Java EE Approfondi

-

Cours 2

Cours de 2e année ingénieur
Spécialisation « Génie Informatique »

Présentation

- Lier l'orienté objet et la base de données relationnelle peut être lourd et consommateur en temps.
- **Hibernate** se propose de joindre ces deux univers, à travers le **mapping objet/relationnel** (*Object-Relational Mapping* ou ORM)
- Le terme ORM décrit la technique consistant à faire le lien entre la représentation objet des données et sa représentation relationnelle.
- On peut voir **Hibernate** comme une fine surcouche de JDBC qui lui ajouterait une dimension objet.

Rappel JDBC

JDBC

- L' API **JDBC** (*Java DataBase Connectivity*) permet aux applications Java d'accéder par le biais d'une interface commune à des sources de données pour lesquelles il existe des pilotes JDBC (ex: BDD Oracle, MySQL, …).
- Il est constitué d'un ensemble de classes permettant de développer des applications capables de se connecter à des serveurs de bases de données (SGBD).
- Pour lier notre programme à la BDD, nous devons utiliser un Driver spécifique à cette BDD. Ces Drivers JDBC sont disponibles pour tous les systèmes connus de bases de données relationnelles.

Ex : oracle.jdbc.driver.OracleDriver pour Oracle.

- Hibernate se chargera automatiquement de la connexion à la BDD, il suffira de la lui paramétrer et de lui fournir le Driver correspondant.

Rappels Persistance

Persistance

- A la fin d'une session d'utilisation d'une application orientée objet toutes les données des objets existant dans la mémoire vive de l'ordinateur sont perdues.
- Rendre persistant un objet c'est sauvegarder ses données sur un support non volatile de telle sorte qu'un objet identique à cet objet pourra être recréé lors d'une session ultérieure.
- La persistance des données des objets peut se faire via une base de données relationnelle.

