HIBERNATE / JPA

RAFIQ IMANE

Master - Ingénierie des Systèmes d'Information

Hibernate

Sommaire

- Présentation
- Mapping Objet-relationnel
- Configuration Hibernate
- Exemple de persistance
- HQL

Présentation

Notions de base

- Hibernate est un outil ORM (Object-Relational Mapping) open source pour Java

- Hibernate automatise ou facilite la correspondance entre des données stockées dans des objets et une base de données relationnelle

- Le plus souvent les données sont décrites dans des fichiers de configuration (XML ou Java avec JPA (Java persistence Api)) <On verra les deux dans l'exemple pratique>

Présentation

Fonctionnalités de base

- Recherche et enregistre les données associées à un objet dans une base de données

- Détecte la propriété modifiée d'objet et l'enregistre en optimisant les accès à la base

Présentation

Avantages

- Évite l'écriture de code répétitif
- Gain de 30 à 40 % du nombre de lignes de certains projets
- Améliore la portabilité du code pour des changements de SGBD

Présentation

Avantages

- Le développeur pense en termes d'objet et pas en termes de lignes de tables
- Sans outil ORM le développeur peut hésiter à concevoir un modèle objet « fin » afin d'éviter du codage complexe pour la persistance
- Le refactoring du schéma de la base de données ou du modèle objet est facilité

Présentation

Pas toujours bénéfique

- Un type d'applications ne bénéficie pas de l'utilisation d'un outil ORM : celles qui modifient un grand nombre de lignes pour chaque update ou qui ne comportent essentiellement que des requêtes select de type « group by »

- En effet, en ce cas la manipulation d'un grand nombre d'objets nuit aux performances

- Par exemple les applications OLAP (online analytical processing), big data….

Mapping objet-relationnel

Fichier de mapping

- Décrit comment se fera la persistance des objets d'une classe
- Format XML
- Se place dans le même répertoire que la classe et se nomme Classe.hbm.xml si la classe s'appelle Classe

Mapping objet-relationnel

Exemple de fichier de mapping (EN-TÊTE)

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping
-3.0.dtd">
```

Mapping objet-relationnel

Exemple de fichier de mapping (Formation.hbm.xml)

```
<hibernate-mapping>
   <class name="fr.fssm.java.Formation" table="FORMATION">
      <id name="id" column="FORMATION_ID" />
      <generator class="increment" />
      <property name="titre" column="TITRE" />
      <many-to-one name="ecole" column="ECOLE_ID" class="fr.fssm.java.Ecole" />
   </class>
</hibernate-mapping>
```

Mapping objet-relationnel

Classe Formation.java

JAVA

```java
public class Formation {

    private Long id;

    private String titre;

    private Ecole ecole;

    ....

    // Getters & Setters

    ....
}
```

Mapping objet-relationnel

Exemple de fichier de mapping (Ecole.hbm.xml)

```
<hibernate-mapping>
   <class name="fr.fssm.java.Ecole" table="ECOLE">
      <id name="id" column="ECOLE_ID">
         <generator class="increment" />
      </id>
      <property name="nom" column="NOM" />
      <set name="formations">
         <key column="FORMATION_ID" />
         <one-to-many class="fr.fssm.java.Formation" />
      </set>
   </class>
</hibernate-mapping>
```

Mapping objet-relationnel

Classe Ecole.java

```java
public class Ecole {

    private Long id;

    private String nom;

    private Set<Formation> formations;

    ....

    // Getters & Setters

    ....
}
```

Mapping objet-relationnel

Tags Hibernate pour les associations

- Pour les collections : `<set>`, `<list>`, `<map>`, `<bag>`, `<array>` et `<primitive-array>`
- Les cardinalités avec les tags suivants : `<one-to-one>`, `<one-to-many>`, `<many-to-one>`, `<many-to-many>` (on ajoutera `<join>` pour un cas particulier)

Configuration Hibernate

Notions de base

- Il faut décrire en particulier le **SGBD** utilisé pour la persistance

- La configuration est contenue dans un fichier **`hibernate.cfg.xml`** placé dans le classpath

Configuration Hibernate

Session Factory

Un tag **`session-factory`** par base de données (le plus souvent, une seule base) :

```
<hibernate-configuration>
  <session-factory>
    </session-factory>
    <session-factory>
      . . .
   </session-factory>
</hibernate-configuration>
```

Configuration Hibernate

Session Factory

Chaque session-factory contient les informations pour obtenir une connexion à la base, le dialecte SQL (Oracle, DB2,…) et divers autres informations

Configuration Hibernate

Exemple d'un fichier Hibernate.cfg.xml

```
<hibernate-configuration>
   <session-factory>
      <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
      <property name="connection.url">jdbc:hsqldb:hsql://localhost</property>
      <property name="connection.username" />
      <property name="connection.password" />
      <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
      <property name="hbm2ddl.auto">create</property>
      <property name="show_sql">true</property>
      <mapping resource="mappings/Formation.hbm.xml" />
      <mapping resource="mappings/Ecole.hbm.xml" />
   </session-factory>
</hibernate-configuration>
```

Exemple de persistance

Implémentation d'une classe utile pour instancier la session factory

JAVA

