml (descriptor de proyecto). En lugar de crear este a mano podemos hacer uso de un gran número de plantillas o archetype. Por ejemplo para crear un proyecto para struts.

```
mvn archetype:create -DgroupId=tutorial
-DartifactId=tutorial -DarchetypeGroupId=org.
apache.struts -DarchetypeArtifactId=struts2-
archetype-starter -DarchetypeVersion=2.0.2-
SNAPSHOT
```

Estructura de un Proyecto

Maven define una estructura de carpetas por defecto para todos los elementos que componenen un desarrollo. Esta estructura puede ser modificada en el pom.xml.

- src/main/java: Código Fuente
- src/main/resources: Otros recursos
- src/main/webapp: páginas, tags, etc.
- src/main/webapp/WEB-INF: Contiene el web.xml y otros ficheros de configuración.
- src/test/java: Código Fuente de pruebas.
- src/test/resources: Otros recursos para las pruebas.
- site: Carpeta con información para generar una página del proyecto.
- target: Carpeta donde se guardan los resultados.

Gestión Declarativa.

En maven un proyecto se define a partir de su fichero **pom.xml**. Este fichero se encuentra siempre en la raiz del proyecto y contienen toda información necesaria para el ciclo de vida del proyecto: **dependencias, plugins, repositorios de donde obtener estos, configuración de los informes**.

Además tambíen otros datos de interes: **SCM del proyecto, Equipo de Desarrollo, listas de distribución**, etc.

Esta información puede ser luego publicada en la site del proyecto que se puede generar con maven.

Un pom.xml Básico

```
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSche
ma-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>ejemplos</groupId>
  <artifactId>EjemploMaven1</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>EjemploMaven1</name>
  <url>http://maven.apache.org</url>
```

Un Pom.xml Básico

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

Dependencias

- Son aquellos otros componentes que nuestro software necesita en algún momento del ciclo de vida.
- Una de las grandes novedades de maven2 es que las dependencias no acompañan al código fuente de nuestro desarrollo.
- Según el momento en el que se necesite una dependencia tiene distinto ámbito o 'scope', este puede ser: **compile, test, provided, run-time, system**. Por defecto se entiende que son para compilar.
- Maven obtiene automaticamente las dependencias del proyecto, bien del repositorio local si ya esta o bien de un repositorio remoto.

Dependencias

- Elementos que describen la dependencia:
  - **groupId**: Hace referencia a la organización, suele coincidir con el packete.
  - **artifacId**: Identificador del componente que se esta desarrollando.
  - **version**: Version actual del desarrollo.
  - **scope**: Ámbito en el que se usa (compile, test, provided, run-time, system).
  - **type**: Empaque tamiemo de la dependencia.
  - **Classifier**: Texto adicional que describe la dependencia como (prod indicar que esta en producción por ejemplo).

Plugins

- En maven tenemos un conjunto goals asociados al ciclo de vida que forman el nucleo de maven **(compile, test package, install, deploy, site).**
- Además existen un gran número de plugins por internet que al igual que las dependencias se pueden instalar manualmente o obtener simplemente declarando su uso en el pom.xml
- Tambien es posible crear plugins nuevos a partir de clases **MOJO** (Maven Old plain Java Objects).

El Ciclo de Vida

Las principales fases del ciclo de vida en maven son:

validate: valida que el proyecto esta correcto y tienen toda la información necesaria par su construcción.

compile: compila el código fuente del proyecto.

test: lanza los test de la aplicación. Estos test no necesitan que la aplicación este empaquetada ni desplegada.

package: toma las clases compiladas y recursos y crea un paquete con el proyecto (jar, war, ear)

El Ciclo de Vida

integration-test: procesa y despliega el paquete antes si es necesario para que corran las pruebas de integración.

verify: realiza algun tipo de chequeo para comprobar si el paquete cumple unas rormas de calidad.

install: instala el paquete en el repositorio local para ser usado como dependencia por otros proyectos localmente.

deploy – copia el paquete a un reposiotorio remoto para ser compartido con otros usuarios y proyectos.