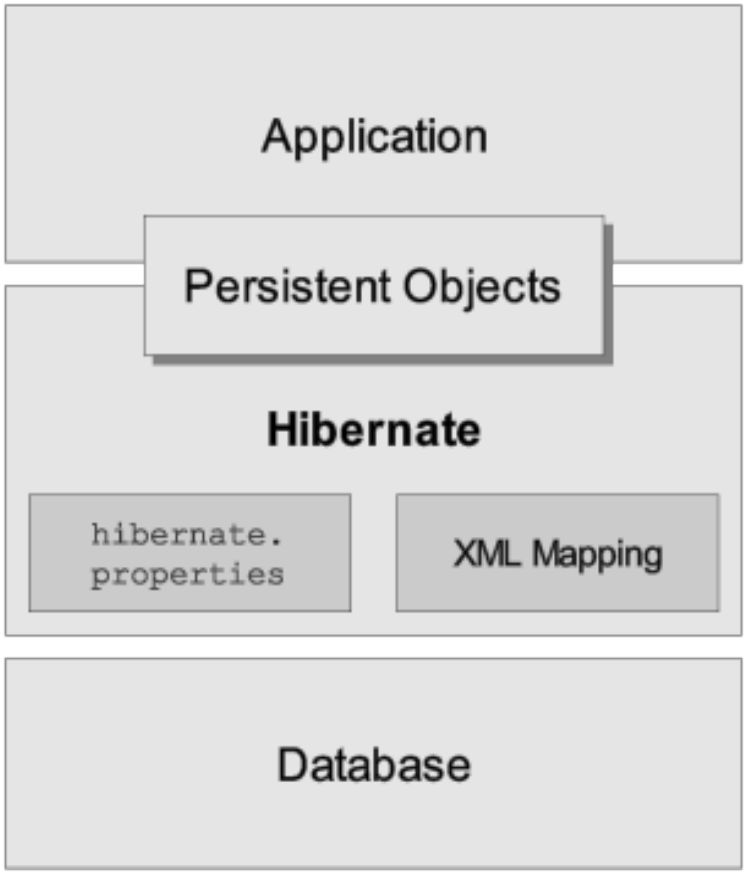
Persistance

- La gestion de la persistance des objets peut parfois être complexe :
  - A quel moment se connecte-t-on à la BDD?
  - Doit-on recréer tous les objets ou seulement les objets modifiés?
  - Quelle transformation (ou mapping) utiliser pour adapter la structure des données relationnelles au modèle objet?
  - Faut-il monter en mémoire l'ensemble des données utilisées par tous les objets impliqués dans une transaction en une seule fois ? Ou extraire les données de la base au fur et à mesure que les objets sont exécutés ?
- D'où l'utilité d'utiliser un framework se chargeant d'une partie de la gestion de la persistance.

Hibernate

Hibernate

- Voici comment se présente très globalement l'architecture d'Hibernate :



Hibernate

- Hibernate permet de manipuler facilement les objets persistants mais demande une configuration rigoureuse.
- La dernière version d'Hibernate (v3), permet de prendre en compte le traitement et la transformation des documents XML pour rendre leurs données facilement persistantes
- Sa configuration se fait par le biais de 2 type de fichiers :
  - **hibernate.properties** ou **hibernate.cfg.xml**

    Ces 2 fichiers ont la même utilité : lier Hibernate à notre BDD et au Driver à utiliser. L'unique différence étant que le second respecte la syntaxe XML. Les valeurs de **hibernate.cfg.xml** surchargent celles du fichier **hibernate.properties** si les deux fichiers sont présents.
  - **Nom_Classe.hbm** : Les fichiers de mapping, qui permettent de lier la BDD avec les objets persistants. Ils respectent la syntaxe XML.

Hibernate

Etape 1 :

Une fois que les librairies d'Hibernate sont incluses au projet, nous devons tout d'abord créer la BDD qui nous servira à stocker les objets persistants. (Hibernate peut également le faire tout seul à partir de JavaBean ou de POJO).

On crée alors une Table différente par Objet Persistant en lui donnant un attribut **id** qui permettra à Hibernate de lui associer une clef primaire et ainsi de différencier les objets de même type (il doit par ailleurs être défini en auto_increment).

Ex:

```
CREATE TABLE Etudiant (
    nom VARCHAR( 10 ) NOT NULL ,
    prenom VARCHAR( 10 ) NOT NULL ,
    age INT( 10 ) NOT NULL ,
    id INT( 10 ) NOT NULL AUTO_INCREMENT
)
```

Hibernate

- **Etape 1** : De même, il nous faut créer la classe représentant l'objet de la donnée Java que l'on veut rendre persistante.
- Pour cela nous pouvons écrire une classe sous forme de POJO ou de JavaBean(conseillé).
- Le plugin d'Eclipse que nous utiliserons permettra de générer automatiquement cette classe à partir des tables de la BDD.

Ex : Classe *Etudiant* sous forme de **JavaBean**

```
public class Etudiant implements Serializable{  // un JavaBean doit implémenter la classe Serializable
    private String nom;                          /*
    private String prenom;                       /*        Les attributs doivent être privés.
    private int age;                             /*        On les fera correspondre aux champs de la
    private int id;                              /*        Table dans le fichier de mapping.

    Etudiant()                                   // Le constructeur est vide
    {
    }

    public String getNom() { return Nom; }                  // Getter pour l'attribut nom
    public void setNom(String nom) { this.nom = nom; }      // Setter pour l'attribut nom

    …       // On écrit tous les getter et setter
}
```

Hibernate

- **Etape 2 :** Créer le fichier de configuration **hibernate.cfg.xml,** il contient les informations de connexion à la BDD.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
      "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory >

<!-- local connection properties -->
<property name="hibernate.connection.url">jdbc:mysql://localhost/test</property>  //URL de ma BDD
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property> //Driver à utiliser
<property name="hibernate.connection.username">login</property> //Login
<property name="hibernate.connection.password">password</property> //Password

