MAVEN

<https://www.javaworld.com/article/2071772/java-app-dev/the-maven-2-pom-demystified.html>

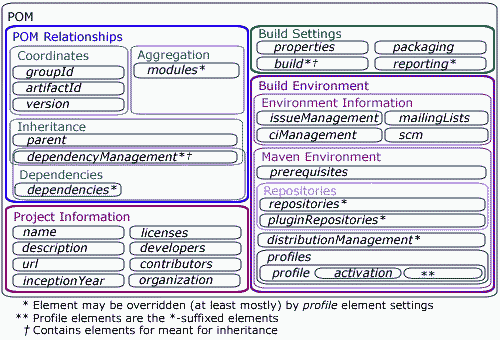
Ver también el ciclo de Maven.

Más que una herramienta de construcción de proyectos, Maven es un *framework* de construcción de proyectos que separa limpiamente el código de los ficheros de configuración, la documentación y las dependencias. De hecho, el proyecto “mora” dentro de la estructura de Maven.

El fichero de configuración de un proyecto dentro de Maven es el .pom (Project Object Model).

Todas las configuraciones del proyecto dentro del .pom pueden agruparse en cuatro unidades lógicas:

* Las relaciones
* La información del proyecto
* La configuración de la construcción (build)
* El entorno de construcción o compilación



**Relaciones**

Hay cuatro tipos de relaciones en Maven: Coordenadas, dependencias basadas en coordenadas, herencia entre proyectos y agregación.

* Coordenadas (coordinates): cada proyecto Maven contiene su identificador único, llamado también las “coordenadas” del proyecto, que actúa como la “dirección” única de un artefacto. Estas coordenadas son <groupId>, <artifactId> y <version>, que son campos obligatorios.
* Dependencias: la gestión de las dependencias es uno de los fuertes de Maven, así como las dependencias transitivas de Maven 2. Con un repositorio local común a todas las dependencias y proyectos, Maven resuelve el problema atroz y recurrente del “Jarmageddon” y el “jar hell”.
* Herencia: un proyecto Maven puede heredar de un padre (definido con el tipo de empaquetado = pom; ver “packaging”). Así, el hijo tendrá que usar las coordenadas del padre entre las etiquetas <parent>, heredando su <groupId> y su <version> y definiendo sólo su <artifactId>. En realidad, todos los .pom heredan de un padre, el “super pom”, lo hagan o no explícitamente.

Es importante resaltar la etiqueta <dependencyManagement>, que es un instrumento muy útil para configurar un conjunto de dependencias a través de un padre común. Ver la etiqueta en detalle.

* Agregación: etiqueta <modules>. Un proyecto que define estos módulos se conoce como “multi-módulo”. Los módulos son proyectos que se ejecutan como un conjunto (“the reactor” los ejecuta en un orden determinado y necesario). Ver la diferencia con una dependencia y su necesidad.

**Información del proyecto:**

Los elementos de esta parte tienen un propósito informativo para quien usa el proyecto.

**Configuración de la construcción (build settings):**

Esta sección se compone de cuatro etiquetas: <build>, <reporting>, <packaging> y <properties>, de las cuales las dos primeras son las más importantes. Empezaremos por las menos:

* Packaging: si no se especifica toma el valor “jar”. Los otros valores posibles son “pom”, “maven-plugin”, “ejb”, “war”, “ear”, “rar”, “par” y “ejb3”. Cada uno de ellos está asociado con una lista de “goals” a ejecutar correspondiente a su ciclo de construcción particular.
* Properties: es un apartado en que se definirán propiedades con sus valores. Dado que hay muchas otras maneras de definir propiedades, no es una característica muy importante.
* Build: define toda la información necesaria para construir / compilar el proyecto, desde la localización del código fuente a la configuración de los plugins de maven que usará, entre otros. Mucha de esta información se herada del .pom padre. Todas las rutas que no contengan “/” serán relativas a ${baseDir} (¿?).

La etiqueta <build> puede contener además, para dar cabida a todas esas características configurables, las siguientes:

<plugins>

<executions>

<configuration>

**Maven build lifecycle:**

El ciclo de vida de una aplicación (el proceso de construcción y distribución de un artefacto) Maven se compone de fases. Hay tres ciclos de vida que vienen con el propio Maven (aunque se pueden definir otros), que son el ciclo “default”, “clean” y “site”.

Ver <https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference>

**default**:

Maneja el despliegue del proyecto, y tiene las siguientes fases:

* validate
* compile: compila el código fuente del proyecto
* test
* package: empaqueta el código compilado en un formato distribuible
* verify
* install: instala el paquete en el repositorio local para ser utilizado como dependencia por otros proyectos en local
* deploy

Las fases de un “build lifecycle” se ejecutan secuencialmente . Si ejecutamos “mvn install”, se ejecutarán todas las fases previas hasta llegar a “install”. Si fuera “test”, ejecutaría “validate” y “compile” previamente, hasta llegar a “test”, y ahí se detendría.

**clean**:

Maneja la limpieza del proyecto. Fases:

* pre-clean
* clean
* post-clean

**site**:

Maneja la creación de documentación para el proyecto.

Se puede llamar directamente a las fases. Por ejemplo: “mvn clean install” es muy típica. Una fase de construcción (por ejemplo “install”) está hecha (se subdivide) de “plugin goals”. Un “goal” puede no estar asociado a una fase. Si así fuera, se ejecutaría invocándolo directamente. Así mismo, una fase puede no tener ningún “goal” asociado, con lo cual esa fase no se ejecutará. Tanto las fases como los “goals” se ejecutan en el orden en que hayan sido llamados, o definidos.

Un “goal” representa una tarea específica (es de grano más fino que una fase).

¿Los “goals” pertenecen siempre a un plugin? Ver la relación entre los “goals” y los plugins.

Las fases que llevan en el nombre un guión (pre-\*, post-\* o process-\*) no se suelen usar desde la línea de comandos porque representan pasos intermedios de algún proceso de construcción y no tienen utilidad por sí solas. En algunos casos, invocar estas fases puede colgar el entorno Java.

Packaging

Como es obvio, se define en el tag <packaging> de los “build settings”. Como se ha dicho, si no hay un valor explícito, el empaquetado por defecto es “jar”.

Cada tipo de “packaging” lleva asociados unos “goals” diferentes para cada fase.

Default bindings for packaging:

<https://maven.apache.org/ref/3.5.3/maven-core/default-bindings.html>

Lifecycles:

<https://maven.apache.org/ref/3.5.3/maven-core/lifecycles.html>