Hibernate Vs JDBC

Aujourd'hui c'est match ! Nous allons opposer *JDBC* à *Hibernate / JPA*, deux façon d'accéder aux données dans les application java et les couches *DAO - data access object*.

Ce sujet est une discussion très fréquente lors du lancement d'un nouveau projet.

Hibernate

Hibernate est un ORM - Object Relational Mapping, et un ORM est une interface entre une application et une base de données, et plus exactement entre des objets et des tables.

JPA est la normalisation des ORM dans le monde Java : Java Persistece API.

JDBC

JDBC est le mécanisme qui permet à Java de lancer des requêtes SQL sur des bases de données. C'est l'équivalent de ODBC dans le monde Microsoft.

Hibernate utilise JDBC pour dialoguer avec les bases de données.

Les avantages d'hibernate par rapport à JDBC

Configuration over code

Il est très simple de définir un objet qui sera mappé à une table de base de donnée avec les annotations ***@Entity*** et ***@Id*** :

```
@Entity
public class Article {

  @Id
  private long id;

  private String code;

  + Getters et Setters
}
```

L'objet **Article** est mappé avec une table Article qui contient les colonnes ***id*** et ***code***. Par convention le nom de l'objet correspond au nom de la table, et le nom des attributs au nom des colonnes de la table.

Il est obligatoire d'avoir une colonne idantifiant (ou un idantifiant clé composé de plusieurs colonnes).

Ensuite Hibernate permet d'enregistrer l'objet dans la base, de le modifier, de le supprimer sans devoir écrire une seule ligne de code SQL. Par exemple pour récupérer l'article avec id 3 :

```
Article a = articleFactory.get(3);
```

En pur JDBC il faudrait écrire 3 fonctions SQL, et récupérer les données, et les assigner à un objet java.

Si on ajoute une colonne, en Hibernate il suffit de l'ajouter dans la classe. En JDBC il faudra modifier toutes les fonctions.

Portabilité du code

Hibernate utilise un système de proxy pour dialoguer (ou dialect) avec les bases de données, les requêtes sont codées dans le language "HQL" et c'est le dialect Hibernate qui se charge de traduire les requêtes SQL en HQL.

On peut donc passer d'une base de données à une autre simplement en modifiant la configuration Hibernate. Ou alors créer une application qui gère plusieurs bases de données.

Voilà les "dialect" proposés par Hibernate :

Base de donnée	**Dialect**
DB2	org.hibernate.dialect.DB2Dialect
DB2 AS/400	org.hibernate.dialect.DB2400Dialect
DB2 OS390	org.hibernate.dialect.DB2390Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL	org.hibernate.dialect.MySQLDialect
MySQL with InnoDB	org.hibernate.dialect.MySQLInnoDBDialect

Base de donnée	Dialect
MySQL with MyISAM	org.hibernate.dialect.MySQLMyISAMDialect
Oracle (toutes versions)	org.hibernate.dialect.OracleDialect
Oracle 9i	org.hibernate.dialect.Oracle9iDialect
Oracle 10g	org.hibernate.dialect.Oracle10gDialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Ingres	org.hibernate.dialect.IngresDialect
Progress	org.hibernate.dialect.ProgressDialect
Mckoi SQL	org.hibernate.dialect.MckoiDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Pointbase	org.hibernate.dialect.PointbaseDialect
FrontBase	org.hibernate.dialect.FrontbaseDialect
Firebird	org.hibernate.dialect.FirebirdDialect

Et de plus il est assez simple d'implémenter un nouveau Dialect.

Jointures / Objets liés

Hibernate permet de facilement faire des requête sur plusieurs objets ou tables liées. Exemple :

```
"Select article where article.fournisseur.departement = 67"
```

En JDBC il faudrait faire une requête avec trois jointures ...

Pourquoi JDBC ?

Hibernate ne semble proposer que des avantages par rapport à JDBC.

JDBC est plus simple à apprendre et à mettre en œuvre (bien que des framework comme Spring Boot permettent d'utiliser Hibernate avec un minimum de configuration).

On pourrait aussi dire que JDBC est plus adapté pour les applications de petites tailles qui n'ont que quelques tables.

Les performances

Il faut bien comprendre le fonctionnement d'Hibernate afin d'éviter de se retrouver avec une application très lente.

En utilisant Hibernate, le développeur n'a aucune idée des requêtes générées vers la base de données. Prenons le cas d'un objet 'Client' qui aurait 60 colonnes, et on souhaite afficher une liste des clients uniquement les noms en prénom. La table contient 1 000 clients.

En SQL on ferait :

```
select nom, prenom from Client;
```

En HQL :

```
select client from Client;
```

Ce qui sera être traduit en SQL par :

```
select * from Client;
```

Et qui va instancier 1000 objets Clients avec les 60 colonnes alors qu'on ne souhaite afficher que les nom et prénom des clients. La requête SQL sera énorme par rapport, et l'impact sur la mémoire également car on chargera des données qui ne sont pas nécessaires sur le moment.

Prenons maintenant le cas d'un objet contact relié à un objet société. On veut afficher les noms et prénoms des contacts et la raison sociale de leur entreprise. En HQL :

```
select contact from Contact;
```

Ce qui nous retournera en Java une liste des contacts avec un lien dans chaque contact vers leur société.

Hibernate va générer une requête jointe et complète entre les Contacts et les sociétés, alors que ne voudra utiliser que certaines colonnes des deux objets.

Encore une fois, la requête générée sera disproportionnée par rapport au besoin.

Résultat du match

Pour moi victoire de Hibernate, mais ...

Ma conclusion est que Hibernate est vraiment génial pour effectuer des opérations de gestion sur les objets : création, modification, lecture d'un enregistrement, suppression. Et permet, même sur des petites applications, de développement très rapidement des fonctionnalités simples.

Dès que l'on souhaite récupérer un ensemble de données, il faut faire attention à l'impact sur les performances et dans certains cas préférer une requête ciblée en SQL / JDBC qui sera beaucoup plus efficace.

L'autre solution est de mettre en place un système de cache qui va conserver en mémoire une copie des objets stockés en base de données, et qui évitera des répéter sans arrêt les mêmes requêtes.

Je conseille d'utiliser Hibernate et d'activer l'option ***show_sql*** qui affichera dans un fichier de log les requêtes générées, et donc le développeur pourra constater le nombre de requêtes ainsi que leur complexité, ... et en cas de lenteur décider soit de mettre les données en cache, soit d'utiliser une requête JDBC / SQL.