<property name="current_session_context_class">thread</property>
<!-- dialect for MySQL -->
<property name="dialect">org.hibernate.dialect.MySQLDialect</property> //Mode de discussion entre Hibernate et la BDD
                                                                        // (existe par défaut pour chaque type de BDD)
<property name="hibernate.show_sql">false</property> // Hibernate affiche ou non toutes les requetes SQL passées à la BDD
<property name="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</property>
<mapping resource="Modele/Eleve.hbm" />  // fichier de Mapping
<mapping resource="Modele/Etudiant.hbm" />  // fichier de Mapping
</session-factory>
</hibernate-configuration>
```

Hibernate

- **Etape 3 :** Créer le ou les fichiers de mapping permettant de relier un objet Java à une Table de la BDD.

Ex : **Etudiant.hbm**

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

<hibernate-mapping >// Package dans lequel seront stockés nos JavaBean
    <class name= "Modele.Etudiant" table="etudiant"> // Lien objet / table
        /* Le champs Id est différent des autres champs */
        <id name="Id" type="integer" column="id"><generator class="native"/></id>
        /* Différents champs de cette Table et en quel attribut il sera mappé */
        <property name="Nom" column="nom" type="string" not-null="true" length="10"/>
        <property name="Prenom" column="prenom" type="string" not-null="true" length="10"/>
        <property name="Age" column="age" type="integer" not-null="true" length="10"/>
    </class>
</hibernate-mapping>
```

- **<u>Etape 4 :</u>** Arrivés à ce niveau, nous avons complètement initialisé le mode de fonctionnement d'Hibernate. Il ne reste plus qu'à nous en servir.
- Au démarrage, Hibernate crée un objet SessionFactory. Une SessionFactory peut ouvrir des nouvelles Sessions. Une Session représente une unité de travail simplement "threadée", la SessionFactory est un objet global "thread-safe", instancié une seule fois.
- Pour simplifier cette partie, nous allons créer une classe d'aide **HibernateUtil** qui s'occupera du démarrage et rendra la gestion des Sessions plus facile.
- Son implémentation est donnée dans l'aide d'Hibernate.

Hibernate

HibernateUtil.java

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

	static
	{
		try
		{
			// Crée  l'objet SessionFactory à partir de hibernate.cfg.xml
			sessionFactory = new Configuration().configure().buildSessionFactory();
		}
		catch (Throwable ex)
		{
			System.err.println("Initial SessionFactory creation failed." + ex);
			throw new ExceptionInInitializerError(ex);
		}
	}

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

Hibernate

- **Etape 5:** Nous pouvons maintenant ajouter des objets Etudiant dans la BDD comme suit :

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();

session.beginTransaction();
Etudiant e = new Etudiant(); // On crée l'objet étudiant
e.setAge(21);
e.setNom("Gibbon");
e.setPrenom("Jean");

session.save(e); // On demande à Hibernate de le sauvegarder dans la BDD

session.getTransaction().commit(); // On fait un commit
HibernateUtil.getSessionFactory().close();
```

Hibernate

- **Etape 5 bis** : On peut également récupérer des objets de la BDD:

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
List result = session.createQuery("from Etudiant").list(); // requête HQL
session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
```

- Le langage que nous utilisons pour faire des requêtes à la BDD est propriétaire à Hibernate et se nomme le **HQL**. Hibernate générera le SQL approprié et l'enverra à la BDD.
- Vous pouvez créer des requêtes plus complexes avec HQL, bien sûr.
- Il existe également des méthode de Session permettant d'update, remove, etc…

Webographie

- http://www.hibernate.org/hib_docs/v3/reference/fr-FR/html/
  => Documentation officielle Hibernate v3 en Français
- http://www.hibernate.org/
  => Site officiel
- http://defaut.developpez.com/tutoriel/java/eclipse/hibernate/
  => Présentation d'Hibernate et de son utilisation sous Eclipse