Accès aux objets
Hibernate

GUINKO Tonguim Ferdinand

3 novembre 2011

Sommaire

Persistance des données : définition

Une donnée persistante est :

- une donnée à laquelle on accède de façon irrégulière et qui n'est pas souvent modifiée ;
- une donnée qui existe et qui persiste d'une session à une autre.

Les données persistantes sont généralement stockées dans des bases de données.

Le contraire est donnée dynamique (ou transactionnelle) :

- les données dynamiques sont mises à jour de façon asynchrone ;
- les données dynamiques ne sont pas stockées dans des bases de données.

Persistance des données : définition

Mécanisme permettant à un objet de survivre au processus qui l'a créé.
Caractéristiques de la persistance :

- Stockage, organisation et récupération des données structurées (tri, agrégation) ;
- Concurrence et intégrité des données ;
- Partage des données.

Persistance des données : opérations de base

Le mécanisme de persistence renferme 4 opérations de base désignée généralement sous l'accronyme CRUD :

1. Create pour créer une nouvelle entité dans la base
2. Retrieve pour retrouver une ou plusieurs entités de la base
3. Update pour modifier une entités de la base
4. Delete pour supprimer une entité de la base

ORM : Object/Relational Mapping

- technique de programmation informatique qui :
  - crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé ;
  - crée une base de données orientée objet virtuelle ;
  - permettant de lier les bases de données relationnelles aux concepts de la POO ;
- correspondance entre le paradigme objet et le paradigme relationnel.

ORM : Object/Relational Mapping

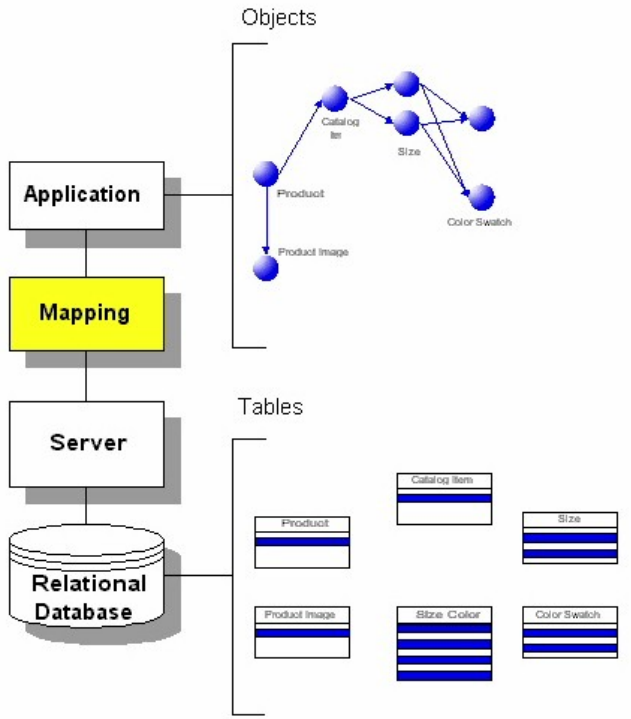
En d'autres termes, l'ORM définit :

- la persistance automatisée et transparente d'objets métiers vers une bases de données relationnelles ;
- decrit à l'aide de méta-données de la transformation réversible entre un modèle relationnel et un modèle de classes ;
- la capacité à manipuler des données stockées dans une base de données relationnelles à l'aide d'un langage de POO ;

Relation objet $\Longleftrightarrow$ SGBD : problématique

Problèmes de correspondance des modèles objet et relationnel :

- Identification des objets
- Traduction des associations
- Traduction de l'héritage
- Navigation entre les objets
- Objets dépendants



Relation objet $\Longleftrightarrow$ SGBD : problématique (... suite)

Comment établir la correspondance entre la classe et la table ?

- chaque membre de la classe doit correspondre à une colonne de la table : ainsi si une colonne est ajoutée à la table, ou un nouveau champs est ajouté à l'objet, le code doit être modifié en conséquence ;
- Correspondance des relations : il faut gérer les dépendances relationnelles entre les objets d'une part et les dépendances entre les tables d'autre part, définir les clés primaires, étrangères, etc.