```java
public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch(Throwable t) {
            System.err.println("Echec de création de la SessionFactory : " + t);
            throw new ExceptionInInitializerError(t);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
```

Configuration Hibernate

Méthodes de gestion des données

- **Save (objet)**
- **Load (CLASS , new Integer(1)) // getById();**
- **Update (objet)**
- **Delete (objet)**

Exemple de persistance

Création d'une formation

JAVA

```java
public class Exemple1 {

    public static void main (String[] args){
        Session session = HibernateUtil.getSessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Formation formation = new Formation("Hibernate");

        Long formationId = (Long)session.save(formation);
        System.out.println("Clé primaire : " + formationId);

        tx.commit();
        session.close();
    }
}
```

Exemple de persistance

Création d'une école avec une formation

JAVA

```
public class Exemple2(

  public static void main (String[] args) {

        Session session = HibernateUtil.getSessionfactory.openSession();
        Transaction tx = session.beginTransaction();
        Ecole ecole = new Ecole("FSSM”);
        Long ecoleId = (Long)session.save(ecole);
        ecole.setId(ecoleId);
        Formation formation = new Formation("Hibernate”);
        formation.setEcole(ecole);
        session.save(formation);
        tx.commit();
        session.close();

  }
}
```

HQL

Requête hibernate

- Hibernate Query Language
- Syntaxe entre SQL et Java
- Sélection des formations

```sql
(SELECT * )FROM Formation
```

SQL

  - Formation <=> Classe Java / ~~Table~~
  - Case sensitive
- Accès à toutes les notions du langage SQL
- Possibilité d'effectuer des Update / Delete

HQL

3 façons de retrouver les données

- **Langage HQL :**

```
session.createQuery("from Employe e where e.nome like 'Dup%'");
```

- **L'API Criteria pour QBC (*query by criteria*) et pour QBE (*query by example*) :**

```
session.createCriteria(Employe.class).add(Expression.like("nome", "Dup%"));
```

- **SQL direct avec mapping automatique du résultat avec les objets :**

```
session.createSQLQuery("select {e.*} from EMPLOYE {e} where NOM like
'Dup%'", "e", Employe.class);
```

Java Persistence API

Sommaire

- Présentation
- Entity Beans
- Annotations
- EntityManager
- Fichier persistence.xml
- API Criteria

Présentation

La spécification Java persistence API

- **Java Specification Request (317)**
  - Ne fonctionne pas tout seul
  - Besoin d'une implémentation

- **Différentes implémentations :**
  - Hibernate - TopLink - EclipseLink

- **Java Persistence Query Language :**
  - Langage de requêtage
  - Équivalant à HQL pour Hibernate
  - Obligation de préciser la clause SELECT

- **Gestion d'entités par l'EntityManager**

Entity Beans

- **1 Entity <=> 1 Table :**
  - Lien faits par Java Annotations
- **Plain Old Java Object :**
  - Objet de données
  - Constructeur par défaut
  - Getter / Setter
  - Aucune interface nécessaire
- **Annotation :**
  @Entity

```java
@Entity
public class Formation {
    ....
}
```

Annotations

- Association d'une table en BDD :

```java
@Entity
@Table(name = "FORMATION")
public class Formation {
    ....
}
```

- Association d'une colonne de la table :

```java
@Column(name = "TITRE", nullable = false, length = 25)
private String titre;
```

Annotations

Définition de l'identifiant technique

JAVA

```java
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
@Column(name = "ID", unique = true, nullable = false)
private Long id;
```

Annotations

Définition de type enuméré

```java
@Enumerated(EnumType.ORDINAL)
@Column(name= "TYPE_FORMATION_ID", nullable = true)
private TypeFormation type;
```

Annotations

Relation entre les Entity Beans

JAVA

```java
@OneToOne
@JoinColumn(name = "DIRECTEUR_ID", nullable = false)
private Personne directeur;

@ManyToOne
@JoinColumn(name = "ECOLE_ID", nullable = false)
private Ecole ecole;

@OneToMany(fetch = FetchType.LAZY, mappedBy = "formation");
private List<Module> modules;
```

Annotations

Relation entre les Entity Beans (suite)

```
@ManyToMany
@JoinTable(name = "ASS_FORMATION_ETUDIANTS",
           joinColumns = {@JoinColumn(name="FORMATION_ID", referencedColumnName="ID")}
           inverseJoinColumns = {@JoinColumn(name="ETUDIANT_ID", referencedColumnName="ID")}
private List<Personne> etudiants;
```

EntityManager

Gestion des intéractions entre l'application et BDD :

- Recherche
- Création
- Suppression
- Modification

- **Intéraction avec le gestionnaire de transactions (Si défini)**

- **Configuration via le fichier persistence.xml**

- **Agit comme un "cache des Entity Beans :**
  - Mise à jour BDD (éventuellement) différée

- **Création par le conteneur JEE**

EntityManager

Récupération de l'EntityManager

- Via l'EntityManagerFactory :

