DevOps

Maven

Thomas Ropars

thomas.ropars@univ-grenoble-alpes.fr

2020

Maven

Ce que c'est:

- Outil pour la gestion et l'automatisation de production de projets logiciels
- Cible principalement Java et en particulier les applications Java EE

Motivations

Limites de Ant

- Pas de structure *standard* de projet
  - Nouveau `build.xml` à écrire pour chaque projet
  - Arriver dans un nouveau projet peut être difficile (absence de conventions)
  - Que se passe-t-il si plusieurs équipes de développement utilisent des conventions différentes?
- Pas de cycle de vie *standard* de projet
  - Définition manuelle des cibles et des dépendances
- Gestion *manuelle* des bibliothèques dont dépendent le projet
  - Problème de la mise à jour des versions

pom.xml pour le TP liste chaînée

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.uga.erods</groupId>
  <artifactId>my-list-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>my-list-app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

L'approche Maven

Convention plutôt que configuration

- Par défaut, tous les projets se ressemblent
  - Initialiser, Compiler, Tester, Assembler, ...
- Maven définit une structure de projet par défaut
  - Ensemble de conventions *raisonnables*
  - Il faut préciser ce qui ne suit pas les conventions

Décrire plutôt que programmer

- Approche déclarative
- On indique les particularités du projet et non la manière de le construire

POM

- Project Object Model
- Fichier xml: `pom.xml`
- `modelVersion` définit le modèle de structure de projet.
  - `4.0.0` est le modèle par défaut
- `pom.xml` étend en fait un fichier super-POM défini dans Maven
  - On peut aussi avoir des `pom.xml` parents

Les concepts de Maven

Plugin

- Fragment de logiciel qui se spécialise dans une tâche donnée
- Ex: compilation, tests, ...

Goals (Tâches)

- Un plugin peut exécuter un ensemble de goals (tâches unitaires)
- Ex: compile du plugin Compiler, test du plugin surefire, ...

Création d'un projet

Plugin `archetype`[1]

- Permet à l'utilisateur de créer un projet à partir d'un template
- Création du projet pour une liste chaînée
  - `mvn archetype:generate -DgroupId=fr.uga.erods -DartifactId=my-list-app -DarchetypeArtifactId=maven-archetype-quickstart`
  - `archetype` est l'identifiant du *plugin*
  - `generate` est l'identifiant du *goal*
- Exemple de passage de paramètres au plugin via la ligne de commande
  - Sans l'option `maven-archetype-quickstart`, Maven nous aurait demandé quel type de projet créer.

[1] `http://maven.apache.org/archetype/maven-archetype-plugin/`

Structure standard des fichiers[1]

- src/main/java: sources de l'Application/Library
- src/main/resources: ressources de l'Application/Library
- src/test/java: sources des tests
- src/test/resources: ressources des tests
- src/site: Site web
- LICENSE.txt
- NOTICE.txt
- README.txt
- pom.xml

[1] http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

La section build

<build> ...</build>

- voir
  https://maven.apache.org/pom.html#Build_Settings

- Section permettant de déclarer des proriétés générales de votre projet

```
<build>
  <defaultGoal>install</defaultGoal>
  <directory>${basedir}/target</directory>
  ...
</build>
```

  - defaultGoal definie la phase par défaut à exécuter
  - directory définit le répertoire où les fichier générés seront stockés

La section `build`

- Section permettant également de déclarer et de configurer les plugins
- Exemple de configuration pour le compilateur java.
  - Par défaut `source` et `target` sont fixés à 1.5

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>7</source>
                <target>7</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Cycle de vie

Pour identifier et enchaîner les tâches de base dans un projet, Maven se base sur:

- Les plugins et les tâches associées
- Un cycle de vie

Cycle de vie

- Série de phases ordonnées
- Définit les étapes clés de la construction du projet
- 3 cycles de vie prédéfinis:
  - default: *construire* votre projet
  - clean: *nettoyage* du projet
  - site: création de la documentation du projet

Cycle de vie par défaut

Les phases principales du cycle de vie par défaut[1]:

- validate: valide que le projet est correct et que toutes les infos nécessaires sont disponibles
- compile
- test
- package: package les sources compilées dans un format distribuable (par ex JAR)
- integration-test
- verify: Lance les tests pour vérifier la qualité du package
- install: Installe le package dans le dépôt local
- deploy: Copie le package final dans un dépôt distant pour le partager

[1] Pour une liste complète voir: `http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html`

Cycle de vie

Demander l'exécution d'une phase d'un cycle entraîne l'exécution de toutes les phases précédentes.

