Maven

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 16 avril 2019

Problème des dépendances transitives

- Souhait d'utiliser une bibliothèque tierce
- Cette bibliothèque requiert d'autres bibliothèques

⇒ Nécessité de télécharger et configurer IDE pour chaque dépendance

- Si d'autres utilisent mon code : amplification du problème pour eux
- Bonne pratique : découpe du code en bibliothèques séparées (outils, etc.) pour gestion d'entités plus petites
- Amplifie encore le problème !

Exemple (JasperReports)

- 18 dépendances (Central, détails)
- Plus facultatif (exemple) Apache Batik (génération de SVG), qui requiert batik-awt-util, batik-util, xml-apis

Problème de l'environnement de compilation

- Partage du code source avec autres développeurs
- Utilisent un autre IDE
- Doivent à leur tour configurer les dépendances
- Et autres paramètres du projet
- Exemple : répertoires code source, répertoires ressources
- Quand changement de configuration : à répercuter sur tous les environnements !

Intégration continue

- Envoi du code sur un serveur
- Le serveur compile et exécute les tests
- Permet d'assurer la reproductibilité de la construction
- Requiert manière automatique de préciser l'environnement (dépendances, paramètres)

Présentation de Maven

- Apache Maven
- Outil de gestion de configuration de projet
- En particulier : gestion de dépendances
- Description de votre projet via POM (Project Object Model)
- *Convention over configuration* : peu de configuration grâce aux valeurs par défaut
- Dépôt central avec publications open source
- Fortement basé sur plugins

Le POM

- Fichier XML
- Décrit un projet ou module et comment le construire (*build*)
- Un projet *peut* être composé de modules

Exemple de POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="..." xsi:schemaLocation="...">
  <modelVersion>4.0.0</modelVersion>
  <groupId>myGroupId</groupId>
  <artifactId>myArtifactId</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

Structure du projet

Structure déterminée normalement par *conventions*

Arborescence de base

```
pom.xml
/src
    /main
        /java
        /resources
    /test
        /java
        /resources
```

Répertoires de base

`/src/main/...`	fichiers code et ressources « normales »
`.../java`	code java
`.../resources`	images, etc., devant être dans classpath
`/src/test/...`	fichiers code et ressources pour tests

Exécution

- Lors exécution de Maven, préciser un plugin et un goal
- (ou préciser une phase, voire plus loin)
- Maven accompagné de nombreux plugins
- Plugins téléchargés à la demande et mis en cache
- Exemple : `mvn compiler:compile`
- Appelle goal `compile` dans plugin `compiler`
- Compile sources dans répertoire destination
- `compiler` est le nom *court* du plugin
- Coordonnées complètes : groupId, artifactId, version
- Version par défaut dépend de la version de Maven installée
- Commande complète : `mvn org.apache.maven.plugins:maven-compiler-plugin:3.8.0:compile` (en un seul mot)

Cycles de vie Maven

- On veut généralement exécuter un ensemble ordonné de goals
- Maven définit des cycles de vie
- Cycles embarqués : default ; clean et site mais plus utilisé
- Cycle : ensemble ordonné de *phases*
- Lors exécution de Maven, préciser une phase (ou un plugin et un goal)
- Maven en déduit le cycle
- Cycle “clean” contient essentiellement phase “clean”

Cycle “default”

Phases (non exhaustif) dans cycle « default » :

validate	valide informations du projet
process-resources	copie vers destination
compile	compilation du code source
test	lancement des tests
package	création d’un paquet
integration-test	tests d’intégration
verify	vérification de la validité du paquet
install	installation en local
deploy	déploiement dans dépôt configuré

Phases

- Chaque phase associée à un ensemble de plugins et d'objectifs (*goals*)
- Phase process-resources associée par défaut à plugin **Resources**, objectif `resource`
- Phase test associée par défaut à plugin **Surefire**, objectif `test`
- Phase package associée par exemple à plugin **JAR**, objectif `JAR`

Exécution

Lancement de Maven avec `mvn phasechoisie` :

- Maven détecte de quel cycle il s'agit
- Maven exécute toutes les phases jusqu'à « phasechoisie »
- Exemple : exécution systématique de `test` avant `package`

Dépendances

- Maven permet de gérer les « dépendances »
- Pour compiler (dépendance statique) ; s'exécuter ; pour tests uniquement...
- Maven gère les dépendances transitives pour vous !
- Dépendances prises par défaut dans Maven Central Repository
- Dans POM : section `<dependencies>`
- Dans cette section : ajouter une section `<dependency>` pour chaque dépendance à gérer

Dépendances : exemples

Exemple : dépendance vers Google Guava

```
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>27.1-jre</version>
</dependency>
```

Exemple : dépendance vers junit

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

Dépendances dans POM

- Trouver `groupId` et `artifactId` : voir site du projet
- Trouver `version` : voir Central
- Presque tous les projets Java récents font une release Maven

Portées (liste non exhaustive) :

`compile` Par défaut

`test` Bibliothèque incluse uniquement lors phase tests

`runtime` Bibliothèque incluse uniquement lors exécution, pas lors compilation

Configuration des plugins

- Voir Liste pour plugins de Apache
- Configuration parfois utile
- Dans POM : ajouter section
  `<build><plugins><plugin>...</plugin></plugins></build>`
- Exemple : pour configurer la compilation, voir la page “Apache Maven Compiler Plugin”

Propriétés

- Propriété `ma propriété` : accessible via `${ma propriété}`
- Nommage souvent hiérarchique : `catégorie.sous-catégorie.nom-propriété`

Dans POM :

```
<properties>
  <cat.etc.prop1>valeur1</cat.etc.prop1>
  <cat.etc.prop2>valeur2</cat.etc.prop2>
</properties>
```

Puis possible d'utiliser `${cat.etc.prop1}`.

Conventions et configurations classiques

- Utiliser comme groupeId un nom unique : généralement un nom de domaine inversé
- Le paquet de base de toutes les classes doit être ce nom
- Indiquer propriété `project.build.sourceEncoding` avec valeur `UTF-8`
- Indiquer propriétés `maven.compiler.source` et `maven.compiler.target` avec valeur `8`

Maven et Eclipse

- M2Eclipse (m2e) fournit support Maven pour Eclipse
- Maven embarqué
- Wizards pour démarrer ou importer un projet maven

Licence