Maven

Apache Maven

Propósito y Gestión de Dependencias

¿Qué es Maven?

Herramienta de automatización y gestión de proyectos Java que proporciona un modelo estándar para la construcción de proyectos.

Propósito Principal

- Gestión centralizada de dependencias
- Automatización del ciclo de vida del build
- Estandarización de la estructura de proyectos

Beneficios Clave

Gestión de Dependencias
Descarga y gestiona automáticamente las bibliotecas y sus dependencias

Estandarización
Estructura de proyecto consistente y convenciones predefinidas

Ciclo de Vida
Fases predefinidas: clean, compile, test, package, install, deploy

Filosofía Maven

"Convention over Configuration" - Minimiza la necesidad de configuración al seguir convenciones predefinidas.

POM.XML

Project Object Model

Estructura y Elementos Clave

Estructura Básica

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>My Application</name>
  <description>...</description>

  <properties>...</properties>

  <dependencies>...</dependencies>
```

Coordenadas Maven

groupId: Identificador del grupo/organización

artifactId: Nombre del proyecto

version: Versión del proyecto

packaging: Formato del empaquetado (jar, war, ear)

Elementos Clave

dependencies

Gestiona las bibliotecas externas que el proyecto necesita.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Alcances (scopes): compile, provided, runtime, test, system, import

build

Configura el proceso de construcción y los plugins.

Elementos Comunes:

- finalName
- directory
- resources
- testResources

Configuración de Plugins:

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupI
    <artifactId>maven-compiler-plugin</artifa
    <version>3.8.0</version>
    <configuration>
      <source>11</source>
      <target>11</target>
    </configuration>
  </plugin>
</plugins>
```

profiles

Personaliza la construcción para diferentes entornos.

```
<profiles>
  <profile>
    <id>development</id>
    <properties>
      <db.url>jdbc:h2:mem:dev</db.url>
    </properties>
  </profile>
  <profile>
    <id>production</id>
    <properties>
      <db.url>jdbc:mysql://prod-server/db</db.url>
    </properties>
  </profile>
</profiles>
```

Activación: `mvn install -Pproduction`

Ciclo de Vida y Objetivos

Maven Lifecycle

Automatización del proceso de construcción

Ciclo de Vida de Maven

Fases Principales

- validate: Validar estructura del proyecto
- compile: Compilar código fuente
- test: Ejecutar pruebas unitarias
- package: Empaquetar código compilado
- verify: Ejecutar pruebas de integración
- install: Instalar en repositorio local
- deploy: Copiar a repositorio remoto

Consejo

Cada fase incluye todas las fases anteriores. Por ejemplo, `mvn install` ejecutará: validate, compile, test, package, verify e install.

Objetivos Principales (Goals)

clean

Elimina archivos generados en compilaciones anteriores

```
mvn clean
```

Elimina el directorio `target/` y todos sus contenidos

package

Empaqueta código compilado en formato distribuible

```
mvn package
```

Crea archivos JAR, WAR o EAR según la configuración

install

Instala el paquete en repositorio local para uso como dependencia

```
mvn install
```

Permite que otros proyectos usen este como dependencia

debug

Ejecuta Maven en modo de depuración

```
mvnDebug [goal]
```

Activa puerto de depuración para conectar un debugger

Directorio Target

Maven Output

Salida de la construcción y artefactos generados

¿Qué es el directorio target?

Es el directorio donde Maven almacena todos los **artefactos compilados** y **salidas generadas** durante el proceso de construcción.

Características principales:

- Temporal: Se crea durante el build
- Eliminable: Se borra con `mvn clean`
- No versionado: Debe excluirse de Git/SVN
- Reconstruible: Todo su contenido puede regenerarse

Buenas Prácticas

- Nunca incluir `/target` en control de versiones
- No modificar manualmente archivos en `/target`
- Ejecutar `mvn clean` antes de compartir código
- Añadir `target/` a tu archivo `.gitignore`

Estructura del Directorio

```
target/
  classes/              → Archivos .class compilados
  test-classes/         → Tests compilados
  generated-sources/
  surefire-reports/     → Resultados de tests
  maven-archiver/
  myapp-1.0.jar         → Artefacto final
  myapp-1.0-sources.jar → Código fuente
  myapp-1.0-javadoc.jar → Documentación
```

Archivos Importantes

.jar / .war / .ear
Artefacto principal que contiene la aplicación compilada lista para distribución

