CSC4251_4252 : Cours pratique — Maven et autres

Denis Conan et J. Paul Gibson

Novembre 2024

Sommaire

1. Construction de logiciel avec `Maven`
2. Écriture des tests avec JUnit

1 Construction de logiciel avec Maven

1.1 Motivations — Pourquoi Maven ?
1.2 Dépôt et arborescence standard
1.3 Processus de construction par défaut
1.4 Processus de construction avec JFlex et CUP

. https://maven.apache.org
. http://books.sonatype.com/mvnex-book/reference/index.html

1.1 Motivations — Pourquoi `Maven` ?

- Construction de logiciel JAVA indépendante de l'*IDE*
  - Description de la construction dans un schéma XML
    - Certains trouvent XML trop verbeux et préférerait par exemple Gradle
      - Gradle est basé sur Maven
    - Vous n'aurez pas à écrire de fichier XML dans ce module car ils sont fournis
- Mode commande possible pour permettre d'éviter les avalanches de clics
- Séparation entre le code pour le compilateur et le code pour les tests du compilateur
- Maven est un standard dans le monde JAVA

- Processus standard de construction : `javac`, `jar`, `java`, etc.
  AVEC, pour ce module, besoin d'ajouter l'exécution de JFlex et de CUP

. Pour les curieux, amélioration par rapport à `make` et `Ant`

1.2 Dépôt et arborescence standard

- Dépôts (*Repositories*)
  - Beaucoup de dépôts sont disponibles
    - P.ex. `https://mvnrepository.com/` ou `https://search.maven.org/`
  - Dépôt local dans le répertoire : `~/.m2/repository`
    - Peuplé comme un *cache* des dépôts distants
- Arborescence standard — version simplifiée
  - `pom.xml` : POM (*Project Object Model*), toujours au niveau le plus élevé
    - Déclaration de configuration du processus de construction du module
  - `src/main/java` : code source du module
  - `src/main/resources` : ressources du module $\Longleftarrow$ les fichiers JFlex et CUP
  - `src/test/java` : code source des tests du module
  - `src/test/resources` : ressources des tests $\Longleftarrow$ les fichiers des tests
  - `target` : tous les contenus générés `*.java`, (`*.class`, `*.jar`, site Web Javadoc...)

