Persistence objet avec JPA

V2, 15/12/2020

Plan

- **Généralités**
- **Entités**
- **Configuration**
  - Prérequis
  - Mapping
  - Fichier persistence.xml
- **Utilisation**
  - Contexte / unité de persitence
  - EntityManager
  - Transactions
- **Callbacks**
- **JPQL**
- **Autres concepts**

Généralités

JPA = Java Persistence API

Spécification inclus dans Java EE à partir de la version 5

Il s'agit d'une ORM (Object-Relationnal Mapping)

Plus précisément, c'est une interface d'ORM ; il appartient donc à des fournisseurs tierces d'en fournir les implémentations

Généralités

Plusieurs implémentations sur le marché actuellement

- Hibernate
- OpenJPA
- EclipseLink
- ...

L'intérêt principal de JPA est de définir une interface unique pour le mapping objet-relationnel.

Plan

- **Généralités**
- **Entités**
- **Configuration**
  - Prérequis
  - Mapping
  - Fichier persistence.xml
- **Utilisation**
  - Contexte / unité de persitence
  - EntityManager
  - Transactions
- **Callbacks**
- **JPQL**
- **Autres concepts**

Entités

Objet dans le domaine de persistance

Représente une table dans une base de données relationnelle, et chaque instance de cette entité correspond à un enregistrement de table

C’est un simple POJO

Plan

- **Généralités**
- **Entités**
- **Configuration**
  - Prérequis
  - Mapping
  - Fichier persistence.xml
- **Utilisation**
  - Contexte / unité de persitence
  - EntityManager
  - Transactions
- **Callbacks**
- **JPQL**
- **Autres concepts**

Configuration

La configuration pour l'ORM consiste en :

- Le choix et l'intégration de l'implémentation JPA au projet
- Le mapping proprement dit des classes entités
- Le fichier persistence.xml

Plan

- **Généralités**
- **Entités**
- **Configuration**
  - Prérequis
  - Mapping
  - Fichier persistence.xml
- **Utilisation**
  - Contexte / unité de persitence
  - EntityManager
  - Transactions
- **Callbacks**
- **JPQL**
- **Autres concepts**

Configuration > Prérequis

Une classe entité doit respecter les règles suivantes :

- Obligatoires
  - Être un javabean
  - Redéfinir les méthodes equals() et hashcode() sur l'/les attribut(s) clé primaire
  - L'attribut clé primaire doit être de type objet
- Recommandées (fortement)
  - Implémenter l'interface Serializable
  - Redéfinir la méthode toString()

Plan

- **Généralités**
- **Entités**
- **Configuration**
  - Prérequis
  - Mapping
  - Fichier persistence.xml
- **Utilisation**
  - Contexte / unité de persitence
  - EntityManager
  - Transactions
- **Callbacks**
- **JPQL**
- **Autres concepts**

Configuration > Mapping

2 façons différentes de faire le mapping

- Annotations
- Fichier XML

Nous abordons ici le mapping avec les annotations

Configuration > Mapping

Annotations obligatoires

@javax.persistence.Entity

déclare la classe comme classe entité

@javax.persistence.Id

déclare l'attribut comme correspondant à la colonne clé primaire de la table

Configuration > Mapping

Mapping entre le bean entité et la table

@javax.persistence.Table

Préciser le nome et autres propriétés de la table concernée par le mapping

@javax.persistence.Column

Associer un champ de la table à la propriété

@javax.persistence.GeneratedValue

Demander la génération automatique de la clé primaire au besoin

@javax.persistence.Basic

Représenter la forme de mapping la plus simple ; utilisé par défaut

@javax.persistence.Transient

Ne pas tenir compte du champ lors du mapping

Configuration > Mapping

Mapping d’associations

@javax.persistence.OneToOne

@javax.persistence.ManyToOne

@javax.persistence.OneToMany

@javax.persistence.ManyToMany

Indique que l’attribut provient d’une association, et indique la multiplicité

@javax.persistence.JoinColumn

Permet de configurer la colonne « clé étrangère » d’une relation 1 - *

@javax.persistence.JoinTable

Permet de configurer la table d’association d’une relation * - *

Configuration > Mapping

Mapping de l’héritage

- @javax.persistence.MappedSuperclass
  - Les champs de la classe annotée sont persistantes dans la base de donnée
  - Il n'est pas possible de faire des requêtes sur cette classe
  - La classe annotée ne sera pas mappée sur une table dédiée
  - La classe doit être abstraite et ne pas posséder d’annotation @Entity

