Licence Professionnelle SIL

TD Maven

Intervenants

Benoît Cotinat (benoit.cotinat@netapsys.fr)

Vivien Ouairy (vivien.ouairy@gmail.com)

Table des matières

Description du sujet

Scénario

Nous souhaitons développer un site d'achat en ligne. Pour cela, nous avons récupéré du code d'une ancienne application et nous souhaitons le récupérer comme base pour nos développements. Nous allons construire l'application en utilisant Maven en introduisant les concepts petit à petit.

Prérequis

Il faut avoir installer le JDK 6 et Maven 3.

Installation de Maven

Pour vérifier l'installation, utiliser la commande : `mvn --version` ou `mvn -v`. On doit obtenir :

```
Apache Maven 3.0.4 (r1232337; 2012-01-17 09:44:56+0100)
Maven home: /opt/commun/apache-maven
Java version: 1.7.0_04, vendor: Oracle Corporation
Java home: /usr/java/jdk1.7.0_04/jre
Default locale: fr_FR, platform encoding: UTF-8
OS name: "linux", version: "3.2.0-26-generic", arch: "amd64", family: "unix"
```

Récupération du support pour le TD

Récupérer les supports à l'adresse : https ://bit.ly/16SA21v.

Parmi les fichiers récupérés, on trouve un fichier `settings.xml`, qui indique à Maven la configuration du proxy de l'IUT pour lui permettre de se connecter à l'Internet. Il faut placer ce fichier dans le répertoire `.m2` situé dans votre `home directory` (si besoin, afficher les dossiers cachés en faisant `Ctrl + H` sous Linux).

Un ensemble de dossiers `tdMavenEtapeXX` sont fournis. Ils contiennent les supports nécessaires au déroulement du TD. Ces dossiers doivent être conservés sans modifications. Le travail à réaliser sera fait dans d'autres répertoires dans lesquels on copiera les fichiers fournis. Les fichiers récupérés (source, fichiers de paramétrage, ...) seront utilisés tel quel. Il ne sera pas nécessaire de les modifier sauf si cela est précisé. Les librairies (`*.jar`) fournies servent à indiquer les dépendances et les versions nécessaires pour le programme. Elles serviront pour l'exécution du programme mais pas pour sa construction. Un simple éditeur de texte sera suffisant pour dérouler le TD.

Ressources

Site officiel : http ://maven.apache.org/
Pour commencer : http ://maven.apache.org/guides/getting-started/index.html
Organisation des répertoires : http ://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html
Les phases : http ://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html
Structure du pom.xml : http ://maven.apache.org/ref/3.0.3/maven-model/maven.html
Recherche des dépendances et plugins : http ://mvnrepository.com/ et http ://search.maven.org/

Etape 1

Objectifs

- Créer un pom.xml ;
- Compiler un programme avec Maven.

Description

Nous cherchons à construire un jar avec Maven à partir du code contenu dans le répertoire src fourni.
Le package java de l'application est : `edu.univ.nantes.tdMaven`.
Créer un répertoire `tdmaven-calculpanier` pour le projet. Ecrire un `pom.xml` minimal. Il ne doit contenir ni plugins, ni dépendances.
Le `groupId` sera `edu.univ.nantes`, l'`artifactId` sera `tdmaven-calculpanier`, et la version `1.0-SNAPSHOT`.
Placer les fichiers fournis dans votre projet en suivant l'organisation des répertoires standards définis par Maven.
Tester les commandes suivantes et regarder l'impact sur les répertoires `target` et `.m2/repository` (répertoire `edu`). Identifier quelle commande permet de construire les `.class`, le `.jar`, et quelle commande permet de placer le `.jar` dans le repository.

- `mvn clean`
- `mvn compile`
- `mvn package`
- `mvn install`
- `mvn clean compile`

Regarder les lignes ayant pour format :

- Sous Linux : `[INFO] [resources:resources]`
- Sous Windows : `[INFO] --- maven-resources-plugin:2.4.3:resources (default-resources) @ tdmaven-calculpanier ---`

Validation

Exécuter le programme depuis le répertoire `target/classes` (appel direct des classes) :

```
java edu/univ/nantes/tdMaven/CalculPanier 2 2
```

Exécuter le programme en utilisant le jar généré depuis le répertoire `target` :

```
java -cp tdmaven-calculpanier-1.0-SNAPSHOT.jar edu/univ/nantes/tdMaven/CalculPanier
2 2
```

Etape 2

Objectifs

- Gérer une dépendance ;
- Gérer un fichier de ressource.

