MAVEN

PAR ELIAS ZGHEIB

Table des matières

1. Introduction Maven

Maven est un outil de construction de projets (build) open source développé par la fondation Apache et surtout les projets Java et Java EE en particulier.
Il permet de faciliter et d'automatiser certaines tâches de la gestion d'un projet :

- Créer une arborescence standard du code et de ses ressources(convention)
- télécharger, mettre à jour et configurer les bibliothèques nécessaires au projet
- compiler le projet, effectuer des tests unitaires sur le code et packager le résultat.
- Gestion du cycle de vie d'un projet
- générer des documentations concernant le projet

Le site web officiel de MAVEN est http://maven.apache.org

Maven ne se limitent pas à la manipulation de projets Java. Il existe des plugins Maven dédié au support d'autres technologies que le Java. C'est le cas notamment de PHP , .net , Ruby , Flex

2. Histoire

Plusieurs versions majeures de Maven ont été diffusées :

- MAVEN 1 (2003)
- MAVEN 2 (2005)
- MAVEN 3 (2010)

Maven 2 est très différent de Maven 1.

Maven 3 est compatible avec Maven 2 et apporte notamment de meilleures performances.

3. Concepts

Maven repose sur l'utilisation de plusieurs concepts :

- Les artéfacts : composants identifiés de manière unique
- Le principe de convention over configuration : utilisation de conventions par défaut pour standardiser les projets

- Le cycle de vie et les phases : les étapes de construction d'un projet sont standardisées
- Les dépôts (local et distant)