Compilación

En este ejemplo podemos ver como se compila con **mvn compile** el proyecto y el resulta do que se genera en la carpeta **target**.

```
prueba@guadalinex:~/Proyectos/comunicacion_interna_default$ ls
pom.xml  src
prueba@guadalinex:~/Proyectos/comunicacion_interna_default$ mvn compile > salida_compilacion.txt
prueba@guadalinex:~/Proyectos/comunicacion_interna_default$ ls
pom.xml  salida_compilacion.txt  src  target
prueba@guadalinex:~/Proyectos/comunicacion_interna_default$ cd target/
prueba@guadalinex:~/Proyectos/comunicacion_interna_default/target$ ls
classes
prueba@guadalinex:~/Proyectos/comunicacion_interna_default/target$ cd classes/
prueba@guadalinex:~/Proyectos/comunicacion_interna_default/target/classes$ ls
cap  css  images  js
prueba@guadalinex:~/Proyectos/comunicacion_interna_default/target/classes$
```

Empaquetamiento e Instalación

En maven empaquetar el proyecto es tan sencillo como usar el goal package. También podemos instalarlo después en el repositorio local para que este disponible para otros proyectos con el goal **install**.

```
prueba@guadalinex:~/Proyectos/comunicacion_interna_default$ ls
pom.xml  src  target
prueba@guadalinex:~/Proyectos/comunicacion_interna_default$ mvn package > salida_package.txt
prueba@guadalinex:~/Proyectos/comunicacion_interna_default$ ls
pom.xml  salida_package.txt  src  target
prueba@guadalinex:~/Proyectos/comunicacion_interna_default$ cd target/
prueba@guadalinex:~/Proyectos/comunicacion_interna_default/target$ ls
classes  comunicacion-interna  comunicacion-interna.war
prueba@guadalinex:~/Proyectos/comunicacion_interna_default/target$ mvn install > salida_install.txt
```

El tipo de empaquetamiento se indica en el pom.xml del proyecto.

<packaging>war</packaging>

Pruebas

Desde maven podemos compilar el código de las pruebas que alojemos en **src/test** con el goal **test-compile**.

Luego las lanzaremos con el goal **test** y veremos en la consola los resultados que se guardan bajo la carpeta **proyecto/target/surefire-reports.**

También podemos differnciar entre que clases queremos que se ejecuten como pruebas unitarias antes de crear el paquete o en la fase de pruebas de integración.

Esta información también aparecerá cuando generemos la web para nuestro proyecto.

Crear el Site del Proyecto

Para ello tendremos que crear nuestro descriptor, **site.xml**, en la carpeta **src/site** y luego ejecutar el goal **site**, o **site-deploy** si queremos desplegar la web en algun sitio tras generarla.

site/apt	Esta carpeta contiene ficheros con formato apt que ses un formato tipo wiki para escribir documentos de texto estructurados de forma simple. La referencia a este formato la podemos encontrar en esta url: http://maven.apache.org/guides/mini/guide-apt-format.html
site/fml	Esta carpeta contiene ficheros de formato FAQ. Un formato xml simple para gestión de FAQs.
site/xdoc	Esta carpeta contiene ficheros xml. Estos ficheros son traducidos automáticamente por maven a páginas html. Usaremos este sistema para crear una página propia dentro del sitio web que acceda a documentos guardados en *Alfresco* junto con las que genera maven.
site/resources	Aquí se guardan todos los recursos que queremos que se copien al sitio web cuando se genere, y que no necesitan ningún otro tratamiento. Por ejemplo las css, scripts, imágenes, etc.

Menu del Site

En el menú lateral podemos ver los siguientes apartados:

- Información del proyecto
- Informes del proyecto
- Documentación Asociada.