1.3 Processus de construction par défaut I

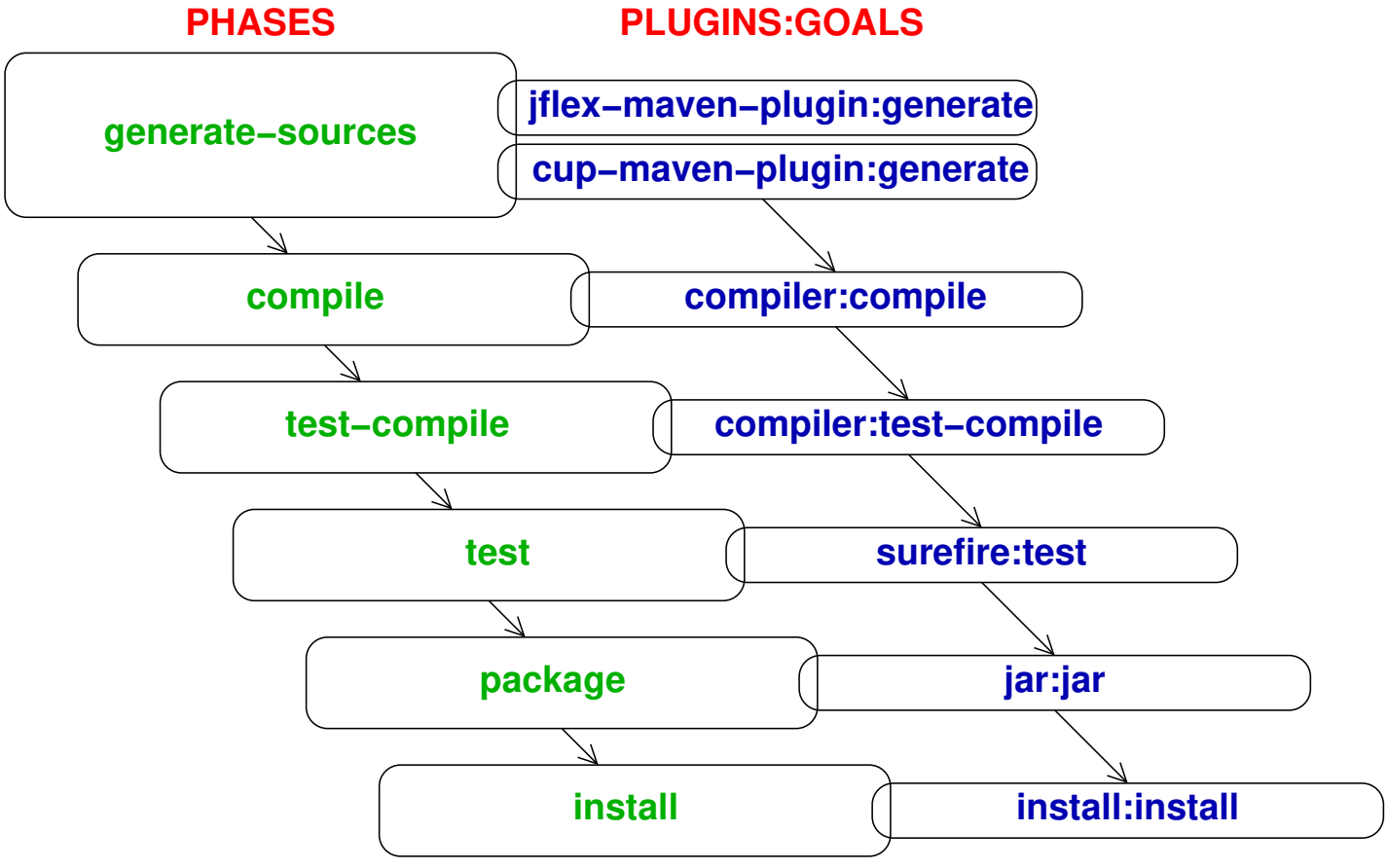
Extrait de la documentation Maven :
https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

- `validate`: validate the project is correct and all necessary information is available.
- `initialize`: initialize build state, e.g. set properties or create directories.
- `generate-sources`: generate any source code for inclusion in compilation.
- `process-sources`: process the source code, for example to filter any values.
- `generate-resources`: generate resources for inclusion in the package.
- `process-resources`: copy and process the resources into the destination directory, ready for packaging.
- `compile`: compile the source code of the project.
- `process-classes`: post-process the generated files from compilation, for example to do bytecode enhancement on Java classes.
- `generate-test-sources`: generate any test source code for inclusion in compilation.
- `process-test-sources`: process the test source code, for example to filter any values.
- `generate-test-resources`: create resources for testing.
- `process-test-resources`: copy and process the resources into the test destination directory.
- `test-compile`: compile the test source code into the test destination directory.
- `process-test-classes`: post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes.

1.3 Processus de construction par défaut II

- `test:` run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
- `prepare-package:` perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package.
- `package:` take the compiled code and package it in its distributable format, such as a JAR.
- `pre-integration-test:` perform actions required before integration tests are executed. This may involve things such as setting up the required environment.
- `integration-test:` process and deploy the package if necessary into an environment where integration tests can be run.
- `post-integration-test:` perform actions required after integration tests have been executed. This may including cleaning up the environment.
- `verify:` run any checks to verify the package is valid and meets quality criteria.
- `install:` install the package into the local repository, for use as a dependency in other projects locally.
- `deploy:` done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

1.4 Processus de construction avec JFlex et CUP



2 Écriture des tests avec JUnit

- Écriture de tests qui peuvent être exécutés et ré-exécutés de manière automatique lors de la construction du logiciel
  - Important car de nombreux tests
- Données en entrée des tests :
  - Des expressions (chaînes de caractères) en entrée
    ou des fichiers contenant les expressions à analyser
- Nous proposons des tests
  - Et sauf erreur de notre part, votre solution doit « passer » ces tests
- Classes de tests JAVA écrites en JUnit 5 dans l'arborescence `src/test/java`
- Classes de tests exécutées avec le greffon par défaut `maven-surefire-plugin`