3.3.7 Reverse-engineering

Introduction

Cet exposé a pour but de revenir sur l'utilisation d'un certain nombre de plugins maven, cela afin de pouvoir générer des classes Java à partir d'une structure de base de données.

L'outil utilisé est en réalité double :

- le plugin exec-maven-plugin
- la librairie openjpa

Utilisation du plugin exec-maven-plugin

Ce plugin est le chef d'orchestre de la manipulation. En effet, c'est ce plugin qui sert à lancer les classes effectuant le traitement, et ce, comme si elles étaient lancées en ligne de commande.

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>1.2</version>
    <executions>
        <execution>
            <goals>
                <goal>java</goal>
            </goals>
            <phase>generate-resources</phase>
        </execution>
    </executions>
    <configuration>
        <mainClass>org.apache.openjpa.jdbc.schema.SchemaTool</mainClass>
    </configuration>
    ...
</plugin>
```

Utilisation de la librairie openjpa

Production d'un fichier représentant la structure de la base de données

Il s'agit ici d'un travail de préparation. En effet, il faut produire un fichier représentant la structure de la base de données qui sera ensuite utilisé pour la génération des classes Java.

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>1.2</version>
    <executions>
        <execution>
            <goals>
                <goal>java</goal>
            </goals>
            <phase>generate-resources</phase>
        </execution>
    </executions>
    <configuration>
        <goal>generate-schema</goal>
        <mainClass>org.apache.openjpa.jdbc.schema.SchemaTool</mainClass>
        <commandlineArgs>
            -a reflect -p ${persistence.location} -f ${basedir}/src/main/resources/META-INF/schema_complet.xml
        </commandlineArgs>
        <systemProperties>
            <property>
                <key>persistence.location</key>
                <value>${persistence.location}</value>
            </property>
        </systemProperties>
        <includePluginDependencies>true</includePluginDependencies>
    </configuration>
    <dependencies>
        <dependency>
            <groupId>javax.validation</groupId>
            <artifactId>validation-api</artifactId>
            <version>1.0.0.GA</version>
        </dependency>
        <dependency>
            <groupId>org.apache.openjpa</groupId>
            <artifactId>openjpa-all</artifactId>
            <version>2.2.0</version>
        </dependency>
    </dependencies>
</plugin>
```

Grâce à ce code, le fichier *schema_complet.xml* va être généré dans *src/main/resources/META-INF/* du module courant.
La classe utilisée à tout de même besoin de connaître la base sur laquelle se connecter. Pour cela on lui passe, grâce à l'option -p, le chemin d'un fichier ayant la forme suivante :

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence
/persistence_2_0.xsd"
    version="2.0">
    <persistence-unit name="sample">
        <properties>
            <property name="openjpa.ConnectionUserName" value="username" />
            <property name="openjpa.ConnectionPassword" value="password" />
            <property name="openjpa.ConnectionDriverName" value="driver.class.name" />
            <property name="openjpa.ConnectionURL" value="jdbc:dbType:thin:server.univ.fr:port:DB" />
        </properties>
    </persistence-unit>
</persistence>
```

Si le fichier généré n'est pas complet, il est possible de le modifier. Plus de détails : http://openjpa.apache.org/builds/1.2.2/apache-openjpa-1.2.2/docs/manual/ref_guide_schema_schematool.html

NOTE : La génération préalable du schéma XML n'est obligatoire que du fait d'un bug (https://issues.apache.org/jira/browse/OPENJPA-2022?page=com.atlassian.jira.plugin.system.issuetabpanels:all-tabpanel) d'openjpa qui devrait être résolu en version 2.3.0. À compter de cette version, il devrait être possible de générer directement les entités à partie de la base de données.

Génération des classes Java

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>1.2</version>
    <executions>
        <execution>
            <goals>
                <goal>java</goal>
            </goals>
            <phase>generate-sources</phase>
        </execution>
    </executions>
    <configuration>
        <mainClass>org.apache.openjpa.jdbc.meta.ReverseMappingTool</mainClass>
        <commandlineArgs>
            -annotations true -useGenericCollections true -metadata none -nullableAsObject true -
innerIdentityClasses false -useBuiltinIdentityClass true -primaryKeyOnJoin true -accessType fields -d ${basedir}
/target/generated-sources/jpa -pkg x.y.z.beans -p ${persistence.location} ${basedir}/src/main/resources/META-INF
/schema.xml
        </commandlineArgs>
        <systemProperties>
            <property>
                <key>persistence.location</key>
                <value>${persistence.location}</value>
            </property>
        </systemProperties>
        <includePluginDependencies>true</includePluginDependencies>
    </configuration>
    <dependencies>
        <dependency>
            <groupId>javax.validation</groupId>
            <artifactId>validation-api</artifactId>
            <version>1.0.0.GA</version>
        </dependency>
        <dependency>
            <groupId>org.apache.openjpa</groupId>
            <artifactId>openjpa-all</artifactId>
            <version>2.2.0</version>
        </dependency>
    </dependencies>
</plugin>
```

Ce code va générer, sous *target/generated-sources/jpa/*, une classe par table présente dans le fichier *schema.xml*. Toutes ces classes vont avoir pour package *x.y.z.beans*.
Important : La ligne de commande passée dans les balises commandlineArgs doit être écrite sur une seule ligne.

Pour plus de détails sur les options : http://openjpa.apache.org/builds/1.2.2/apache-openjpa-1.2.2/docs/manual/ref_guide_pc_reverse.html

Pour que ces classes générées puissent être utilisées dans le reste de l'application, nous avons utilisé un autre plugin maven qui les ajoute en tant que sources :

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <executions>
        <execution>
        <id>add-source</id>
        <phase>generate-sources</phase>
        <goals>
            <goal>add-source</goal>
        </goals>
        <configuration>
            <sources>
                <source>${project.build.directory}/generated-sources/jpa/</source>
            </sources>
        </configuration>
        </execution>
    </executions>
</plugin>
```

Remarques

En cas d'utilisation du plugin m2e dans eclipse, il faudra sans doutes ajouter un autre plugin maven dans le pom.xml :

```
<pluginManagement>
    <plugins>
        <!--This plugin's configuration is used to store Eclipse m2e settings only. It has no influence on the
Maven build itself.-->
        <plugin>
            <groupId>org.eclipse.m2e</groupId>
            <artifactId>lifecycle-mapping</artifactId>
            <version>1.0.0</version>
            <configuration>
                <lifecycleMappingMetadata>
                    <pluginExecutions>
                        <pluginExecution>
                            <pluginExecutionFilter>
                                <groupId>
                                    org.codehaus.mojo
                                </groupId>
                                <artifactId>
                                    exec-maven-plugin
                                </artifactId>
                                <versionRange>
                                    [1.2,)
                                </versionRange>
                                <goals>
                                    <goal>java</goal>
                                </goals>
                            </pluginExecutionFilter>
                            <action>
                                <ignore />
                            </action>
                        </pluginExecution>
                    </pluginExecutions>
                </lifecycleMappingMetadata>
            </configuration>
        </plugin>
    </plugins>
</pluginManagement>
```

Il a pour but rendre l'utilisation du plugin maven-exec-plugin compatible avec le cycle de vie de m2e.