Hibernate : ORM solution
(In Action, C. bauer & G. King Ed. Manning)

Fabrice Jouanot

Interrogation avancée

Exécution d'une requête

- L'interface standard est HQL via createQuery()

```
Query hqlQuery = session.createQuery("from User");
```

- équivalente à la requête SQL:

```
Query sqlQuery = session.createSQLQuery (
    "select {u.*} from USERS {u}", "u", User.class) ;
```

- qui peut s'écrire sous forme de critère

```
Criteria crit = session.createCriteria(User.class);
```

Exploitation des résultats

- Pagination des résultats:

```
Query query = session.createQuery("from User u order by u.name asc");
query.setFirstResult(0);
query.setMaxResults(10);
```

- Récupération d'une liste

```
List result = query.list();
```

- Récupération d'un élément

```
Bid maxBid = (Bid) session.createQuery("from Bid b order by b.amount desc")
    .setMaxResults(1).uniqueResult();

Bid bid = (Bid) session.createCriteria(Bid.class)
    .add(Expression.eq("id",id).uniqueResult();
```

Paramétrage des requêtes

- On évitera de construire une règle sous la forme d'une String prêt à l'emploi : on utilise l'instanciation de paramètres
  - en nommant les paramètres

```
String query="from Item item where item.description like :searchString;
List result = session.createQuery(query)
    .setString("searchString",searchString).list();
ou  .setParameter("searchString",searchString,Hibernate.STRING).list();
```

  - en positionnant les paramètres

```
String query = "from Item item where item.description like ? and item.date > ?";
List result = session.createQuery(query).setString(0,searchString).setDate(1,minDate)
    .list();
```

Requête de base

from Bid ⇔ Session.createCriteria(Bid.class)
est traduit en
select B.BID_ID, B.AMOUNT, B.ITEM_ID, B.CREATED from BID B

- Utilisation d'Alias

from Bid as bid (ou Bid bid)

- requête Polymorphique

from BillingDetails
et pour avoir les objets concrets des sous-classes :
from CreditCard

- Filtrage : via la clause WHERE
  - opérateur de comparaison classique
  - opérateur arithmétique
  - String matching LIKE
  - opérateur logique
  - opérateur IS NULL

from User u where u.email is not null
⇔
session.createCriteria(User.class).add(Expression.isNotNull("email")).list();

Jointure

- 4 manières d'exprimer des jointures
  - jointure ordinaire dans la clause from
  - Fetch jointure
  - Theta-style
  - implicite
- Jointure en mode Fetch (jointure externe optimisée)
  - mode HQL

```
from Item item
left join fetch item.bids where item.description like "%gc%"
```

  - mode critère

```
session.createCrieria(Item.class).setFetchMode("bids",FetchMode.EAGER)
    .add(Expression.like("description","gc",MatchMode.ANYWHERE)).list();
```

  - traduction SQL

```
select I.DESCRIPTION, I.CREATED, I.SUCCESSFUL_BID, B.BID_ID,
    B.AMOUNT,B.ITEM_ID,B.CREATED
from ITEM I left outer join BID B on I.ITEM_ID=B.ITEM_ID
where I.DESCRIPTION like '%gc%'
```

Jointure ordinaire

- Une jointure classique utilise l'opérateur join et les Alias:

```
from Item item join item.bids bid
where item.description like '%gc%' and bid.amount>100
```

- A la différence du Fetch mode, l'association n'est pas reformatée : le résultat est un tableau de paires (Item, Bid)

```
Query q= session.createQuery("from Item item join item.bids bid");
Iterator pairs=q.list().iterator();
while (pairs.hasNext() ) {
    Object[] pair=(Object[]) pairs.next();
    Item item=(Item)pair[0]; Bid bid=(Bid)pair[1];
}
```

Jointure implicite

- Permet une interrogation à la manière OQL (BDOO)
  - pour interroger des objets composants
  - pour naviguer dans des associations implicites

```
from User u  where u.address.city = 'Grenoble'
⇔
session.creatCriteria(User.class)
    .add(Expression.eq("address.city","Grenoble"));

from Bid bid where bid.item.category.name like 'Laptop%'
```

Jointure Theta-Style

- Utile pour réaliser une jointure avec un critère ne faisant pas intervenir des attributs liés par une association

```
from User user, LogRecord log
where user.username = log.username
```

qui s'utilise via un Iterator

```
Iterator i= session.createQuery("from User user, LogRecord log
where user.username = log.username").list().iterator();
while (i.hasNext() ) {
    Object[] pair=(Object[]) i.next();
    User user=(User)pair[0]; LogRecord log=(LogRecord)pair[1];
}
```

Agrégation

- Hibernate permet d'écrire via HQL des projections, des Group By Having, des fonctions d'agrégations, la suppression de doublon (distinct)

```
Select p.LASTNAME, count(a)
      From Person p join Addresse a
      group by p.LASTNAME
      having count(a)>10
```

Notion de sous-requêtes

- Hibernate est l'un des rares ORM a proposé des sous requêtes dans les clauses FROM et WHERE

```
from User u where 10<(
    select count(i) from u.items i where i.successfulBid is not null )

from Bid bid where bid.amount +1 >= (
    select Max(b.amount) from Bid b )
```

- Opérateurs ANY, ALL, SOME, IN disponibles.

Gestion des transactions

Définition d'une transaction

- Une transaction est définie entre les appels beginTransaction() et commit()

```
Session session=sessions.OpenSession();
Transaction tx=null;
try {
      tx=session.beginTransaction();
      concludeAuction();
      tx.commit();
} catch (Exception e) {
      if (tx!=null) {
            try { tx.rollback(); }
            { catch (HibernateException he {…}
      } throw e;
} finally {
      try { session.close(); }
      catch (HibernateException he) { throw he;}
}
```

Niveau d'isolation

- Par défaut celui de JDBC (soit read committed soit repeatable read)
- Option de configuration (donc pour tout le pool de connections):

  Hibernate.connection.isolation = x

- Où x peut être :
  - 1 - Read uncommitted
  - 2 - Read commited
  - 3 - Repeatable read
  - 8 - Serializable