***-sources.jar**
Contiene el código fuente, útil para usuarios de tu biblioteca

***-javadoc.jar**
Documentación generada a partir de los comentarios Javadoc

surefire-reports/*.xml
Informes XML de los resultados de las pruebas unitarias

Limpieza

Para eliminar el directorio `target` y todos sus contenidos, ejecuta:

```
$ mvn clean
```

Archivos de Configuración

Maven Config

Personalización del comportamiento de Maven

Archivos de Configuración Principales

settings.xml

Ubicaciones:
- Global: `${maven.home}/conf/`
- Usuario: `${user.home}/.m2/`

Configura:
- Repositorios
- Servidores (credenciales)
- Mirrors
- Proxies

```
<settings>
  <servers>
    <server>
      <id>my-server</id>
      <username>user</username>
      <password>pass</password>
    </server>
  </servers>
</settings>
```

toolchains.xml

Configuración de herramientas específicas del sistema (JDK, compiladores).

Ubicaciones:
- Global: `${maven.home}/conf/`
- Usuario: `${user.home}/.m2/`

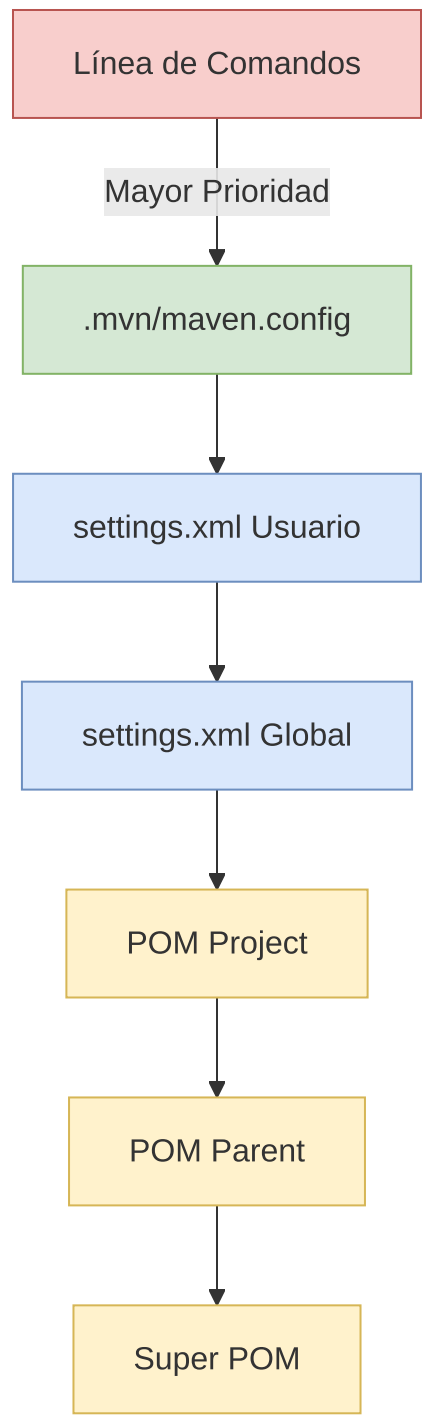
Configura:
- Versiones de JDK
- Versiones de Maven
- Herramientas específicas

Directorio .mvn/

Configuración a nivel de proyecto, incluida con el código fuente.

Archivos comunes:
- `maven.config` - Opciones de línea de comandos predeterminadas
- `extensions.xml` - Extensiones para cargar
- `jvm.config` - Opciones de JVM

Orden de Precedencia



Regla de Prioridad

Las configuraciones más específicas tienen mayor prioridad. Una configuración de nivel superior sobrescribe una configuración de nivel inferior.

Otros Archivos de Configuración

Archivos de Propiedades

`*.properties` en distintas ubicaciones:
- `src/main/resources`
- `src/test/resources`
- Filtrado en builds

```
# application.properties
app.name=Mi Aplicación
app.version=${project.version}
db.url=jdbc:mysql://localhost/db
```

extensions.xml

Permite cargar extensiones de Maven antes de que se procese el POM.

```
<extensions>
  <extension>
    <groupId>org.example</groupId>
    <artifactId>custom-extension</artifactId>
    <version>1.0</version>
  </extension>
</extensions>
```

Mejores Prácticas

- Usar variables de propiedades para mantener la configuración consistente
- Definir perfiles específicos para distintos entornos (dev, test, prod)
- Utilizar herencia de POM para compartir configuraciones entre proyectos
- Mantener las credenciales en el `settings.xml` de usuario (no en el POM)