- `mvn deploy`
  - Exécute toutes les phases du cycle par défaut
- `mvn clean install`
  - Exécute la phase clean (et précédentes) puis install (et précédentes)

Qu'est ce qui est exécuté par une phase?

- Les tâches qui lui ont été associées

Associer des tâches à des phases

Définir le packaging de son projet.

- `<packaging>jar</packaging>`
- Le packaging associe des tâches au phases du cycle par défaut.
- Le packaging par défaut est `jar` (autres: ejb, ear, war, ...)

Associer des tâches à des phases

Définir le packaging de son projet.

- `<packaging>jar</packaging>`
- Le packaging associe des tâches au phases du cycle par défaut.
- Le packaging par défaut est `jar` (autres: ejb, ear, war, ...)

Tâches associées par le packaging `jar`:

phase	**tâche**
process-resources	resources:resources
compile	compiler:compile
process-test-resources	resources:testResources
test-compile	compiler:testCompile
test	surefire:test
package	jar:jar
install	install:install
deploy	deploy:deploy

Associer une tâche d'un plugin à une phase

```
...
 <plugin>
   <groupId>com.mycompany.example</groupId>
   <artifactId>display-maven-plugin</artifactId>
   <version>1.0</version>
   <executions>
     <execution>
       <phase>test</phase>
       <goals>
         <goal>time</goal>
       </goals>
     </execution>
   </executions>
 </plugin>
...
```

- La tâche `display:time` sera executée dans la phase `test`
- Ordre d'exécution des tâches:
  - Celles définies par le packaging en premier
  - Puis exécution selon l'ordre d'apparition dans le POM

Exemple: plugin de couverture de code EclEmma[1]

```
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.7.6.201602180812</version>
  <executions>
    <execution>
      <goals><goal>prepare-agent</goal></goals>
    </execution>
    <execution>
      <id>default-report</id>
      <phase>prepare-package</phase>
      <goals><goal>report</goal></goals>
    </execution>
  </executions>
</plugin>
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.14.1</version>
  <configuration>
    <argLine>${argLine}</argLine>
  </configuration>
</plugin>
```

[1] http://eclemma.org/jacoco/trunk/doc/maven.html

Coordinates (coordonnées)

- Le fichier POM fournit un ensemble d'identifiants uniques du projet:

```
<groupId>fr.uga.erods</groupId>
<artifactId>my-list-app</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
```

- `groupId:artifactId:packaging:version` identifie de manière unique le projet
  - `groupId`: identifie l'entité qui gère le projet
  - `artifactId`: identifie le projet
  - `version`: Numéro de version du projet
    - `SNAPSHOT`: Mot clé indiquant à Maven que le projet est en cours de développement

Coordinates (coordonnées)

- Les dépôts maven (publics, privés, locaux) sont organisés autour de ces coordonnées
- Lorsque qu'un projet est *installé* localement, il devient disponible pour tout autre projet
- Il suffit de déclarer une dépendance en utilisant les coordonnées de l'artefact.

Les dépôts Maven

- Un dépôt Maven stocke des artefacts:
  - Stocke un ensemble d'artefacts de projet rangés selon une structure de répertoires correspondant aux coordonnées Maven
  - Dépot distant par défaut: `http://repo.maven.apache.org/maven2/`
  - Dépôt local par défaut: `$HOME/.m2/repository`
  - Possibilité d'ajouter des dépôts
  - Possibilité de créer des dépôts privés

Les dépôts Maven

- Les plugins et dépendances sont obtenues depuis les dépôts
  - Si un artefact n'est pas dans le dépôt local, recherche dans le dépôt distant
  - Stockage dans le dépôt local pour résolution locale lors du prochain appel
  - Attention: Grand nombre de téléchargements lors des premières utilisations
  - `mvn install` installe le projet dans le dépôt local
- Si un artefact a le tag `SNAPSHOT`, vérification à chaque appel qu'une version plus récente n'est pas disponible sur le dépôt distant

Les dépendances

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- Définition des dépendances dans la section `dependencies`
- Maven gère les dépendances transitives
- Les dépendances ont une portée (`scope`)

Résolution transitive des dépendances

À chaque artefact est associé un fichier `pom.xml` dans lequel est défini ses dépendances.