3.0 LES ARTEFACTS

Un artéfact est un composant packagé possédant un identifiant unique composé de trois éléments : un groupId, un artifactId et un numéro de version.

```
<groupId>com.presentation</groupId>
<artifactId>maven-demo</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Les versions standard correspondent à des releases de l'artéfact.

Les versions en cours de développement se terminent par -SNAPSHOT : ce sont des versions intermédiaires de travail en local.

3.1 CYCLE DE VIE

La construction d'un projet Maven repose sur la notion de cycle de vie qui définit les différentes étapes de la génération de l'artéfact nommées phases.

Trois cycles de vie sont définis :

default : il permet de générer et déployer l'artéfact du projet

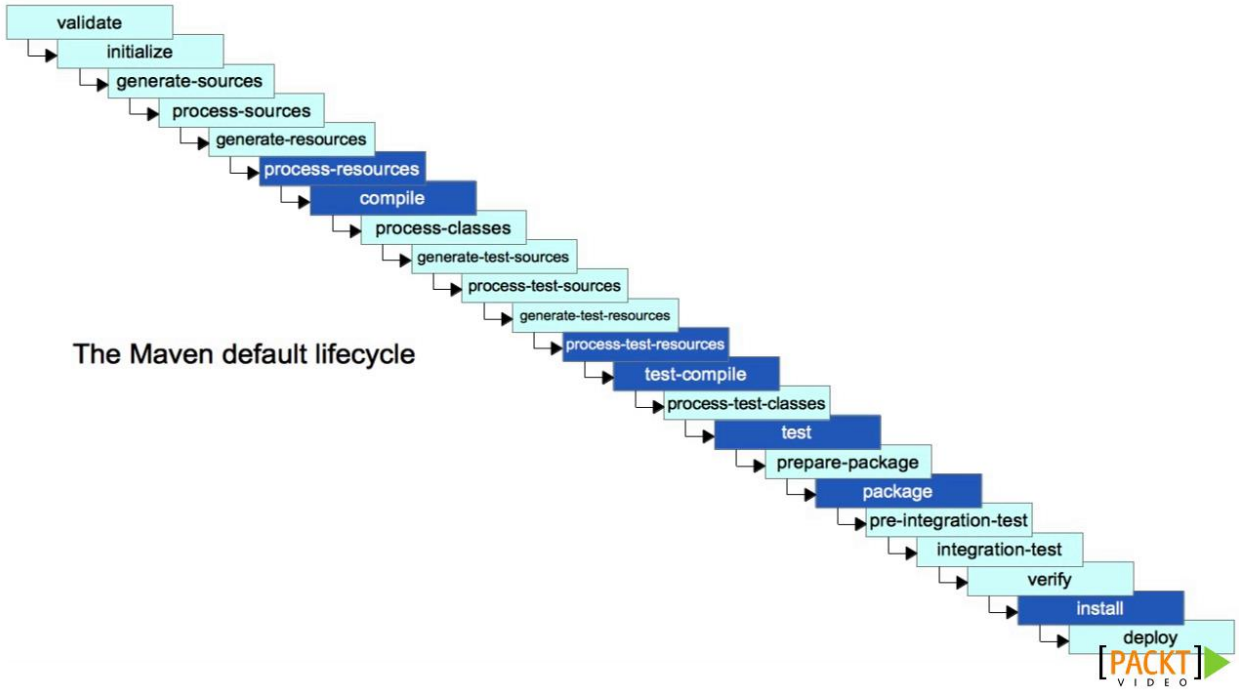
clean : il permet de nettoyer les fichiers générés du projet

site : il permet de générer un site web pour accéder à la documentation du projet

Le cycle de vie par défaut (build) de Maven contient plusieurs phases dont les principales sont : validate, compile, test, package, install et deploy :

Phase	Rôle
Validate	Valider les informations nécessaires à la génération du projet
Initialize	Initialiser la génération du projet

generate-sources	Générer les sources qui doivent être incluses dans la compilation
process-sources	Traiter des sources (par exemple, application d'un filtre)
generate-resources	Générer les ressources qui doivent être incluses dans l'artéfact
process-resources	Copier les ressources dans le répertoire target
Compile	Compiler le code source du projet
process-classes	Traiter les fichiers class résultant de la compilation (par exemple, pour faire du bytecode enhancement)
generate-test-sources	Générer des sources qui doivent être incluses dans les tests
process-test-sources	Traiter les sources pour les tests (par exemple, application d'un filtre)
generate-test-resources	Générer des ressources qui doivent être incluses dans les tests
process-test-resources	Copier les ressources dans le répertoire test
test-compile	Compiler le code source des tests
process-test-classes	Effectuer des actions sur les classes compilées (par exemple, pour faire du bytecode enhancement)
Test	Compiler et exécuter les tests unitaires automatisés. Ces tests ne doivent pas avoir besoin de la forme diffusable de l'artéfact ni de son déploiement dans un environnement
Prepare-package (à partir de Maven 2.1)	Effectuer des actions pour préparer la génération du package
Package	Générer l'artéfact sous sa forme diffusable (jar, war, ear, ...)
pre-integration-test	Effectuer des actions avant l'exécution des tests d'intégration (par exemple configurer l'environnement d'exécution)
integration-test	
(à partir de Maven 3)	Compiler et exécuter les tests d'intégration automatisés dans un environnement dédié au besoin en effectuant un déploiement
post-integration-test	Effectuer des actions après l'exécution des tests d'intégration (par exemple faire du nettoyage dans l'environnement)
Verify	Exécuter des contrôles de validation et de qualité
Install	Installer l'artéfact dans le dépôt local pour qu'il puisse être utilisé comme dépendance d'autres projets
Deploy	Déployer l'artéfact dans un environnement dédié et copier de l'artéfact dans le référentiel distant



Ces différentes phases sont exécutées séquentiellement de la première étape jusqu'à la phase demandée à Maven. Toutes les phases jusqu'à la phase demandée seront exécutées.

Exemple :

mvn deploy
mvn integration-test