Las dos primeras forman parte se generan automáticamente y extraen su información de lo que aparece en el pom.xml. La tercera 'Documentación asociada' la hemos añadido en el site.xml, y hace referencia a una página documentos.html que maven genera aútomaticamente a partir del fichero documentos.xml que situamos en la carpeta **src/site/xdoc**

Información del Proyecto

En 'información del proyecto', la información que aparece es del tipo:

- Si el proyecto usa **integración continua**.
- **Dependencias** del proyecto.
- Información sobre la **gestión de incidencias**.
- **Listas de correo.**
- **Licencias del proyecto**.
- **Equipo** de trabajo y **colaboradores** del proyecto.
- **Repositorio de código** con el que se trabaja.
- etc.

Informes de Métrica

En la parte de 'informes del proyecto' podemos destacar:

- El test de **cobertura**
- **CPD**
- **PMD**
- **Xref** (Informe de referencias cruzadas).
- Informe sobre **tags**.
- Documentación del **javadoc**.
- Resultado de los test de **junit**.
- **Checkstyle**.

Configurar un Plugin

Para modificar la configuración por defecto de los plugins usamos la etiqueta <configuration>

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-site-plugin</artifactId>
<configuration>
<locales>es_ES</locales>
<encoding>UTF-8</encoding>
<inputEncoding>UTF-8</inputEncoding>
<outputEncoding>UTF-8</outputEncoding>
</configuration>
</plugin>
```

Configurar un Plugin

Además podemos indicar la configuración del plugin para una fase concreta del ciclo de vida.

```
<executions>
        <execution>
          <id>it-test</id>
          <phase>integration-test</phase>
          <goals>
            <goal>test</goal>
          </goals>
          <configuration>
            <includes>
              <include>**/*TestIt.class</include>
            </includes>
          </configuration>
        </execution>
 </executions>
```

SCM

El plugin SCM nos permite comunicarnos con sistemas de control de versiones com Subversion, CVS, Clearcase, Visual Source Safe, etc.

<connection>
scm:svn:
http://localhost/repos/compulsa
</connection>

Su url
http://maven.apache.org/scm/plugins/

Plugin Cargo

Este plugin nnos da la posibilidad de trabajar directamente desde Maven2 con distintos contenedores y servidores de aplicaciones (Tomcat, Jboss, Jetty, jo!, OC4J, Orion, Resin, Weblogic).

Para usarlo añadimos la configuración al pom.xml de nuestro proyecto ejecutamos: **mvn cargo:deploy,** o **mvn cargo:undeploy** para quitar el paquete antes desplegado. Tiene más usos y goals, toda la documentación en la url: http://cargo.codehaus.org/Maven2+plugin
También podemos configurarlo para usar una instalación de tomcat locar al proyecto y levantarlo y tirarlo con el plugin: **mvn cargo:start mvn cargo:stop**.

Perfiles

Podemos definir distitos perfiles cada uno con una configuración específica para el cilo de vida.

De esta manera podemos tener un perfil para pruebas y otro para producción con propiedades diferentes y desplegando en entornos diferentes.

Para definir perfiles se podemos usar las etiquetas <profile> </profile> en el pom.xml

O tambíen crealos en un fichero profiles.xml aparte.

Perfiles

```
<profiles>
        <profile>
                <id>tomcat5x</id>
                <activation>
         <activeByDefault>
                true
         </activeByDefault>
        <build>
         <plugins> </plugis>
        </build>
</activation>
```

Filtering (I)

El Filtering nos da la posibilidad de cambiar los properties que se incluiran en el paquete de la aplicación de forma dinámica. Para esto añadiremos a nuestra aplicación Compulsa lo siguiente:

```
<build>
<!-- para filtrar los recursos y poder usar properties
<resources>
        <resource>

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

Filtering (II)

A continuación creamos un fichero **prueba.properties** en nuestra carpeta **src/main/resources:**

```
#prueba.properties
application.name=${pom.name}
application.version=${pom.version}
```

Si ejecutamos **mvn process-resources** podemos ver que nos crea en la carpeta target/classes/ un fichero prueba.properties con el contenido:

```
#prueba.properties
application.name=Compulsa
application.version=1.0
```