- Si mon projet dépend de A et que A dépend de B, mon projet dépend de B
- Maven installera automatiquement B

Remarques en plus

- Médiation si conflit entre dépendances (*nearest definition*)
- Contrôle de la version des dépendances transitives par déclaration explicite
- Utilisation de la balise `<exclusion>` pour exclure des dépendances

Portée des dépendances

Chaque dépendance a une portée (`<scope>`):

- Permet de couper l'arbre des dépendances
- Influence le `classpath` utilisé dans chaque phase

6 portées possibles:

- compile (portée par défaut)
  - Disponibles dans tous les classpaths
  - Dépendances propagées aux projets dépendants (transitivité)
- provided
  - Classpath pour la compilation et les tests
  - Pas de transitivité
  - Dépendance résolue pour la compilation et le test mais supposée déjà disponible dans le contexte d'exécution
- runtime
  - Classpath pour les tests et au runtime

Portée des dépendances

- test
  - Classpath pour la compilation et l'exécution des tests
- system
  - Similaire à provided
  - La dépendance doit être fournie (pas de résolution par les dépôts)
- import
  - Utilisé dans le contexte de la balise `<dependencyManagement>`

Chaque scope affecte les dépendances transitives de manière différente[1].

[1] `http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html`

A propos des dépendances et des dépôts

Simplification de la gestion des bibliothèques:

- Automatisation de la gestion des dépendances
- Vérification de l'intégrité des bibliothèques téléchargées (hachage)
- Niveau de confiance dans les bibliothèques fournies par un dépôt public

Hiérarchie de projets[1]: Héritage

- Utilisation: Factorisation de plusieurs projets avec des configurations similaires
- Définition d'un POM parent
- Modèle objet: un POM hérite des attributs de son parent sauf si il les redéfinit
  - Les dépendances
  - Les plugins
  - Configuration des plugins
  - ...
- Arborescence de fichiers
  - `pom.xml` (parent)
  - `my-module/pom.xml`

[1] `http://maven.apache.org/guides/introduction/introduction-to-the-pom.html#Project_Inheritance_vs_Project_Aggregation`

Exemple d'héritage

pom.xml

```
<project>
  <parent>
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <version>1</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>my-module</artifactId>
</project>
```

pom.xml du parent

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
</project>
```

Hiérarchie de projets: Agrégation

- Utilisation: Regrouper un ensemble de projets à construire ensemble
- Définition d'un POM connaissant un ensemble des modules
- Quand une commande est exécutée sur le POM parent, elle est aussi exécutée sur les modules du parent
  - Le POM parent doit avoir le packaging `pom`
  - Peut être combiné avec de l'héritage
- Arborescence de fichiers
  - `pom.xml` (parent)
  - `my-module/pom.xml`

Exemple d'agrégation

pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-module</artifactId>
  <version>1</version>
</project>
```

pom.xml du parent

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
  <packaging>pom</packaging>

  <modules>
    <module>my-module</module>
  </modules>
</project>
```

Quelques détails en plus

- Afficher un fichier POM complet (utile pour le debug)
  - `mvn help:effective-pom`
- Afficher la liste des tâches d'un plugin
  - `mvn help:describe -Dplugin=pluginName`
- Afficher la liste des paramètres d'une tâche
  - `mvn help:describe -Dcmd=pluginName:goal -Ddetail`

Résumé

Maven, c'est:

- Convention plutôt que configuration
- Décrire plutôt que programmer

Plus précisement:

- Un identifiant unique de projet (sous forme de Coordonnées)
- Un cycle de vie par défaut (qui définit des phase)
- Un packaging de projet qui définit les tâches par défaut associées à chaque phase
- Des plugins qui peuvent exécuter une ou plusieurs tâches
  - Ces tâches peuvent être associées à des phases du cycle de vie
- Des dépendances à des artefacts
  - Qui sont gérées automatiquement (au travers de dépôts publiques ou privés) – tout comme les plugins
  - Qui peuvent être associées à certaines phases

Une alternative à Maven: Gradle

Existant:

- Ant: Flexible mais nécessite une description complète de projet
- Maven: "Convention plutôt que configuration" mais manque de flexibilité

Gradle

- Vise à combiner le meilleur des 2 mondes
- Principe de bases similaires à Maven
  - Cycle de vie
  - Plugins
  - Dépendences
- Utilise une DSL (Groovy) à la place du xml
  - Permet d'include du code pour mettre en œuvre de nouvelles fonctionalités.

Références

- Notes de D. Donsez
- Apache Maven par N. De Loof et A. Héritier
- Traduction française de "Maven: The Definitive Guide"[1]
- `http://maven.apache.org/guides/index.html`
- *A practical guide to continuous delivery* par E. Wolff

[1] `http://maven-guide-fr.erwan-alliaume.com/maven-guide-fr/site/reference/public-book.html`