

Sommaire

Contribution - Comment parcourir de gros volume de données en 2 lignes de code ? 2

Contribution - Comment parcourir de gros volume de données en 2 lignes de code ?

publié par Jérôme Guibert le mardi 10 mars 2009 - 17:02

2 lignes de code pour accéder et manipuler de larges volumes de données en toute simplicité en utilisant [Spring-framework](#) et [Hibernate](#).

Voilà ce que je vous propose via le Jdk 5. Ce sera ma toute première et très humble contribution à la communauté Open Source qui m'a rendu d'immenses services tout au long de mon parcours professionnel.

2 lignes de code pour accéder et manipuler de larges volumes de données en toute simplicité en utilisant [Spring-framework](#) et [Hibernate](#).

Voilà ce que je vous propose via le Jdk 5. Ce sera ma toute première et très humble contribution à la communauté Open Source qui m'a rendu d'immense service tout au long de mon parcours professionnel.

Lorsque nous ne pouvons rien supposer du volume ou du nombre de résultat qu'une requête va nous retourner (hormis le seul fait que la mémoire vive ne sera pas du tout suffisante...), nous avons deux principales techniques pour faire en sorte que tout ce passe bien :

- utiliser un chargement à la demande des différents objets, ce qui nécessite d'avoir une session ouverte pour toute la durée de traitement,
- utiliser un mécanisme de pagination en suivant le pattern DAO sans garder de session.

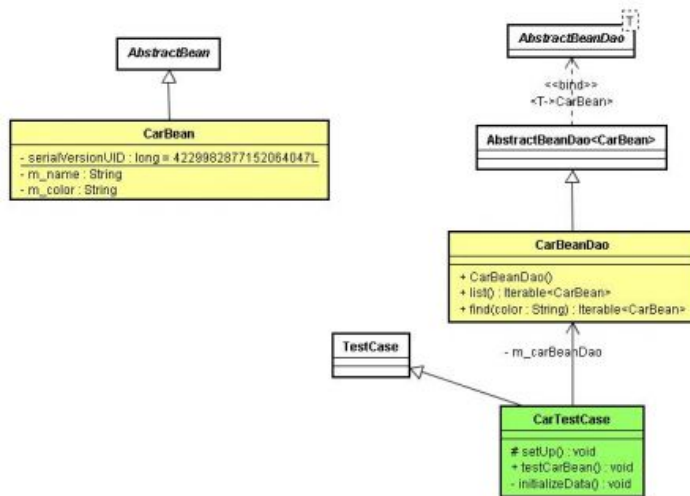
C'est cette dernière technique que je vais utiliser ici. Cependant, un mécanisme de pagination classique a plusieurs inconvénients majeurs car:

- il faut gérer la notion de page courante,
- la navigation au sein des données a de gros impacts sur le code: il est tout de suite moins propre, de compilation et ce mécanisme, de rentre transparents la gestion de ces index et des différents appels à la source de données. 2 lignes de codes cotés clients, voilà ce que je vous propose de réaliser.

La théorie par la pratique imaginez:

- un simple bean `CarBean` qui possède deux attributs `color` et `name`,
- une DAO `CarBeanDao` qui permet les opérations classiques d'enregistrement/suppression et deux méthodes `list` pour parcourir l'ensemble des voitures, et `find` pour lister uniquement celle qui possède un attribut spécifique.