```java
@PersistenceUnit(unitName = "myPersistenceUnit")
private EntityManagerFactory entityManagerFactory;

private EntityManager entityManager = entityManagerFactory.createEntityManager();
```

- Directement :

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

EntityManager

Sélection d'une entité via l'EntityManager

- Récupération directe d'une entité par son identifiant :

```java
Formation formation = this.entityManager.find(Formation.class, 1L);
```

- Requête JPQL typée :

```java
List<Formation> formations =
        this.entityManager.createQuery("SELECT * FROM FORMATION", Formation.class)
                                                    .getResultList();
```

- Requête JPQL non typée :

```java
Formation formation =
        (Formation) this.entityManager.createQuery("SELECT * FROM FORMATION WHERE ID = 1")
                                                    .getSingleResult();
```

EntityManager

- Création d'une entité :

```java
Formation formation = new Formation("Hibernate");
this.entityManager.persist(formation);
```

- Suppression d'une entité :

```java
Formation formation = new Formation(1L, "Hibernate");
this.entityManager.remove(formation);
```

- Modification d'une entité :

```java
Formation formation =  new Formation(1L, "JPA");
formation = this.entityManager.merge(formation);
```

Fichier persistence.xml

- **Fichier de configuration de JPA**
- **A mettre dans le dossier META-INF**
- **Définition :**
  - Nom de l'unité de persistence (PersistenceUnit)
  - Gestionnaire de transaction
  - Fournisseur de persistence (Provider)
  - DataSource
  - Les classes d'entités
  - Propriétés

Fichier persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"version="2.1">
   <persistence-unit name="PERSISTENCE">
      <description>Hibernate JPA Configuration Example</description>
      <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
      <class>net.javaguides.hibernate.entity.Student</class>
      <properties>
         <property name="javax.persistence.jdbc.driver"value="com.mysql.jdbc.Driver" />
         <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/hibernate_db"/>
         <property name="javax.persistence.jdbc.user"value="root" />
         <property name="javax.persistence.jdbc.password"value="" />
         <property name="hibernate.show_sql" value="true" />
         <property name="hibernate.hbm2ddl.auto" value="create-drop" />
         <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL55Dialect"/>
      </properties>
   </persistence-unit>
</persistence>
```

API Criteria

- **Librairie intégrée à JPA**

- **Abstraction totale du JPQL :**
  - Construction dynamique de la requête
  - Manipulation des Entity Beans Java

- **Récupération du CriteriaBuilder :**

```
CriteriaBuilder criteriaBuilder = this.entityManager.getCriteriaBuilder();
```

- **Création d'une requête :**

```
CriteriaQuery<Formation> query = criteriaBuilder.createQuery(Formation.class);
```

API Criteria

- Définition du Root (From initial) :

```java
Root<Formation> from = query.from(Formation.class);
```

- Création d'une jointure :

```java
Join<Formation, Ecole> joinEcole = from.join("ecole");
```

  - **ecole** est le nom de l'attribut dans la classe Formation
  - Inner join par défaut :

```java
Join<Formation, Ecole> joinEcole = from.join("ecole", JoinType.LEFT);
```

API Criteria

Sélection

- CriteriaQuery construite à partir d'un EntityBean :
  - Retour d'un EntityBean ou d'une liste d'EntityBean :

```java
query.getSingleResult();

query.getResultList();
```

- Count :

```java
CriteriaQuery<Long> query = criteriaBuilder.createQuery(Long.class);
Root<Formation> from = query.from(Formation.class);
....
query.select(criteriaBuilder.count(from));
query.getSingleResult();
```

API Criteria

Les clauses (Predicates)

- Clause arithmétiques :

JAVA

```java
Join<Formation, Etudiant> etudiants = from.join("etudiants");
Predicate etudiantPlusDe23ans = criteriaBuilder.greaterThan(etudiants.get("age"), 23);
Predicate etudiantPlusDe23ans = criteriaBuilder.gt(etudiants.get("age"), 23);

Predicate etudiant23ansEtPlus = criteriaBuilder.ge(etudiants.get("age"), 23);

Predicate etudiantMoinsDe25ans = criteriaBuilder.lessThan(etudiants.get("age"), 25);
Predicate etudiantMoinsDe25ans = criteriaBuilder.lt(etudiants.get("age"), 25);

Predicate etudiant25ansEtMoins = criteriaBuilder.le(etudiants.get("age"), 25);
```

FIN DE LA PRÉSENTATION

Pour se référencer encore plus:

https://docs.jboss.org/hibernate/orm/3.6/reference/