Filtering (III)

process-resources es la fase del ciclo de vida de construcción donde los recursos son filtrados y copiados.
También podemos añadir a una property de nuestro fichero un valor de otro fichero externo.
Por ejemplo tomando el properties compulsa.properties como fuente, aña dimos a lo de antes lo siguiente. Lo suyo sería ponerlo en la carpeta de filtros y no en la de recursos

```
<build>
 <filters>

<filter>src/main/resources/compulsa.properties</filter
>
 </filters>
 ....
</build>
```

Filtering (IV)

Si volvemos a ejecutar **mvn process-resources** veremos que en target/classes hay un prueba.properties con el siguiente contenido:

application.name=Compulsa
application.version=1.0
servidorfirmaDesarrollo=ws083.juntadeandalucia.es

Otra forma de añadir propiedades es añadir a nuestro pom.xml una sección <properties>

<properties>

<my.filter.value>hello</my.filter.value>
</properties>

Filtering (V)

<properties>

<my.filter.value>hello</my.filter.value>
</properties>

Nuevamente para hacer referencia en nuestro properties de prueba seria **${my.filter.value}** y si ejecutamos el goal de la fase copiado y filtrado volveremos a ver el mismo resultado que en los anteriores ejemplos.

Filtering (VI)

Por último podemos pasar el valor de las propiedades por linea de comandos por ejemplo.

mvn process-resources "-Dproperty=valor1"

Y hacer la referencia en nuestro property igual que antes con ${property}

Proyecto Multi Modulo (I)

Si nuestro proyecto es realemente un conjunto de varios modulos o proyectos independentes, como puede ser auna apliación enterprise con un modulo de persistencia, uno de negocio y otro web y queremos construrlos todos a la vez, ¿Como lo hacemos?

```
+- pom.xml
        +- capaWeb
                | +- pom.xml
        +- capaNegocio
                | +- pom.xml
        +- capaPersistencia
                | +- pom.xml
```

Proyecto Multi Modulo (II)

Como se puede ver en el arbol anterior la idea es tener un pom.xml en lo que seria la carpeta del proyecto global. Sería más o menos así:

```
<project
xmlns="http://maven.apache.org/POM/4.0.0"
.......
  <packaging>pom</packaging>
  <modules>
    <module>capaPersistencia</module>
    <module>capaNegocio</module>
    <module>capaWeb</module>
  </modules>
</project>
```

Proyecto Multi Modulo (III)

A continuación añadiremos a cada pom.xml la dependencia con los otros y la referencia al pom.xml global. Por ejemplo en el pom.xml de la capaWeb.

```
<parent>

<groupId>es.juntadeandalucia.ejemplos</groupId>
    <artifactId>capaWeb</artifactId>
    <version>1.0</version>
</parent>
```

Proyecto Multi Modulo (IV)

```
  <dependency>

<groupId>com.mycompany.app</groupId>
    <artifactId>capaNegocio</artifactId>
    <version>1.0</version>
   </dependency>

  <dependency>

<groupId>com.mycompany.app</groupId>
    <artifactId>capaPersistencia</artifactId>
    <version>1.0</version>
   </dependency>
```

Assembly

Este plugin se usa para crear una distribución binaria de nuestro proyecto en Maven que incluya soporte de scripts, ficheros de configuración y dependencias en tiempos de ejecución.

Vamos a usar Assembly para crear un zip con todo lo necesario para entregar. Hay varios ya predefinidos

mvn assembly:assembly -DdescriptorId=bin
mvn assembly:assembly -DdescriptorId=jar-with-dependencies
mvn assembly:assembly -DdescriptorId=src

Assembly

Tambíen podemos crear un fichero dep.xml donde indicar que cosas empaquetar dentro de un zip. Y añadimos la configuración al pom.xml para que coja por defecto este fichero.

```
<plugin>
                    <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
          <descriptors>
            <!-- Sitio en el que esta el fichero anterior -->

<descriptor>src/main/assembly/dep.xml</descriptor>
          </descriptors>
        </configuration>
      </plugin>
```

