Introduction à JPA

Sébastien Nedjar et Fabien Pesci

1 Introduction

L'objectif de ce document est de vous présenter rapidement JPA 2.0 et de vous le faire tester sur un projet extrêmement simple. Pour rappel, Java Persistence API est une spécification standard de Java [1] permettant aux développeurs de gérer et manipuler des données relationnelles dans leurs applications. Cette API, bien qu'originaire du monde Java EE (Java Entreprise Edition), peut être utilisée aussi bien dans une application Java SE (Java Standard Edition) que dans un conteneur d'application Java EE.

JPA étant uniquement une spécification, il peut en exister plusieurs implémentations différentes. Si le code d'un projet se conforme parfaitement à la spécification, il est possible de passer d'une implémentation à une autre sans trop de difficulté. Comme Maven, JPA utilise la philosophie "Convention plutôt que configuration". L'idée est de faire décroître le nombre de décisions qu'un développeur doit prendre en lui proposant une convention adaptée au cas d'utilisation le plus classique qu'il pourra amender pour correspondre à ce qu'il veut faire. Ainsi la configuration n'est plus la norme mais l'exception.

Pour effectuer la configuration de la persistance, JPA utilise soit le mécanisme des annotations soit un fichier XML. Nous n'étudierons que les annotations car elles sont plus simples à mettre en œuvre et à maintenir en cohérence avec le code. Les annotations sont des méta-informations qui sont rajoutées aux classes métiers pour indiquer à JPA le travail qu'il doit faire pour les rendre persistantes. Le code des ces classes n'étant pas modifié, chacune d'elles reste un POJO ("Plain Old Java Object" que l'on pourrait traduire par "Bons Vieux Objets Java") que l'on pourra facilement tester (voir le tutoriel sur JUnit) comme n'importe quel POJO.

2 Mise en place de l'environnement de travail

Pour simplifier au maximum l'utilisation de JPA 2.0 dans ce tutoriel, Maven sera utilisé pour la construction et la gestion des dépendance. Il est donc requis de faire le tutoriel Maven avant d'attaquer celui-ci.

2.1 Installation

Voici la liste des outils qui seront utilisés pour la suite de ce tutoriel :
- Maven : Outil de gestion du cycle de vie d'un projet de développement logiciel.
- JPA 2.0 : Plus besoin de le présenter.
- EclipseLink : Framework open source et implémentation de référence de JPA 2.0.
- Derby : Apache Derby est un système de gestion de base de données relationnelle qui peut être embarqué dans un programme Java. Sa faible empreinte mémoire (moins de 2Mo) lui permet d'être utilisé dans un grand nombre de contexte (test unitaire dans ce tutoriel).
- MySQL : Un SGBD-R open source racheté récemment par Oracle.
- JUnit : Framework de test unitaire java.
- DbUnit : Extension de JUnit pour les applications très orientés BD.

Mis à par Maven et MySQL (déjà installé au département), tous les autres outils seront installés automatiquement grâce au système de gestion de dépendances de Maven.

1. JPA 2.0 est issue du travail de la JSR 317 : http://www.jcp.org/en/jsr/detail?id=317

2.2 Création du projet

Pour commencer, nous allons créer le projet de test que l'on nommera `tutoJPA` et qui sera dans le package `fr.iut.univaix.progbd`. Pour se faire on utilise la commande Maven suivante :

```
mvn archetype:generate -DinteractiveMode=false \
-DarchetypeArtifactId=maven-archetype-quickstart \
-DgroupId=fr.iut.univaix.progbd -DartifactId=tutoJPA
```

Une fois cette commande exécutée, il faut modifier le fichier `pom.xml` pour lui rajouter les dépendances nécessaires à un projet JPA 2.0 :

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                               http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>fr.univaix.iut.progbd</groupId>
    <artifactId>tutoJPA</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>tutoJPA</name>

    <url>http://maven.apache.org</url>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>6</maven.compiler.source>
        <maven.compiler.target>6</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.10</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>javax.persistence</artifactId>
            <version>2.0.0</version>
        </dependency>
    </dependencies>

    <repositories>
        <repository>
            <id>EclipseLink Repo</id>
            <name>EclipseLink Repository</name>
            <url>http://download.eclipse.org/rt/eclipselink/maven.repo</url>
        </repository>
    </repositories>
</project>
```

D'autres dépendances seront ajoutées au fur et à mesure de leur nécessité.

2.3 Création des classes métiers

L'exemple utilisé dans ce paragraphe est similaire à celui présenté dans le cours. Il modélise les employés d'une entreprise et les départements auxquels ils appartiennent. Il y aura donc 2 entités : `Employe` et `Departement`. L'adresse d'un employé sera modélisée par une classe intégrée. Dans le modèle de persistance JPA 2.0, un *entity bean* est une simple classe java (un Pojo) complétée par de simples annotations :

