PRESENTATION DE MAVEN

INTRODUCTION

Apache Maven (couramment appelé Maven) est un outil de gestion et d'automatisation de production des projets logiciels Java en général et Java EE en particulier. Il est utilisé pour automatiser l'intégration continue lors d'un développement de logiciel. *Maven* est géré par l'organisation *Apache Software Foundation*. L'outil était précédemment une branche de l'organisation *Jakarta Project*.

L'objectif recherché est de produire un logiciel à partir de ses sources, en optimisant les tâches réalisées à cette fin et en garantissant le bon ordre de fabrication.

Il peut se comparer au système make sous Unix ou à l'outil Ant.

Maven utilise un paradigme connu sous le nom de *Project Object Model* (POM) afin de décrire un projet logiciel, ses dépendances avec des modules externes et l'ordre à suivre pour sa production. Il est livré avec un grand nombre de tâches prédéfinies, comme la compilation de code Java ou encore sa modularisation.

Un élément clé et relativement spécifique de *Maven* est son aptitude à fonctionner en réseau. Une des motivations historiques de cet outil est de fournir un moyen de synchroniser des projets indépendants : publication standardisée d'information, distribution automatique de modules jar. Ainsi en version de base, *Maven* peut dynamiquement télécharger du matériel à partir des *dépôts* logiciels connus. Il propose ainsi la synchronisation transparente de modules nécessaires.

Maven1 et Maven2 ont été développés en parallèle mais les versions ultérieures sont basées sur la structure de la deuxième version. Les parties suivantes de l'article traitent en priorité Maven2. Une version 3 de Maven est sortie le 8 octobre 2010. La fin de support de la version 2 a été actée le 18 février 2014

OBJECTIFS

L'objectif principal de Maven est de permettre à un développeur de comprendre l'état complet d'un effort de développement dans les plus brefs délais. Afin d'atteindre cet objectif, Maven tente de traiter plusieurs domaines de préoccupation :

- ❒Rendre le processus de construction facile
- ❒Fournir un système de construction uniforme
- ❒Fournir des informations de qualité sur les projets
- ❒Fournir des directives pour le développement des meilleures pratiques
- ❒Permettre une migration transparente vers de nouvelles fonctionnalités

Projet Object Model (POM)

Chaque projet ou sous-projet est configuré par un POM qui contient les informations nécessaires à Maven pour traiter le projet (nom du projet, numéro de version, dépendances vers d'autres projets, bibliothèques nécessaires à la compilation, noms des contributeurs, etc.). Ce POM se matérialise par un fichier `pom.xml` à la racine du projet. Cette approche permet l'héritage des propriétés du projet parent. Si une propriété est redéfinie dans le POM du projet, elle recouvre celle qui est définie dans le projet parent. Ceci introduit le concept de réutilisation de configuration. Le fichier pom du projet principal est nommé *pom parent*. Il contient une description détaillée de votre projet, avec en particulier des informations concernant le versionnage et la gestion des configurations, les dépendances, les ressources de l'application, les tests, les membres de l'équipe, la structure et bien plus

Arborescence d'un projet maven

Maven impose une arborescence et un nommage des fichiers du projet selon le concept de Convention plutôt que configuration. Ces conventions permettent de réduire la configuration des projets, tant qu'un projet suit les conventions. Si un projet a besoin de s'écarter de la convention, le développeur le précise dans la configuration du projet.

Voici une liste non exhaustive des répertoires d'un projet Maven :

- `/src` : les sources du projet
- `/src/main` : code source et fichiers source principaux
- `/src/main/java` : code source
- `/src/main/resources` : fichiers de ressources (images, fichiers annexes, etc.)
- `/src/main/webapp` : webapp du projet
- `/src/test` : fichiers de test
- `/src/test/java` : code source de test
- `/src/test/resources` : fichiers de ressources de test
- `/src/site` : informations sur le projet et/ou les rapports générés suite aux traitements effectués
- `/target` : fichiers résultat, les binaires (du code et des tests), les packages générés et les résultats des tests

Cycle de vie

Les buts (*goals* en anglais) principaux du cycle de vie d'un projet Maven sont:

- `compile`
- `test`
- `package`
- `install`
- `deploy`

L'idée est que, pour n'importe quel but, tous les buts en amont doivent être exécutés sauf s'ils ont déjà été exécutés avec succès et qu'aucun changement n'a été fait dans le projet depuis. Par exemple, quand on exécute `mvn install`, Maven va vérifier que `mvn package` s'est terminé avec succès (le jar existe dans `target/`), auquel cas cela ne sera pas réexécuté.

D'autres buts sont exécutables en dehors du cycle de vie et ne font pas partie du cycle de vie par défaut de Maven (ils ne sont pas indispensables). Voici les principaux :

- `clean`
- `assembly:assembly`
- `site`
- `site-deploy`
- etc.

Ils peuvent néanmoins être rajoutés au cycle de vie via le POM.

Gestion des dépendances

Dans l'outil Maven, la gestion des dépendances s'appuie sur deux notions :

- l'héritage
- les relations transitives (ou transitivité)

La notion de relation transitive est implémentée pour la relation "dépend de" et l'ensemble projet Java.

Exemple : Soit trois projets (A, B, C), si A dépend de B, et B dépend de C, alors A dépendra de C.

Par défaut dans Maven, la transitivité est configurée en automatique. Cette configuration peut générer des contraintes avec les projets volumineux :

- L'inclusion de jars dans les EAR
- L'inclusion de versions non-désirées
- La présence de dépendances pour différentes versions à différents niveaux de la transitivité (un projet A dépend d'un projet C.x et d'un projet B, et ce dernier dépend du projet C.y)
- Une perte d'informations (des dépendances effectives, par exemple) et la présence de dépendances inutiles à d'autres projets.