Une phase est composée d'un ou plusieurs goals. Un goal est une tâche spécifique à la génération ou la gestion du projet.

Exemple :

mvn clean dependency:copy-dependencies package

Dans l'exemple ci-dessus, le cycle de vie clean est exécutée, puis le goal dependency :copy-dependencies puis la phase package avec toutes les phases précédentes du cycle de vie par défaut.

Si le goal est défini dans plusieurs phases alors le goal de chacune des phases sera invoqué.

Le cycle de vie clean de Maven contient plusieurs phases :

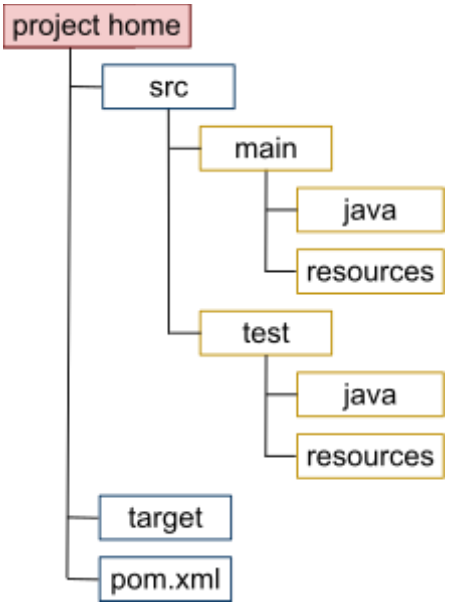
Phase	Rôle
pre-clean	Exécuter des traitements avant de nettoyer le projet
Clean	Supprimer tous les fichiers par le build précédent
post-clean	Exécuter des traitements pour terminer le nettoyage du projet

Le cycle de vie site de Maven contient plusieurs phases :

Phase	Rôle
pre-site	Exécuter des traitements avant la génération du site
Site	Générer le contenu du site de documentation du projet
post-site	Exécuter des traitements après la génération du site
site-deploy	Déployer le site sur un serveur web

3.2 CONVENTION PLUTOT QUE CONFIGURATION

Maven met en œuvre le principe de convention over configuration pour utiliser par défaut les mêmes conventions.



Par exemple, l'arborescence d'un projet Maven est par défaut imposée par Maven :

Répertoire	Contenu
/src	les sources du projet (répertoire qui doit être ajouté dans le gestionnaire de sources)
/src/main	les fichiers sources principaux

/src/main/java	le code source (sera compilé dans /target/classses)
/src/main/resources	les fichiers de ressources (fichiers de configuration, images, ...). Le contenu de ce répertoire est copié dans target/classes pour être inclus dans l'artéfact généré
/src/main/webapp	les fichiers de la webapp
/src/test	les fichiers pour les tests
/src/test/java	le code source des tests (sera compilé dans /target/test-classses)
/src/test/resources	les fichiers de ressources pour les tests
/target	les fichiers générés pour les artéfacts et les tests (ce répertoire ne doit pas être inclus dans le gestionnaire de sources)
/target/classes	les classes compilées
/target/test-classes	les classes compilées des tests unitaires
/target/site	site web contenant les rapports générés et des informations sur le projet
/pom.xml	le fichier POM de description du projet

L'utilisation de ces conventions est un des points forts de Maven car elle permet aux développeurs de facilement être familiarisés avec la structure des projets qui est toujours la même.

Pour des besoins particuliers, il est possible de configurer une autre structure de répertoires mais cela n'est pas recommandé

Les archétypes

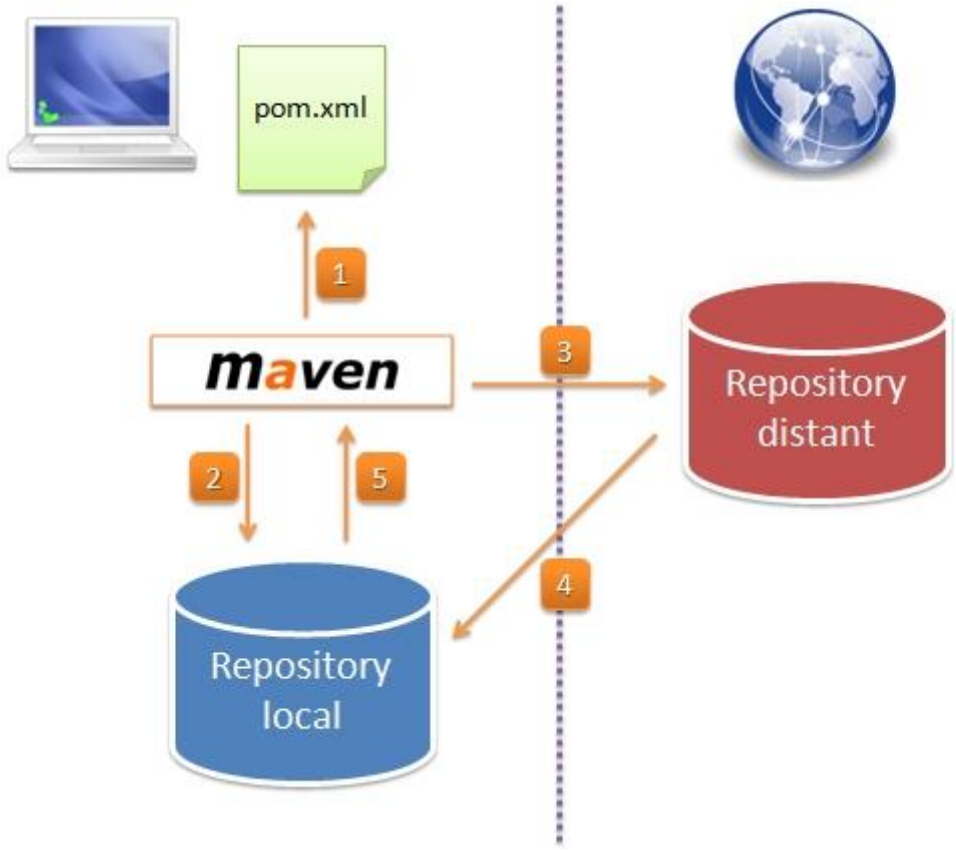
Un archétype est un modèle de projet. Maven propose en standard plusieurs archétypes dont les principaux sont :

Archétype	Description
maven-archetype-archetype	Un archétype pour un exemple d'archétype
maven-archetype-j2ee-simple	Un archétype pour un projet de type application J2EE simplifié
maven-archetype-mojo	Un archétype pour un exemple de plugin Maven
maven-archetype-plugin	Un archétype pour un projet de type plugin Maven
maven-archetype-plugin-site	Un archétype pour un exemple de site pour un plugin Maven
maven-archetype-portlet	Un archétype pour un projet utilisant les portlets
maven-archetype-quickstart	Un archétype pour un exemple de projet Maven
maven-archetype-simple	Un archétype pour un simple projet Maven
maven-archetype-site	Un archétype pour un site Maven
maven-archetype-site-simple	Un archétype pour un site Maven simplifié