```java
package fr.univaix.iut.progbd.tutoJPA;
import javax.persistence.*;

@Entity //1
public class Employe {
    @Id //2
    @GeneratedValue//3
    private int id;
    @Column(length=50) //4
    private String nom;
    private long salaire;
    @Embedded //5
    private Adresse adresse;
    @ManyToOne //6
    private Departement departement;

    public Employe() {}
    public Employe(int id) { this.id = id; }
    public int getId() { return id; }
    // private void setId(int id) { this.id = id; }
    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }
    public long getSalaire() { return salaire; }
    public void setSalaire(long salaire) { this.salaire =salaire; }
    public Adresse getAdresse() { return adresse; }
    public void setAdresse(Adresse adresse) { adresse = adresse; }
    public Departement getDepartement() { return departement; }
    public void setDepartement(Departement departement) {
        this.departement = departement;
    }
}
```

Notez la présence d'annotations à plusieurs endroits dans la classe `Employe` :

1. Tout d'abord, l'annotation `@javax.persistence.Entity` permet à JPA de reconnaître cette classe comme une classe persistante (une entité) et non comme une simple classe Java.
2. L'annotation `@javax.persistence.Id`, quant à elle, définit l'identifiant unique de l'objet. Elle donne à l'entité une identité en mémoire en tant qu'objet, et en base de données via une clé primaire. Les autres attributs seront rendus persistants par JPA en appliquant la convention suivante : le nom de la colonne est identique à celui de l'attribut et le type String est converti en `VARCHAR(255)`.
3. L'annotation `@javax.persistence.GeneratedValue` indique à JPA qu'il doit gérer automatiquement la génération automatique de la clef primaire.
4. L'annotation `@javax.persistence.Column` permet de préciser des informations sur une colonne de la table : changer son nom (qui par défaut porte le même nom que l'attribut), préciser son type, sa taille et si la colonne autorise ou non la valeur null.
5. L'annotation `@javax.persistence.Embedded` précise que la donnée membre devra être intégrée dans l'entité.
6. L'annotation `@javax.persistence.ManyToOne` indique à JPA que la donnée membre est une association N :1.

La classe `Departement` est elle aussi transformée en entité :

```java
package fr.univaix.iut.progbd.tutoJPA;
import javax.persistence.*;

@Entity
public class Departement {
    @Id
    @GeneratedValue
```

```
    private long id;
    private String nom;
    private String telephone;
    public Departement() {}
    public Departement(long id, String nom, String telephone) {
        this.id = id;
        this.nom = nom;
        this.telephone = telephone;
    }

    public long getId() { return id; }
    public String getNom() { return nom; }
    public String getTelephone() { return telephone; }
}
```

La classe `Adresse` doit être annotée par l'annotation `@javax.persistence.Embeddable` pour pouvoir être intégrée dans la classe `Employe` :

```
package fr.univaix.iut.progbd.tutoJPA;
import javax.persistence.*;

@Embeddable
public class Adresse {
    private int numero;
    private String rue;
    private String codePostal;
    private String ville;

    public Adresse() {}
    public Adresse(int numero, String rue, String codePostal, String ville) {
        this.numero = numero;
        this.rue = rue;
        this.codePostal = codePostal;
        this.ville = ville;
    }
    public int getNumero() { return numero; }
    public String getRue() { return rue; }
    public String getCodePostal() { return codePostal; }
    public String getVille() { return ville; }
}
```

2.4 Contexte de persistance

Les paramètres de la connexion à la base de données sont définis dans le fichier `persistence.xml`. Ce fichier doit être situé dans le dossier `META-INF` du `jar` de l'application. Ces paramètres seront utilisés par la suite par le gestionnaire d'entités pour établir la connexion au SGBD.

Pour que Maven place ce fichier au bon endroit à la construction du `jar`, il le faut mettre dans le dossier `src/main/resources/META-INF`.

```
<?xml version="1.0" encoding="UTF-8" ?>

<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
 version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="employePU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>fr.univaix.iut.progbd.tutoJPA.Employe</class>
    <class>fr.univaix.iut.progbd.tutoJPA.Departement</class>
    <class>fr.univaix.iut.progbd.tutoJPA.Adresse</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
                value="jdbc:mysql://localhost:3306/employeBD"/>
      <property name="javax.persistence.jdbc.driver"
```

```
                value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.user"  value="monUser"/>
      <property name="javax.persistence.jdbc.password"  value="monPassword"/>
      <property name="eclipselink.ddl-generation"  value="create-tables"/>
    </properties>
  </persistence-unit>
</persistence>
```

D'après le fichier `persistence.xml` l'application se connectera à la base `employeBD` du serveur MySQL local avec l'utilisateur `"monUser"` et le mot de passe `"monPassword"`. Pour paramétrer correctement le serveur local, il faut exécuter les commandes suivantes :