Assembly

Tambíen podemos crear un fichero dep.xml donde indicar que cosas empaquetar dentro de un zip. Y añadimos la configuración al pom.xml para que coja por defecto este fichero.

```
<plugin>
                    <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
          <descriptors>
            <!-- Sitio en el que esta el fichero anterior -->

<descriptor>src/main/assembly/dep.xml</descriptor>
          </descriptors>
        </configuration>
      </plugin>
```

Assembly

Este podría ser un ejemplo de dep.xml que nos crease un zip con los fuentes por un lado y con el war y los jar necesarios por otro.

```
<assembly>
        <id>entrega</id>
        <formats>
                <format>zip</format>
        </formats>
        <fileSets>
            <fileSet><!--src con los fuentes -->
                <directory>src</directory>

<outputDirectory>/fuentes</outputDirectory>
            </fileSet>
```

Assembly

```
<fileSet><!--pom.xml-->

<outputDirectory>/fuentes</outputDirectory>
        <includes>
           <include>*.xml</include>
        </includes>
</fileSet>
<fileSet>  <!-- war de la aplicación y jar de dependencias -->
        <directory>target</directory>

<outputDirectory>/binarios</outputDirectory>
                <includes>
                        <include>*.jar</include>
                        <include>*.war</include>
                </includes>
</fileSet>
```

Assembly

```
</fileSets>
        <dependencySets>
            <dependencySet>

<outputDirectory>/binarios</outputDirectory>
                    <scope>runtime</scope>
            </dependencySet>
        </dependencySets>
</assembly>
```

Integración con Eclipse

Para casar el uso de maven con Eclipse emplearemos dos plugins.

Primero para desde la carpeta del proyecto maven haremos un **mvn eclipse:eclipse**. Este plugin de maven te crea los ficheros de configuración del proyecto para eclipse (.project .settings .classpath)

A continuación poemos irmos a eclipse e importar el proyecto.

Integración con Eclipse

Ahora aparecea el proyecto en eclipse pero para qeu compile tendremos qeu añadir la variable **M2_REPO** a las variables de classpath con la dirección de nuestro repositorio local /home/usuario/.m2/repository

A continuación nos instalamos el segundo plugin **http://m2eclipse.codehaus.org/update/site.xml** pero este desde eclipse, en help-> install

Este plugin podremos lanzar tareas de maven2 desde el External Tools de Eclipse.

Integración con Eclipse

Tambien instalaremos este otro alojado en **http://code.google.com/p/q4e/wiki/Installation**

Este plugin además de permitirnos lanzar goals nos da otras utilidades:

- Manejar las dependencias del proyecto
- Vista gráfica de las dependencias del proyecto

Herramientas sobre Maven

- **Continuum**: Es un servidor de integración continua para proyectos basados en Java. Soporta proyectos en Maven 2, Maven 1, Ant, Shell scripts. La integración continua es una metodología informática propuesta inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de tests de todo un proyecto.

Herramientas sobre Maven (II)

- **Maven Proxy**: Es un motor ligero de servlets que se instala en tu servidor e imita otros repositorios como ibiblio por ejemplo. Cuando recibe una petición de una jar y no lo tiene, lo consigue de los servidores y lo almacena en cache antes de devolver la petición. No tiene administración, ni navegador visual.

- **Archiva**: Es el gestor para el repositorio de Maven. Esta aún en desarrollo. Algunas de sus funcionalidades son: busqueda de información en los pom.xml y ficheros, gestión del repositorio, mantenimiento y reporting.

Herramientas sobre Maven (III)

- **Artifactory:** Ofrece funcionalidades de proxy, cache y seguridad muy avanzas y tiene funciones de backup y de búsqueda de librerías. Artifactory utiliza un JSR - 170 compatible con Java Content Repository (JCR) para el almacenamiento, lo que hace fácil de manejar los metadatos totalmente indexados y buscables para proporcionar características extendidas como la seguridad, las operaciones transaccionales,la realización de auditorías, bloqueo, etc