maven-archetype-webapp	Un archétype pour un projet de type application web

D'autres achétypes peuvent être proposés par des tiers. Il est aussi possible de développer ses propres archétypes.

3.3 LES DEPOTS (REPOSITORIES)



Maven télécharge beaucoup de fichiers à partir de dépôts distants (les dépendances des projets, les plugins et les dépendances de ces plugins…). Tous ces éléments téléchargés sont stockés dans le dépôt local pour n'avoir à faire cette opération de téléchargement qu'une seule fois tant que l'élément est contenu dans le dépôt local.

Pour gérer les dépendances du projet vis-à-vis de bibliothèques, Maven utilise un ou plusieurs dépôts. Ces dépôts peuvent être locaux (local repository) à la machine ou accessibles via HTTP(remote). Un dépôt contient des livrables, des dépendances, des plugins, ... Ceci permet de centraliser ces éléments qui sont généralement utilisés dans plusieurs projets.

Ainsi, à sa première exécution, Maven télécharge les différents plugins dont il a besoin et les installe dans le répertoire local de l'utilisateur.

Ces mêmes librairies peuvent être réutilisées entre les différents projets.

- dépôt local : il stocke une copie des dépendances et plugins requis par les projets à générer en local. Ces artéfacts sont soit téléchargés des dépôts centraux soit créés avec Maven.
- dépôt central : il stocke des dépendances et les plugins utilisables par tout le monde car disponible sur le web; ce sont généralement des artéfacts open source.

Maven utilise une structure de répertoires particulière pour organiser le contenu d'un référentiel et lui permettre de retrouver les éléments requis :

Chemin_referentiel/groupId/artifactId/version

Par exemple avec Junit 3.8.2, dont les identifiants sont :

<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.2</version>

La structure de répertoires dans le dépôt est :

\junit\junit\3.8.2

Le répertoire de la version contient au moins le projet et son POM mais il peut aussi éventuellement contenir d'autres fichiers liés contenant une archive avec les sources, la Javadoc, la valeur du message digest calculée avec SHA-1, ...

Maven recherche un élément dans un ordre précis dans les différents référentiels :

- tout d'abord dans le dépôt local
- puis un des dépôts distant configuré s'il n'est pas trouvé

Si un élément n'est pas trouvé dans le répertoire local, il sera téléchargé dans ce dernier à partir du premier dépôt distant dans lequel il est trouvé.

Par défaut, le dépôt local est contenu dans le sous-répertoire .m2\repository du répertoire personnel de l'utilisateur.

Le dépôt central de Maven par défaut est à l'url http://repo.maven.apache.org/maven2/

Il peut être utile d'utiliser un autre dépôt pour plusieurs raisons : il est plus prêt géographiquement, il possède des artéfacts qui ne sont pas disponibles dans le dépôt par défaut, ...

Des miroirs publics du dépôt par défaut sont listés dans un lien de la page http://maven.apache.org/guides/mini/guide-mirror-settings.html.

Il est aussi possible de créer un dépôt intermédiaire gérer par un gestionnaire d'artéfacts qui va contenir une copie des artéfacts requis. L'utilisation d'un gestionnaire d'artéfacts possède plusieurs avantages :

- limiter l'utilisation de la bande passante sur internet et ainsi améliorer les performances
- restreindre les artéfacts utilisés uniquement à ceux présents dans le dépôt

Pour remplacer ou ajouter un autre dépôt, il faut modifier le fichier de configuration settings.xml tel que chaque dépôt est défini dans un tag <mirror>.

Exemple :