Configuration > Mapping

Mapping de l'héritage

- @javax.persistence.Inheritance
  - Annotation principale
- Trois stratégies
  - single table (SINGLE_TABLE) : une seule table est utilisée pour toutes les classes de la hiérarchie
  - joined subclass (JOINED) : une table est utilisée pour chaque classe de la hiérarchie
  - table per class (TABLE_PER_CLASS) : une table est utilisée pour chaque classe concrète
- @javax.persistence.DiscriminatorColumn
  - Stratégies SINGLE_TABLE et JOINED
- @javax.persistence.DiscriminatorValue
  - Stratégies SINGLE_TABLE et JOINED

Configuration > Mapping

Mapping de clés composites

@javax.persistence.IdClass

Permet de mapper une clé composite

@javax.persistence.EmbeddedId

Permet de mapper une clé composite

À utiliser conjointement avec @Embeddable

@javax.persistence.Embeddable

Indique que la classe n'est pas une entité, mais que ces attributs mappées, seront intégrées à la classe entité dans laquelle elle sera utilisée (comme type d'un attribut) ;

L'attribut dans la classe entité doit être marquée @Embedded

@javax.persistence.Embedded

(Voir ci-dessus)

Configuration > Mapping

Autres mapping

- @javax.persistence.Lob
  - Propriété sur une colonne de type Blob ou Clob
- @javax.persistence.Enumerated
  - Propriété de type énumération

Plan

- **Généralités**
- **Entités**
- **Configuration**
  - Prérequis
  - Mapping
  - Fichier persistence.xml
- **Utilisation**
  - Contexte / unité de persitence
  - EntityManager
  - Transactions
- **Callbacks**
- **JPQL**
- **Autres concepts**

Configuration > persistence.xml

Contient la configuration de base pour le mapping,

Notamment en fournissant les informations sur la connexion à la base de données à utiliser

Doit être stocké dans le répertoire META-INF

Plan

- **Généralités**
- **Entités**
- **Configuration**
  - Prérequis
  - Mapping
  - Fichier persistence.xml
- **Utilisation**
  - Contexte / unité de persitence
  - EntityManager
  - Transactions
- **Callbacks**
- **JPQL**
- **Autres concepts**

Utilisation

L’utilisation demande une bonne compréhension des concepts de

- Unité de persistance
- Contexte de persistance
- EntityManager

Utilisation > Contexte / unité de persitence

Unité de persistance

= Ensemble défini de classes entité

La définition est assurée dans un fichier de description nommé persistence.xml

Utilisation > Contexte / unité de persitence

EntityManager

= Objet dédié qui réalise toutes les actions de persistance sur les objets entité

Contexte de persistance

= Ensemble d'entités géré par un EntityManager

= « Base de donnée objet virtuelle » exposée par l'ORM à l'application

Les objets entités peuvent être de 2 types

Géré (Managed)

Non géré (Unmanaged)

Lorsqu'un contexte de persistance est fermé, toutes les entités du contexte deviennent « non gérés »

Utilisation > Contexte / unité de persitence

2 types de contexte de persistance

- Transaction-scoped
  - Le contexte est ouvert pendant toute la durée d'une transaction
  - La fermeture de la transaction entraîne la fermeture du contexte
  - N'est utilisable que dans le cadre de l'utilisation dans un conteneur (dans un serveur d'application JEE par exemple) qui va assurer la prise en charge de la transaction
- Extended persitence
  - Le contexte reste ouvert après la fermeture de la transaction

Utilisation > EntityManager

Assure les interactions entre la base de données et les objets entité

Assure aussi les interactions avec un éventuel gestionnaire de transactions (par exemple JTA dans un environnement JEE)

Utilisation > EntityManager

Il est obtenu :

Soit à travers un objet **EntityManagerFactory**

```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("MaBaseDeTestPU") ;
EntityManager em = emf.createEntityManager() ;
```

Soit par injection avec l'annotation **@PersistenceContext**

```
@PersistenceContext(unitName="MaBaseDeTestPU")
private EntityManager entityManager;
```

On peut éalement obtenir un objet **EntityManagerFactory** par injection

```
@PersistenceUnit(unitName="MaBaseDeTestPU")
private EntityManagerFactory factory;
```

Utilisation > EntityManager

Méthodes

```
boolean contains(Object)
void clear()
void flush()
void refresh(Object)
find(Class, Object)
void persist(Object)
Object merge(Object)
void remove(Object)
...
```