Relation objet $\Longleftrightarrow$ SGBD : problématique (... suite)

- Gérer les types de données : supposons que j'ai une valeur de type booléen dans mon objet ; comment faire si la table d'accueil ne possède ce type de données ; je suis obligé de gérer cela manuellement ;
- Gérer les changements d'états : si au niveau de l'objet une valeur change, par exemple la valeur de l'adresse de courriel, il faut effectuer manuellement une requête SQL au niveau de la table pour effectuer la mise à jour nécessaire.

Relation objet $\Longleftrightarrow$ SGBD : problèmes à résoudre

En somme :

1. Variation du code de persistance en fonction :
   - du type de support de persistance (BD relationnelle BD objet relationnelle, objet, fichiers, etc.) ;
   - des différentes implémentations des fournisseurs de SGBD ;
2. Difficulté à changer de support de persistance en cas d'imbrication du code de persistance et du code métier.

Relation objet $\Longleftrightarrow$ SGBD : solution

- Séparation des entrées sorties des classes métier entrées-sorties ;
- Utilisation d'un objet particulier - objet d'accès aux données (Data Access Model)- pour abstraire et encapsuler les accès aux données.

Data Access Model (DAO)

Le DAO est un motif (modèle) de conception (design pattern) – également connu sous le nom de Data Mapper.

Utilité des DAO :

- faciliter la modification du modèle de base de données ;
- factoriser le code d'accès aux données ;
- faciliter l'optimisation des accès à la base en les regroupant au sein d'objets particuliers.

Le modèle DAO est implanté dans la couche dite d'accès aux données.

Data Access Model (DAO)

