

# JPA : Java Persistence API



P.Mathieu

LP DA2I Lille  
<http://www.iut-a.univ-lille.fr>  
[prenom.nom@univ-lille.fr](mailto:prenom.nom@univ-lille.fr)

6 février 2019

## 1 JPA

## JPA Présentation



- JPA : Java Persistence API
- JPA est une norme, une spécification qui impose à un ORM un fonctionnement précis
- JPA fournit les bases d'un framework respectant un Design Pattern DAO
- JPA est défini dans le package `javax.persistence`
- Définit un mapping Objet-Relationnel assurant la persistance des objets métier
- Fournit le langage JPQL (Java Persistence Query Language)
- Fonctionne à partir d'annotations : `Entity`, `Id`, `Table`, `Column`

## JPA Implémentations



Plusieurs ORM implémentent JPA

- EclipseLink (implémentation de référence)
- Hibernate (la plus connue)
- OpenJPA
- TopLink
- DataNucleus
- OrmLite, Jdbi, JEasyOrm, ...

- Définir le système de persistance dans un fichier XML  
META-INF/persistence.xml
- Créer des POJO pour des entités avec les bonnes annotations
- Dans les programmes, utiliser un gestionnaire d'entités (EntityManager) qui gère la persistance et permet de manipuler les objets via le CRUD

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  <persistence-unit name="testjpa" transaction-type="RESOURCE_LOCAL">
    <class>da2i.Client</class>
    ...
    <class>da2i.Fournisseur</class>
    <properties>
      <property name="eclipselink.jdbc.batch-writing" value="JDBC"/>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:postgresql://localhost/template1"/>
      <property name="javax.persistence.jdbc.user" value="mathieu"/>
      <property name="javax.persistence.jdbc.driver" value="org.postgre
    </properties>
  </persistence-unit>
</persistence>
```

```
@Entity
@NamedQuery(name="Client.findAll", query="SELECT c FROM Client c")
public class Client implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String nom;
    private String prenom;

    // accesseurs
    ....
}
```

voir la javadoc de [javax.persistence](#)

```
@Entity
@Id

@Table(name="xxx")
@Column(name="xxx")
@GeneratedValue(strategy=GenerationType.xxx)

@OneToMany(mappedBy="xxx")
@ManyToOne
@JoinColumn(name="xxx", referencedColumnName="yyy")

@Embeddable // pour une classe de definition d'une clé multi-att

@NamedQuery
```

```
public class MonProg
{
    public static void main(String[] args)
    {
        // Création d'un EntityManager
        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("testjpa");
        EntityManager em = emf.createEntityManager();

        // Création d'un nouveau client
        Client client = new Client();
        client.setId(1);
        client.setNom("Mathieu");
        client.setPrenom("Philippe");
        em.getTransaction().begin();
        em.persist(client);
        em.getTransaction().commit();
    }
}
```

Parmi les propriétés du `persistence.xml` la propriété `javax.persistence.schema-generation.database.action` permet de spécifier ce qui se passe à chaque lancement

- none (default)
- create
- drop-and-create
- drop

L'IDE contient aussi en général un outil permettant de faire le passage du SGBD aux POJO et vice-versa

### Scripts SQL associés :

- `javax.persistence.schema-generation.database.action`  
action à effectuer sur le schema de la BDD à chaque lancement :  
none (default), create, drop-and-create et drop
- `javax.persistence.schema-generation.create-script-source`  
le nom du fichier SQL de création de la base
- `javax.persistence.schema-generation.drop-script-source`  
le nom du fichier SQL de suppression de la base
- `javax.persistence.sql-load-script-source`  
le nom du fichier SQL qui remplit les données au démarrage

### Génération de scripts :

- `javax.persistence.schema-generation.scripts.action`  
types de scripts à générer lors de la création : none (default), create, drop-and-create and drop.
- `javax.persistence.schema-generation.create-source`  
Contenu du fichier de génération : metadata, script, metadata-then-script et script-then-metadata.
- `javax.persistence.schema-generation.drop-source`  
types de scripts à générer lors de la suppression : metadata, script, metadata-then-script et script-then-metadata.
- `javax.persistence.schema-generation.scripts.create-target`  
fichier à créer lors de la création
- `javax.persistence.schema-generation.scripts.drop-target`

EntityManager contient quelques méthodes génériques pour manipuler les objets

- **persist** – permet de rendre l'objet persistant  
`em.persist(client);`
- **find** – permet de rechercher un POJO sur sa clé  
`Client x = em.find(Client.class, 17);`
- **contains** permet de savoir si l'em manage cet objet (S'il l'a en mémoire)  
`boolean b = em.contains(client);`
- **remove** permet de détruire un objet (il doit avoir été récupéré)  
`em.remove(client);`
- plusieurs méthodes de définition pour ses propres requêtes

- **createNamedQuery(String)**
  - ▶ Crée une requête à partir d'une définition placée dans le POJO
  - ▶ à utiliser pour toutes les requêtes standard réutilisables (findById, findAll,...)
- **createQuery(String)**
  - ▶ Crée une requête à partir d'une description JPQL
  - ▶ à utiliser pour des requêtes créées dynamiquement (dans des boucles par ex)
- **createNativeQuery(String)**
  - ▶ Crée une requête à partir d'une description SQL dépendant du SGBD sous-jacent
  - ▶ à utiliser pour des requêtes complexe, non supportées par JPQL

Les méthodes `getResultList()`, `getSingleResult()`, `getMaxResults()`, `executeUpdate()` exécutent la requête

Requêtes réutilisables définies dans le POJO en JPQL

