Qualité de développement - MVN

IUT lyon 1

Théo Rabut

Maven Késako ?

Automatisation du traitement des phases du cycle de vie

- Peut être vu comme un "super Makefile"
  - Java comme langage de script
- Lance l'exécution d'outils:
  - Compilation
  - Test automatisés
  - Archives, Déploiement
  - Génération de documentation
  - ...

Alternatives: CMake, Premake, Grunt, Gulp, etc

Architecture

- Basée sur un système de plugins
  - Un plugin $\leftrightarrow$ un script Java
    - i.e. une classe avec une méthode particulière
    - paramétrable via un morceau de XML
  - Une exécution de maven $\leftrightarrow$ suite d'exécution de plugins

- Nombreux plugins disponibles
  - Pas tous installés au départ
  - Système de téléchargement de plugins à la demande

Phases et cycles de vie

- Une phase regroupe un ensemble de tâches (goals)
  - 1 tâche $\rightarrow$ 1 plugin
- Un cycle de vie est une suite de phases
  - Le déclenchement d'une phase déclenche les phases précédentes du cycle de vie
- Le cycle de vie dépend du *packaging* (jar,war, ...)
  - *packaging* = type de projet
  - Format d'archive
  - Ordre des phases
  - Affectation tâches $\rightarrow$ phases
  - Préconfiguration des tâches
  - peut être reconfiguré selon les besoins du projet

Exemple: phases du *packaging* jar

Phase	Tâche(s)
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`

Projet maven: organisation des fichiers

- pom.xml ← config. du projet
- src/ ← sources
  - main/ ← à distribuer
    - java/ ← code Java
    - resources/ ← fichiers à distribuer (config appli, images, etc)
    - webapp/ ← ressources web (pour les war: html, jsp, js, images)
    - antlr4/ ← grammaire pour générer les *parsers*
    - ...
  - test/ ← uniquement pour les tests
    - java/, resources/, antlr4/, etc
- target/ ← tout ce qui est généré, il est supprimé par `clean`
  il ne faut **pas** le versionner (utiliser `.gitignore`)

Projet Maven: le pom.xml

```
<project ...>
  ...
  <groupId>fr.univ-lyon1.info.m1</groupId>
  <artifactId>monProjet</artifactId>

  <dependencies>
    <dependency>...</dependency>
    <dependency>...</dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>...</plugin>
      <plugin>...</plugin>
    </plugins>
  </build>
</project>
```

Repository Maven

- Dépôt contenant:
  - Des plugins
  - Des bibliothèques (en général Java)
- Sur le web
  - Téléchargement automatique à la demande
  - Défaut: `http://repo1.maven.org`
  - Miroirs (Nexus, Archiva, etc)
- Local: `~/.m2/repository`
  - contient les archives des projets locaux
    - phase `install`
  - cache pour les *repository* web

Projet Maven: dépendances dans le pom.xml

```
<project ...>
  <dependencies>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-controls</artifactId>
      <version>11.0.2</version>
    </dependency>
  </dependencies>
  ...
</project>
```

En pratique, morceaux de XML à copier-coller (+ adapter ?) depuis https://search.maven.org/, Maven s'occupe du reste !

Classpath et dépendances

Utilisation de libs externes, Maven s'occupe de :

- Téléchargement
- Gestion des versions ($\Rightarrow$ build reproductible)
- Transitivité des dépendances
- Configuration du CLASSPATH

Également utilisé pour les plugins

Dépendances: *scope*

Dépendances dans le CLASSPATH dans chaque phase selon la porté (scope)

scope	compilation	exécution	test	déploiement
compile	x	x	x	x
provided	x	x	x	
runtime		x	x	x
test			x	

(+ system, import)
Exemple avec JUnit:

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.7.0-M1</version>
  <scope>test</scope> <!-- not available in 'mvn exec:java' -->
</dependency>
```

Archetypes

- Complexité inhérente aux projets maven
  - Difficultés de mise en œuvre
- Archetype = mini-projet de départ
  - D'un type particulier
  - Préconfiguré
- Exemples
  - maven-archetype-quickstart
  - spring-mvc-jpa-archetype
- En ligne de commande: `mvn archetype:generate`

Intégration dans les IDE

Un bon IDE a besoin de connaître les dépendances (erreurs de typage, suggestions, ...). `pom.xml` contient toutes les informations.

- IntelliJ : par défaut
- VSCode :
  - Plugin Maven for Java, inclus dans Java Extension Pack
  - File → Open Folder sur le répertoire contenant le `pom.xml` configure le projet automatiquement.
- Eclipse :
  - Plugin m2e + connecteurs
  - `mvn eclipse:eclipse`
    - génère une configuration de projet Eclipse
    - qui correspond au projet maven
- Netbeans: par défaut