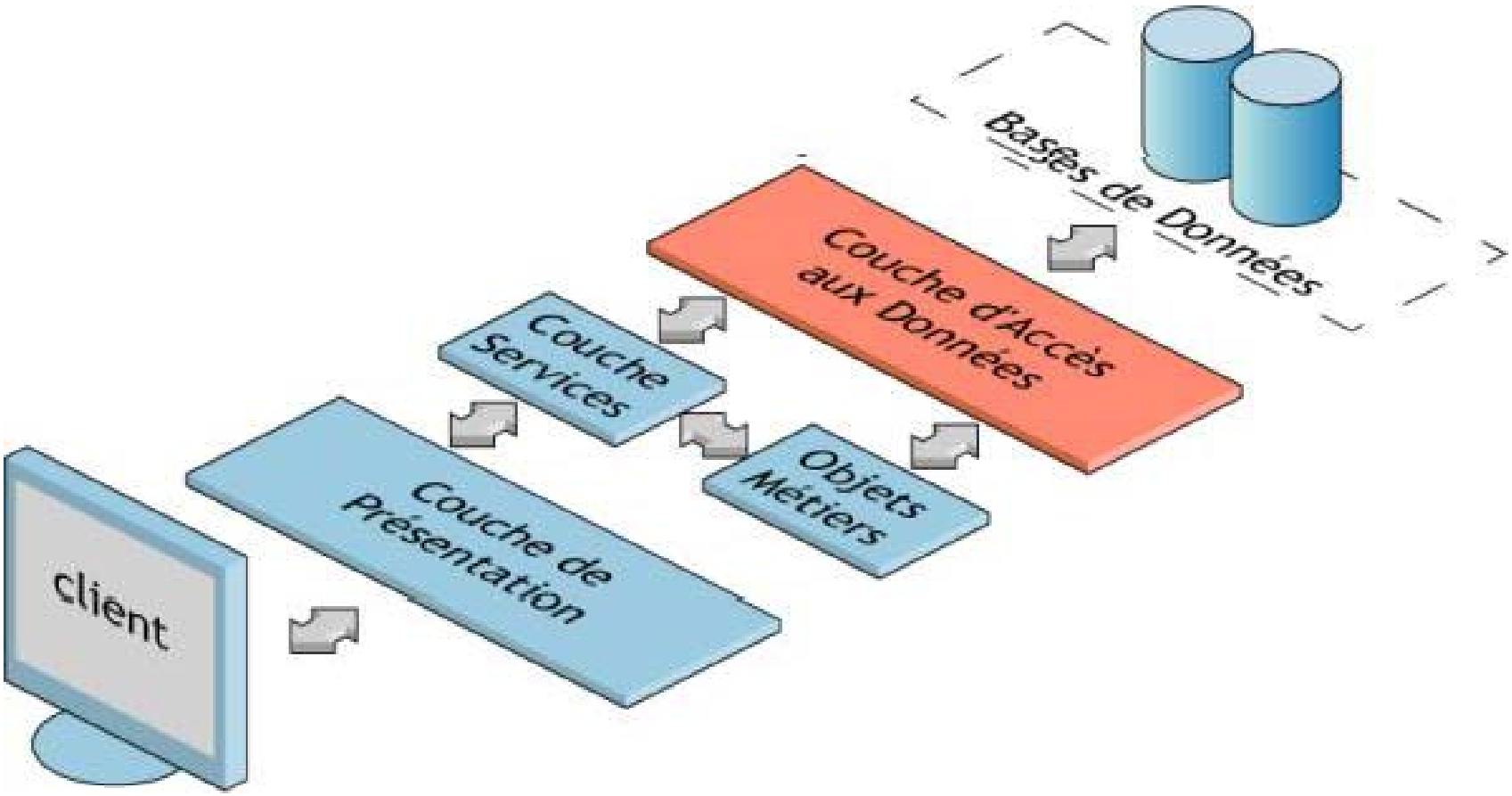


FIGURE: Modèle d'accès aux données

Data Access Model (DAO)

FIGURE: Modèle d'accès aux données simplifié

Exemple d'utilisation des DAO

```
Patient patient = PatientDAO.createPatient();

//Propriétés du patient
patient.setNom("GUINKO Ferdinand");
patient.setDateAdmission(new Date(0));

//Enregistrement des informations du patient
PatientDAO.save(patient);
```

Hibernate

Hibernate :

- Outil ORM ou Cadre (Framework) de persistance libre (open source) gérant automatiquement la persistance des objets Java/J2EE vers base de données relationnelle ;
- Code SQL généré à l'exécution via des informations fournies dans un document de correspondance (mapping) XML ou des annotations ;
- Description à l'aide de méta-données de la transformation réversible entre un modèle relationnel et un modèle de classes.

Hibernate

Architecture d'hibernate :