Description

Dans la nouvelle version de la classe de calcul, nous avons remplacer le système de traces pour utiliser la librairie log4j au lieu d'une sortie avec `System.out`. Il faut prendre en compte ces modifications et gérer la dépendance vers la librairie.
Aller chercher sur le repository Maven sur le web (http ://mvnrepository.com/), le texte à ajouter dans le `pom.xml` pour ajouter la dépendance à utiliser. Le répertoire lib contient la librairie permettant de trouver son nom et sa version.
Positionner correctement le fichier de paramétrage `log4j.xml`.
Pour l'exécution du programme, il faut copier le répertoire lib et son contenu à la racine du projet.

Validation

Sous Linux :

```
java -cp target/tdmaven-calculpanier-1.0-SNAPSHOT.jar:lib/log4j-1.2.14.jar
edu/univ/nantes/tdMaven/CalculPanier 2 2
```

Sous Windows :

```
java -cp target/tdmaven-calculpanier-1.0-SNAPSHOT.jar;lib/log4j-1.2.14.jar
edu/univ/nantes/tdMaven/CalculPanier 2 2
```

Résultat attendu :

```
INFO - Calcul de prix
INFO - Prix:2
INFO - Quantité
DEBUG - 2*2=4
INFO - Prix total:4
```

Etape 3

Objectifs

– Exécuter les tests.

Description

Nous avons récupéré les tests de l'application et nous voulons les exécuter pour valider l'application. Placer le code des tests dans le répertoire standard pour les tests défini par Maven.
Ajouter la dépendance nécessaire dans le `pom.xml` (récupérer la dépendance en se basant sur la librairie présente dans le répertoire libTest). Les librairies du répertoire libTest n'étant pas nécessaire lors de l'exécution du programme, elles ne doivent pas être copiées dans le projet.
Générer le `.jar` de la librairie. Si les tests échouent, le détail de l'erreur se trouve dans le répertoire de rapport des tests surefire-reports. Corriger le programme présentant une anomalie (pas les tests).

Validation

Exécuter la commande :

```
mvn verify
```

On doit voir les tests exécutés :

```
-------------------------------------------------------
T E S T S
-------------------------------------------------------
Running edu.univ.nantes.tdMaven.CalculPanierTest
DEBUG - 3.0*5=15.0
DEBUG - 3.0*5=15.0
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.208 sec

Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
```

Etape 4

Objectifs

- Mise à jour d'une librairie ;
- Notion de dépendance transitive ;
- Plugin pour définir le compilateur.

Description

Nous souhaitons utiliser une version plus récente de JUnit (4.10). Mettre à jour la version de JUnit dans les dépendances du projet.
Compiler l'application et regarder les messages. Mettre à jour la version du compilateur utilisé (voir http ://maven.apache.org/general.html).

Validation

Tester la commande suivante pour voir l'arbre des dépendances utilisées dans le projet, et identifier les raisons de leur présence :
`mvn dependency:tree`

Nous n'avons pas besoin de la dépendance tirée par junit, l'exclure du pom et vérifier ce que cela change sur la commande précédente.

Etape 5.1

Objectifs

- Création d'un application web ;
- Utilisation du plugin jetty ;
- Intégrer une libraire externe.

Description

Nous souhaitons maintenant créer une application web pour utiliser notre logiciel sur Internet. Se placer dans le répertoire parent de notre projet précédant (`tdmaven-calculpanier`).
Pour générer la structure du projet nous allons utiliser la commande :
`mvn archetype:generate -Dfilter=org.apache.maven:`
Le filtre permet de limiter le nombre de choix proposés à ceux commençant par `org.apache.maven`.
Choisir l'archetype suivant :
`org.apache.maven.archetypes:maven-archetype-webapp`
Pour les questions suivantes, choisir la valeur par défaut (entrée) sauf pour :
`groupId : edu.univ.nantes`
`articfactId : tdmaven-calculweb`

Regarder les différences entre le fichier `pom.xml` de notre librairie et celui du projet web généré, en particulier la balise `<packaging>`.

Recopier les fichiers fournis au bon endroit dans la nouvelle arborescence. Corriger la version de junit.
Ajouter la dépendance vers `log4j` (rependre celle du module `tdmaven-calculpanier`).
Ajouter la dépendance vers `servlet-api`, qui peut être trouvé en cherchant `javaservlet` sur http ://mvnrepository.com/.