```
01.<settings>
02....
03.<mirrors>
04.<mirror>
05.<id>UK</id>
06.<name>UK Central</name>
07.<url>http://uk.maven.org/maven2<;/url>
08.<mirrorOf>central</mirrorOf>
09.</mirror>
10.</mirrors>
11....
12.</settings>
```

La valeur centrale du tag <mirrorOf> permet de préciser que le dépôt est un miroir du dépôt central. Maven va alors l'utiliser de préférence au dépôt central.

4. LE POM

MAVEN fournit des moyens de configuration très simple; il est basé sur le concept du Project Object Management POM.

Chaque projet ou sous-projet est configuré par un POM qui se matérialise par un fichier XML pom.xml.
Lorsqu'on exécute une commande MAVEN, il consulte ce fichier.
POM contient les informations nécessaires à Maven pour traiter le projet et les détails de configuration utilisées par MAVEN pour faire le build:

- nom du projet, numéro de version,
- dépendances vers d'autres projets,
- bibliothèques nécessaires à la compilation, plugins,
- etc...

Le fichier POM doit être à la racine du répertoire du projet.

Le tag racine est le tag <project>

File : pom.xml

```
<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/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.javatpoint.application1</groupId>
  <artifactId>my-application1</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

Le tag <projet> peut avoir plusieurs tags fils dont les principaux sont :