Utilisation > EntityManager

Les opérations de persistance (find(), persist(), merge(), remove(),...) doivent se faire OBLIGATOIREMENT dans une transaction

FlushModeType

- Contrôle le moment d'exécution des opérations de persistance (`persist()`, `merge()`,...)
- AUTO
  - Les mises à jour sont reportées dans la base de données avant chaque requête
- COMMIT
  - Les mises à jour sont reportées dans la base de données lors du commit de la transaction
  - Plus performant car il limite les échanges avec la base de données
- Il est possible de forcer l'enregistrement des mises à jour dans la base de données en utilisant la méthode `flush()`

Utilisation > EntityManager

Objet Query

Est obtenu à partir des méthodes suivantes de l'`EntityManager`

```
Query createQuery(String)
Query createNativeQuery(String)
...
```

Exécute des requêtes (JPQL, SQL,...) sur la base de données relationnelle sous-jacente

Méthodes

```
List getResultList()
Object getSingleResult()
int executeUpdate()
void setParameter(String, Object)
void setMinResult(int)
void setMaxResult(int)
```

Utilisation > Transactions

Rappel : Toutes les opérations de persistance doivent se faire dans une transaction

2 façons de gérer les transactions

- JTA
  - Disponible dans un serveur d'application JEE
  - Les méthodes des EJB sont par défaut transactionnelles
- Manuel : À utiliser
  - Soit en dehors d'un serveur d'application JEE
  - Soit si on désactive la gestion des transactions par le serveur d'application JEE

Utilisation > Transactions

Gestion manuelle des transactions

On utilise un objet EntityTransaction, récupéré avec la méthode suivante de EntityManaer

```
EntityTransaction getTransaction()
```

Méthodes

```
void begin()
void commit()
void roolback()
boolean isActive()
```

Plan

- **Généralités**
- **Entités**
- **Configuration**
  - Prérequis
  - Mapping
  - Fichier persistence.xml
- **Utilisation**
  - Contexte / unité de persitence
  - EntityManager
  - Transactions
- **Callbacks**
- **JPQL**
- **Autres concepts**

Callbacks

JPA permet de définir des « callbacks » (méthodes) qui seront appelés sur certains événements

Ils doivent être annotés

- @javax.persistence.PrePersist
- @javax.persistence.PostPersist
- @javax.persistence.PostLoad
- @javax.persistence.PreUpdate
- @javax.persistence.PostUpdate
- @javax.persistence.PreRemove
- @javax.persistence.PostRemove

Plan

- **Généralités**
- **Entités**
- **Configuration**
  - Prérequis
  - Mapping
  - Fichier persistence.xml
- **Utilisation**
  - Contexte / unité de persitence
  - EntityManager
  - Transactions
- **Callbacks**
- **JPQL**
- **Autres concepts**

JPQL

JPQL = Java Persistence Query langage

Langage pour effectuer des requêtes sur les entités JPA

Permet de

- Récupérer des objets avec des requêtes SELECT
- Mettre à jour des objets avec UPDATE et DELETE

EntityManager.createQuery() permet de créer et manipuler les requêtes JPQL

JPQL

Structure

- SELECT … FROM …
  [WHERE …]
  [GROUP BY … [HAVING …]]
  [ORDER BY …]

- DELETE FROM … [WHERE …]

- UPDATE … SET … [WHERE …]

JPQL

Fonctions

- Scalaires
  - UPPER, LOWER, ABS, CONCAT, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, INDEX, LENGTH, LOCATE, MOD, SIZE, SQRT, SUBSTRING, SQRT, TRIM
- d'Agrégation
  - MIN, MAX, AVG, COUNT, SUM

JPQL

Caractéristiques diverses

- Jointures
- Expressions BETWEEN, LIKE
- Sous-requêtes
- Expressions IN, MEMBER OF
- Expressions ANY, ALL, SOME
- Expressions EXISTS
- Expressions CASE
- @NamedQuery
- Eager and Lazy Loading

Plan

- **Généralités**
- **Entités**
- **Configuration**
  - Prérequis
  - Mapping
  - Fichier persistence.xml
- **Utilisation**
  - Contexte / unité de persitence
  - EntityManager
  - Transactions
- **Callbacks**
- **JPQL**
- **Autres concepts**

Autres concepts

Criteria API

Entity graph

Hints