Packager l'application et regarder le fichier produit. Ce fichier ne doit contenir que la librairie log4j. Pour cela, utiliser la notion de scope sur les dépendances :

compile (par défaut) nécessaire pour la compilation et l'exécution du programme

test nécessaire pour la compilation et l'excution des tests (mais pas du programme)

provided nécessaire pour la compilation mais fourni par un tiers à l'exécution

Validation

Jetty est un serveur web simple. Nous allons pouvoir y faire tourner notre application.
Pour pouvoir l'utiliser, nous allons ajouter au `pom.xml` le plugin `jetty-maven-plugin` ayant pour `groupId` : `org.mortbay.jetty` (ne pas préciser la version).
Démarrer le serveur web avec la commande :
`mvn jetty:run`

Dans le fichier `web.xml`, corriger le chemin vers la servlet fournie. Cette classe sera appelée par jetty lorsqu'on accédera à l'application.
L'application est accessible à l'adresse suivante :
http ://localhost :8080/tdmaven-calculweb

Etape 5.2

Objectifs

- Réutiliser notre librairie dans le projet ;
- Réutiliser une libraire non géré par Maven.

Description

Nous souhaitons d'une part utiliser la libraire que nous avons créée précédemment et d'autre part utiliser une librairie qui n'est pas gérée par Maven (pas de fichier `pom`, non disponible sur le net).
Supprimer le fichier `CalculPanier.java` du projet et exécuter la commande :
`mvn clean compile`

Une erreur doit indiquer l'absence de la classe.
Ajouter la dépendance vers la librairie `tdmaven-calculpanier` pour faire fonctionner le projet (l'installer dans le repository local à l'aide de Maven si besoin).

Sur la page de résultat, il est indiqué que la libraire `DateUtils` n'est pas disponible dans l'application. Cette librairie est fournie dans le répertoire `lib`. Il s'agit d'une librairie propriétaire construite sans Maven (pas de fichier `pom`) et qui n'est pas disponible sur Internet. Il faut l'ajouter manuellement à notre repository.
Pour cela, utiliser la commande `install-file`.

Validation

Après l'ajout de la dépendance dans le `pom.xml`, la date doit s'afficher sur la page.

Etape 6

Objectifs

- Gérer un projet parent ;
- Gérer les versions à travers le dependencyManagement ;
- Gérer les plugins à travers le pluginManagement ;
- Construction complète du projet avec les modules.

Description

Nous avons maintenant 2 modules dans notre projet : la librairie et l'application web. Pour garantir une homogénéité, nous allons créer un projet parent qui sera charger de gérer les parties communes :

1. Créer un `pom.xml` parent (répertoire parent de nos deux projets) ;
2. Faire hériter nos deux projets de ce parent commun ;
3. Faire remonter les dépendances vers le parent ;
4. Vérifier la compilation des modules ;
5. Supprimer les balises versions des modules, ils hériteront du numéro du pom parent.

Notre module `tdmaven-calculpanier` est maintenant dépendant à tort des librairies `javax.servlet-api` et `dateUtils`. Utiliser la section `dependencyManagement` du projet parent pour gérer ces librairies et leurs versions.
Dans le module `tdmaven-calculweb`, supprimer le numéro de version "en dur" pour le remplacer par la variable Maven adéquate.

Remonter les plugins communs dans le pom parent (`maven-compiler-plugin`). Remonter les plugins optionnel dans la section `pluginManagement`.

La construction du projet nécessite la compilation des différents projets dans l'ordre des dépendances, ce qui peut être compliqué sur de gros projets. Pour simplifier cette construction, on va référencer les différents module dans le pom parent. La compilation du pom parent va gérer automatiquement l'ordre de construction.

Validation

Après avoir ajouté les modules dans le pom parent, faire un `mvn clean` sur le pom parent. Vérifier que tous les répertoires `target` ont été supprimés.
Exécuter `mvn install` pour construire tous les modules.
Vérifier la bonne exécution du module web.

Etape 7

Objectifs

- Création de la documentation ;
- Modifier la phase d'exécution.

Description

Nous avons besoin de fournir une documentation pour le projet. Utiliser le plugin `maven-javadoc-plugin`.
Executer la commande :
`mvn javadoc:javadoc`
Vérifier la génération dans le répertoire `target/site/apidocs`.

Nous voulons maintenant que notre documentation soit générée dès la phase de `site` (phase spécifique pour la documentation). Vérifier qu'après la commande suivante, la documentation n'est plus présente :
`mvn clean site`

Validation

Ajouter à la définition du plugin, une section `execution` qui va déclencher le goal `javadoc` pendant la phase `site`. Relancer la commande précédente et vérifier la génération de la documentation.

Etape 8

Objectifs

– Ajouter un profil pour générer la documentation.

Description

Nous souhaitons générer la documentation pendant la phase `packaging` lorsque nous construisons l'application pour la production.

Ajouter un profil `dev` et un profil `prod`.
Rendre le profil `dev` actif par défaut si aucun paramètre n'est spécifié.
Exécuter la génération de la documentation en phase `package` pour la `prod` uniquement.