```
@Entity
@NamedQuery(name="Client.findAll", query="SELECT c FROM Client c")
public class Client implements Serializable
{...}
```

```
List<Client> result = em.createNamedQuery("Client.findAll").getResultList();

for (Client c:result)
    System.out.println(c);
```

Requêtes créées dynamiquement en JPQL

```
List<Client> result = em.createQuery(
    "Select c from Client c where c.age<30").getResultList();

for (Client c:result)
    System.out.println(c);
```

## Requêtes écrites en langage natif

```
List<Client> result = em.createNativeQuery(
    "Select * from client", Client.class).getResultList();

for (Client c : result)
    System.out.println(c);
```

```
Query query = em.createQuery(
    "Select c from Client c where c.age < :age");
query.setParameter("age", 18);

List<Client> result = query.getResultList();
for (Client c: result)
    System.out.println(c);
```

```
Query query = em.createQuery(
    "DELETE FROM Client c WHERE c.age < :age");
int nb = query.setParameter("age", 18).executeUpdate();
```

Toutes les opérations se font entre un `em.begin()` et un `em.commit()`

- Création : `em.persist(o)`
- MàJ : si l'objet est persistant, il suffit de le modifier
- Effacement : `em.remove(o)`
- Recherche : `em.find(Classe, clé)`

- 1 : 1 mono-directionnel @OneToOne sur l'attribut lien
- 1 : 1 bi-directionnel @OneToOne (mappedBy="col-liée") sur l'autre attribut lien
- 1 : n mono-directionnel @OneToMany sur la collection côté n
- 1 : n bi-directionnel @OneToMany (mappedBy="fkey" sur la collection côté 1 et ManyToOne sur l'attribut clé étrangère
- n : m mono-directionnel @ManyToMany d'un côté
- n : m mono-directionnel @ManyToMany (mappedBy=" " d'un côté et @ManyToMany de l'autre

- S'assurer d'avoir son driver jdbc et son driver ORM
- File new JPA project (ouvre automatiquement la perspective JPA)
- Database Connection , new , configurer la base
- Click droit sur la base, tester ping
- Créer un projet JPA
- double click sur persistence.xml, onglet Connection, mettre resource Local, JDBC puis
- Ajouter le driver JDBC au projet
- Click droit sur le nom de projet, JPA Tools (generate tables from entities, generate entities from tables)

```
<project>
....
<dependencies>

<dependency>
<groupId>org.eclipse.persistence</groupId>
<artifactId>eclipselink</artifactId>
<version>2.7.3</version>
</dependency>

<dependency>
<groupId>org.eclipse.persistence</groupId>
<artifactId>javax.persistence</artifactId>
<version>2.1.1</version>
</dependency>

</dependencies>
....
</project>
```

```
projet
|-- pom.xml
|-- run.sh
|-- src
    |-- main
        |-- java
            |-- fr
                |-- da2i
                    |-- App.java
        |-- resources
            |-- META-INF
                |-- persistence.xml
```

Le META-INF/persistence.xml doit être placé dans resources

# JPA

## Arborescence TOMCAT + MAVEN

```
projet
|-- pom.xml
|-- run.sh
|-- src
    |-- main
        |-- java
        |   |-- fr
        |       |-- da2i
        |           |-- Servlet1.java
        |-- resources
        |   |-- META-INF
        |       |-- persistence.xml
        |-- webapp
            |--META-INF
            |   |-- context.xml
            |-- WEB-INF
            |   |-- web.xml
            |-- pagel.jsp
```

**Le META-INF/persistence.xml doit etre placé dans resources**

**Le META-INF/context.xml doit etre placé dans webapp**