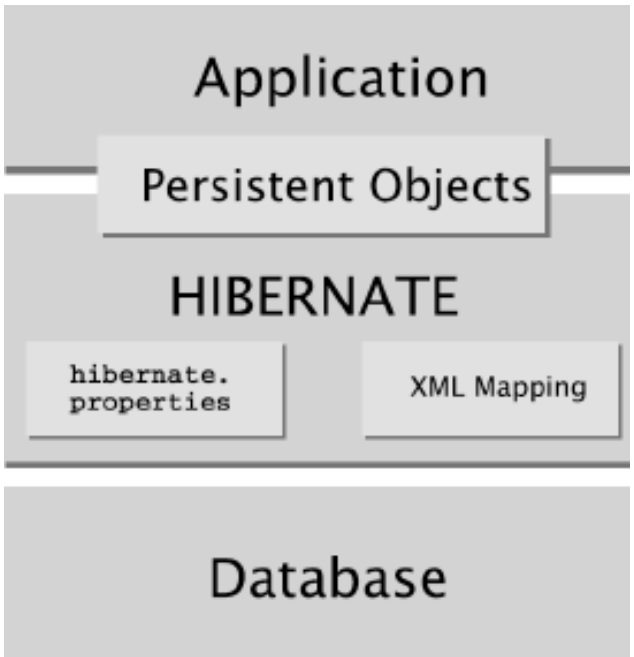


FIGURE: Illustration simplifiée

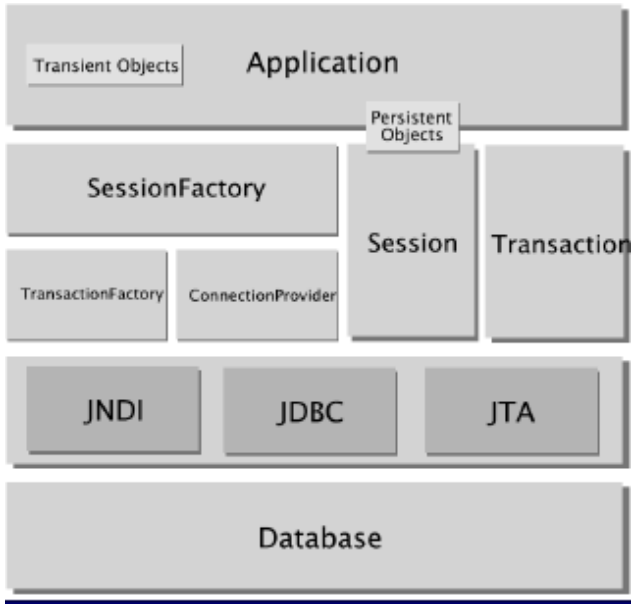


FIGURE: Illustration détaillée

Hibernate : configuration

Sa configuration se fait à travers 2 types de fichiers :

1. hibernate.properties ou hibernate.cfg.xml. Ces 2 fichiers ont la même utilité : lier Hibernate à notre BD 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.
2. Nom_Classe.hbm : Les fichiers de mapping, qui permettent de lier la BD avec les objets persistants. Ils respectent la syntaxe XML.

Hibernate : configuration : étape 1

Une fois que les librairies d'Hibernate sont incluses au projet, il faut :

- tout d'abord créer la BD qui servira au stockage des objets persistants. (Hibernate peut également le faire tout seul à partir de JavaBean ou de POJO) ;
- ensuite créer une table distincte 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 (cet attribut doit par ailleurs être défini en auto_increment). Dans l'exemple suivant, id = NumeroEtudiant.

Exemple :

```
CREATE TABLE Etudiant (
Nom VARCHAR (30) NOT NULL ,
Prenoms VARCHAR (30) NOT NULL ,
Age INT (2) NOT NULL ,
NumeroEtudiant INT(10) NOT NULL AUTO_INCREMENT
)
```

Hibernate : configuration : étape 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 BD.

Hibernate : configuration : étape 1 (... suite)

Exemple : 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 : configuration : étape 2

Il s'agit ici de créer le fichier de configuration hibernate.cfg.xml , il contient les informations de connexion à la BD.

```
<?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 la BD
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property> //Driver à
                    utiliser pour se connecter au SGBD MySQL
    <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 BD
    // (existe par défaut pour chaque type de BD)
    <property name="hibernate.show_sql">false</property> // Hibernate affiche ou non toutes les requêtes
                     SQL passées à la BD
    <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 : configuration : étape 3

Il s'agit ici de créer le ou les fichiers de mapping permettant de relier un objet Java à une Table de la BD.

Exemple : 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 NumeroEtudiant est différent des autres champs */
        <id name="NumeroEtudiant" type="integer" column="NumeroEtudiant"><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="Prenoms" column="Prenoms" type="string" not-null="true" length="10"/>
        <property name="Age" column="age" type="integer" not-null="true" length="10"/>
    </class>
</hibernate-mapping>
```

Hibernate : configuration : étape 4

A ce niveau, le mode de fonctionnement d'hibernate est complètement initialisé. Il ne reste plus qu'à s'en servir.

- Au démarrage, Hibernate crée un objet SessionFactory . Une SessionFactory peut ouvrir de 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 : configuration : étape 4 (... suite)

Création de la classe HibernateUtil

```
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 : configuration : étape 5 (... suite)

Il est maintenant possible ajouter des objets Etudiant dans la BD ainsi qu'il suit :

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
Etudiant e = new Etudiant(); // On crée l'objet étudiant
e.setAge(21);
e.setNom("GUINKO");
e.setPrenom("Ferdinand");
session.save(e); // On demande à Hibernate de le sauvegarder dans la BD
session.getTransaction().commit(); // On fait un commit
HibernateUtil.getSessionFactory().close();
```

Hibernate : configuration : étape 5 (... suite)

Il est aussi possible de récupérer des objets de la BD :

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

Remarque : Le langage utilisé pour faire des requêtes à la BD est propriétaire à Hibernate et se nomme HQL . Hibernate générera le SQL approprié et l'enverra à la BD.

Pour aller plus loin

1. http ://www.hibernate.org/hib_docs/v3/reference/fr-FR/html/
2. http ://www.hibernate.org/