Validation

Vérifier que la documentation n'est générée que pour la `prod` :
`mvn clean package -Pdev`
Pas de répertoire `target/site/apidocs`.

`mvn clean package -Pprod`
Documentation créée dans le répertoire `target/site/apidocs`.

Etape 9

Objectifs

– Compléter la documentation.

Description

Nous souhaitons compléter la documentation générée avec d'autres indicateurs. Checkstyle est un outil permettant de vérifier le respect des conventions de code. Compléter le pom avec le plugin checkstyle pour que ces informations soient générées lors de la génération du site.

Validation

Relancer la génération de la documentation, et vérifier l'ajout de l'entrée `Checkstyle` dans la section `Project Reports`.

Etape 10

Objectifs

– Ajouter un filtre sur les ressources.

Description

Nous voulons configurer différemment les logs en fonction de l'environnement :

debug dans l'environnement de développement ;

error dans l'environnement de production ;

Ajouter les fichiers de filtre dans le répertoire standard Maven dédié à cela.
Modifier le fichier `log4j.xml` et remplacer le niveau de trace (`debug`) par le paramètre défini dans le filtre (syntaxe : `${NOM_PARAMETRE}`) :
`<level value="debug"/>`

Compléter le pom principal pour indiquer l'emplacement des fichiers de filtres, et les ressources à filtrer.

Validation

Packager et tester l'application avec les deux profils, regarder les différences dans la console.

Etape 11

Objectifs

– Construire un jar executable en standalone.

Description

Nous avons vu dans les étapes 1 et 2 que pour exécuter le jar, il fallait indiquer le chemin vers les jar dépendants, ainsi que vers le nom complet de la classe principale.
Le but ici est de configurer notre module `tdmaven-calculpanier` pour qu'il se lance grâce à la commande :
`java -jar tdmaven-calculpanier-1.0-SNAPSHOT.jar 2 2`

Il est impossible de construire un jar contenant d'autres jar. Nous allons donc utiliser le plugin `maven-assembly-plugin` qui va nous permettre d'agréger notre jar et ses dépendances dans une nouvelle archive, ainsi que d'indiquer la classe à appeler.
Ajouter et configurer ce nouveau plugin pour qu'il s'exécute pendant la phase `package`.

Validation

Regarder la différence entre les 2 jar générés, ainsi que le contenu du fichier `META-INF/MANIFEST.MF`, et exécuter le programme grâce à la commande souhaitée.

Etape 12

Objectifs

– Paramétrer le gestionnaire de source au sein de Maven.

Description

Notre projet est maintenant terminé, nous voulons faire une release de la version de production, et la sauvegarder sur notre référentiel de source. Pour ne pas avoir à faire toutes ces étapes à la main, ajouter au `pom` les informations nécessaires pour vous connecter à votre référentiel, ainsi que le plugin `maven-release-plugin`.

Validation

Après avoir préparer la release, regarder les impacts sur les différents `pom`, le contenu du dossier `target` et de votre repository local :
`mvn release:prepare`

Lancez l'exécution de la release, trouver le paramètre ajouter pour qu'elle se termine correctement :
`mvn release:perform`

Annexe

Installation de Maven

Extraire Maven dans un répertoire.
Ajouter Maven au classpath :

- sous linux :

```
export M2_HOME=[Répertoire d'installation]
export M2=$M2_HOME/bin
export PATH=$M2:$PATH
```

- sous windows :
  1. Panneau de configuration → Système et sécurité → Système → Paramètres système avancés → Variables d'environnement
  2. Créez une variable `M2_HOME` (lien absolu vers le répertoire maven)
  3. Créez une variable `M2=%M2_HOME%\bin`
  4. Ajoutez `M2` à la variable d'environnement `PATH`

Glossaire

Maven Outil de construction de projets Java.

Dépendance Librairie nécessaire pour le programme au moment de sa compilation, de son exécution ou pour les tests. Elles sont indispensables au programme qu'il soit construit avec ou sans Maven.

Plugin Programme permettant d'étendre les fonctionnalités de Maven. Ils ne servent pas au programme directement mais aide à le construire.

Repository Dossier contenant l'ensemble des ressources (dépendances et plugins) nécessaires pour construire les projets. Maven cherche d'abord les ressources dans ce dossier et, s'il ne la trouve pas, il va la chercher sur Internet.

GroupId Identifiant unique pour une organisation ou un projet (ex : `org.springframework`).

ArtifactId En général le nom du projet. Il permet, avec le `groupId`, d'identifier de façon unique un projet.

Archetype Projet Maven prédéfini.

PluginManagement / DependencyManagement Permet de définir la configuration par défaut lorsque le plugin ou la dépendance est utilisé.