Tag	Rôle
<modelVersion>	Préciser la version du modèle de POM utilisée
<groupId>	Préciser le groupe ou l'organisation qui développe le projet. C'est une des clés utilisée pour identifier de manière unique le projet et ainsi éviter les conflits de noms
<artifactId>	Préciser la base du nom de l'artéfact du projet
<packaging>	Préciser le type d'artéfact généré par le projet (jar, war, ear, pom, ...). Le packaging définit aussi les différents goals qui seront exécutés durant le cycle de vie par défaut du projet. La valeur par défaut est jar

<version>	Préciser la version de l'artéfact généré par le projet. Le suffixe -SNAPSHOT indique une version en cours de développement
<name>	Préciser le nom du projet utilisé pour l'affichage
<description>	Préciser une description du projet
<url>	Préciser une url qui permet d'obtenir des informations sur le projet
<dependencies>	Définir l'ensemble des dépendances du projet
<dependency>	Déclarer une dépendance en utilisant plusieurs tags fils : <groupId>, <artifactId>, <version> et <scope>
<scope>	Préciser le scope du projet: compile, provided, runtime, test and system.

Les informations d'un POM sont héritées d'un POM parent sauf le POM racine.

5. LE FICHIER SETTINGS.XML

Le fichier settings.xml contient la configuration globale de Maven. C'est un document XML dont le tag racine est le tag <settings>.

Plusieurs tags fils permettent de préciser les éléments de la configuration :

Tag	Rôle
localRepository	Préciser le chemin du dépôt local. Par défaut, c'est le sous-répertoire .m2/repository du répertoire de l'utilisateur
interactiveMode	Utiliser Maven en mode interactif pour lui permettre d'obtenir des informations lorsqu'elles sont requises. Par défaut : true.
offline	Utiliser Maven en mode offline, donc déconnecter du réseau. Par défaut : false.
pluginGroups	
proxies	Définir un ou plusieurs proxy. Chaque proxy est défini dans un tag <proxy>
servers	Fournir des informations d'authentification sur une ressource. Chaque ressource est définie dans un tag <server>
mirrors	Définir des dépôts distants qui sont des miroirs. Chaque miroir est défini dans un tag <mirror>
profiles	Définir des profiles qui pourront être activés. Chaque profile est défini dans un tag <profile>
ActiveProfiles	Définir la liste des profiles qui sont activés par défaut pour tous les builds

La configuration de proxys se fait dans un tag <proxies>. Chaque proxy est défini dans un tag <proxy> qui possède plusieurs tags fils :

Tag	Rôle
<id>	L'identifiant du proxy
<protocol>	Le protocole utilisé pour accéder au proxy
<active>	
<username>	Le nom de l'utilisateur
<password>	Le mot de passe de l'utilisateur
<host>	Le host name du proxy
<port>	Le port du proxy
<nonProxyHosts>	énbsp;

6. LES PLUGINS

Toutes les fonctionnalités pour générer un projet sont sous la forme de plugins qui doivent être présents dans le référentiel local ou téléchargés lors de la première utilisation.

La déclaration et la configuration des plugins à utiliser se fait dans le fichier POM.

Certains plugins utilisés dans le cycle de vie par défaut n'ont pas besoin d'être définit explicitement dans le fichier POM, sauf si la configuration par défaut doit être modifiée.

Il est aussi possible de développer ses propres plugins.

Exemple :

```
01....
02.<build>
03.<plugins>
04.<plugin>
05.<groupId>org.apache.maven.plugins</groupId>
06.<artifactId>maven-compiler-plugin</artifactId>
07.<version>2.0.2</version>
08.<configuration>
09.<source>1.5</source>
10.<target>1.5</target>
11.</configuration>
12.</plugin>
13.</plugins>
14.</build>
15....
```

Les plugins sont des dépendances qui sont stockées dans le dépôt local après avoir été téléchargées.