```
$mysql --user=root --password=mysql --execute="create database employeBD"
$mysql --user=root --password=mysql \
       --execute="grant all privileges on employeBD.* to monUser@localhost
          identified by 'monPassword'"
$mysql --user=root --password=mysql --execute="show databases"
+--------------------+
| Database           |
+--------------------+
| information_schema |
| employeBD          |
| mysql              |
+--------------------+

$mysql --user=monUser --password=monPassword
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43
Server version: 5.1.58

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use employeBD;
Database changed
mysql> show tables;
Empty set (0.00 sec)
```

Comme on peut le voir pour l'instant notre base de données est totalement vide.

2.5 Programme principal

Maintenant que l'entité `Employe` est développée et compilée, nous allons écrire une classe principale qui permettra de créer un objet `Employe` et de le rendre persistant. Pour cela, nous avons besoin d'initialiser le `EntityManager` par une factory, de démarrer une transaction, créer une instance de l'objet, définir le nom et le salaire de l'employé, utilisez `EntityManager.persist()` pour l'insérer dans la base de données, valider la transaction et fermer l'`EntityManager`.

```
package fr.univaix.iut.progbd.tutoJPA;
import javax.persistence.*;

public class App
{
    public static void main(String[] args) {
        // Initializes the Entity manager
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("
            employePU");
        EntityManager em = emf.createEntityManager();
        EntityTransaction tx = em.getTransaction();

        // Creates a new object and persists it
```

```
        Employe employe = new Employe();
        employe.setNom("Dupont");
        employe.setSalaire(5000);
        tx.begin();
        em.persist(employe);
        tx.commit();

        em.close();
        emf.close();
    }
}
```

Avant de lancer ce programme, il faut ajouter dans le fichier `pom.xml` les dépendances au connecteur MySQL et à EclipseLink :

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
</dependency>

<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>eclipselink</artifactId>
    <version>2.2.0</version>
</dependency>
```

Maintenant nous allons utiliser la commande Maven `clean compile` pour compiler notre projet et le plugin `exec` pour lancer la classe principale.

```
$mvn clean compile
$mvn exec:java -Dexec.mainClass="fr.univaix.iut.progbd.tutoJPA.App"
```

Lorsque nous lançons la classe `fr.univaix.iut.progbd.tutoJPA.App` plusieurs choses vont se produire :

- Comme la propriété `eclipselink.ddl-generation` est initialisée à `create-tables` dans le fichier `persistence.xml`, les différentes tables sont créées si tel n'était pas le cas.
- L'employé "Dupont" est inséré dans la base de données (avec un identifiant automatiquement généré).

Regardons l'état de la base de données pour comprendre ce qui s'est passé.

```
$ mysql --user=monUser --password=monPassword employeBD
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 54
Server version: 5.1.58-1ubuntu1 (Ubuntu)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show tables;
+---------------------+
| Tables_in_employeBD |
+---------------------+
| DEPARTEMENT         |
| EMPLOYE             |
| SEQUENCE            |
+---------------------+
3 rows in set (0.00 sec)
```

```
mysql> describe EMPLOYE;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| ID             | bigint(20)   | NO   | PRI | NULL    |       |
| NOM            | varchar(50)  | YES  |     | NULL    |       |
| SALAIRE        | bigint(20)   | YES  |     | NULL    |       |
| CODEPOSTAL     | varchar(255) | YES  |     | NULL    |       |
| NUMERO         | int(11)      | YES  |     | NULL    |       |
| RUE            | varchar(255) | YES  |     | NULL    |       |
| VILLE          | varchar(255) | YES  |     | NULL    |       |
| DEPARTEMENT_ID | bigint(20)   | YES  | MUL | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
8 rows in set (0.00 sec)

mysql> select * from EMPLOYE;
+-----+--------+---------+------------+--------+------+-------+----------------+
| ID  | NOM    | SALAIRE | CODEPOSTAL | NUMERO | RUE  | VILLE | DEPARTEMENT_ID |
+-----+--------+---------+------------+--------+------+-------+----------------+
|   1 | Dupont |    5000 | NULL       |   NULL | NULL | NULL  |           NULL |
+-----+--------+---------+------------+--------+------+-------+----------------+
1 rows in set (0.00 sec)
```

JPA associe une relation à chaque classe marquée par l'annotation `@Entity` et les données membres sont converties en attribut. Il gère aussi les associations N :1 en créant la clef étrangère dont le nom est construit à partir du nom de la clef et de la table liée. La classe `Adresse` est directement intégrée dans la table de l'entité `Employe`. Pour les clef auto-générée, JPA utilise une table nommée `SEQUENCE` pour mémoriser les identifiants déjà attribués.