Voici le chargement de données possible à partir de milliers d'entité `CarBean`

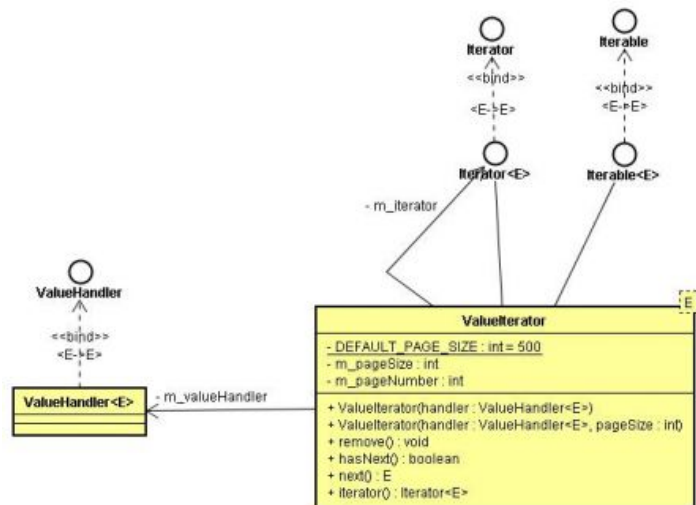


Pour accéder à 10 ou 1000000 entités CarBean, le code client sera toujours semblable à cela:

```
34@ public void testCarBean() {
35
36     initializeData();
37
38     System.err.println("*****");
39     System.err.println("Find all car ");
40     System.err.println("*****");
41
42     for(CarBean car: m_carBeanDao.list())
43         System.err.println(car.getName());
44
45
46     System.err.println("*****");
47     System.err.println("Find all car with a specified color");
48     System.err.println("*****");
49
50     for(CarBean car: m_carBeanDao.find("blue"))
51         System.err.println(car.getName());
52
53 }
54
55@ private void initializeData() {
56     for (int i = 0; i <= 22; i++)
57         m_carBeanDao.save(new CarBean("car " + i, "blue"));
58     for (int i = 0; i <= 23; i++)
59         m_carBeanDao.save(new CarBean("car " + i, "red"));
60 }
```

Personnellement, je n'ai pas encore trouvé plus simple qu'un foreach...

Tout le mécanisme est réalisé au niveau de la classe CarBeanDao en utilisant deux objets ValueIterator et ValueHandler dont voici un diagramme de classe:



ValueHandler est une interface qui définit une méthode qui va interroger la base de donnée à la demande.

```
1 package org.intelligentsia.utility.dao;
2
3 import java.util.List;
4
5 /**
6  * ValueHandler<E> interface define a method to feed paged result of type E.
7  *
8  * @author <a href="mailto:jguibert@intelligents-ia.com">Jerome Guibert</a>
9  * @version 1.0.0
10 */
11 public interface ValueHandler<E> {
12     /**
13      * Feed a result list by executing a request against a source.
14      *
15      * @param pageSize
16      *         page size to use
17      * @param pageNumber
18      *         page number index begin at 0
19      * @return Returns a list of object or an empty list.
20      */
21     public List<E> feed(final int pageSize, final int pageNumber);
22 }
```

ValueIterator est une sorte d'itérateur qui va gérer pour vous toute la problématique liée à la pagination, dont voici les deux principales méthodes:

```
75@  /**
76   * Returns true if we have another next element and feed if necessary.
77   * internal list of value (make a call to @see
78   * {@link ValueHandler#feed(int, int)}).
79   *
80   * @see java.util.Iterator#hasNext()
81   */
82@  public boolean hasNext() {
83      boolean result = (m_iterator != null) ? m_iterator.hasNext() : false;
84      if (!result) {
85          m_iterator = m_valueHandler.feed(m_pageSize, m_pageNumber).iterator();
86          m_pageNumber++;
87          result = (m_iterator != null) ? m_iterator.hasNext() : false;
88      }
89      return result;
90  }
91
92@  /**
93   * Returns next element.
94   *
95   * @see java.util.Iterator#next()
96   */
97@  public E next() {
98      return m_iterator.next();
99  }
100
```

L'implémentation au niveau de la DAO consiste à écrire un code java équivalent à ceci:

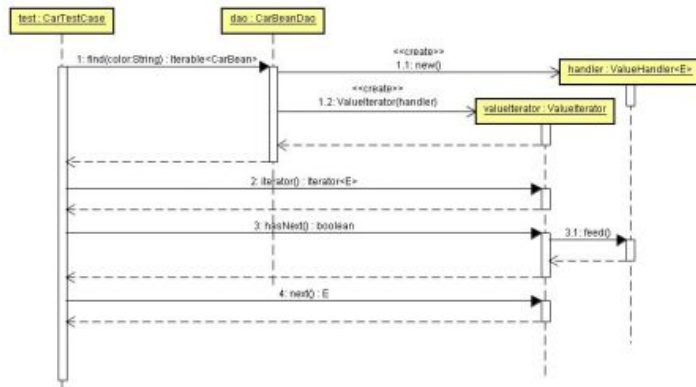
```
/**
 * Return a list of car which has the specified color. Under laying pagination
 * mechanism is used with a default page size set to 10 entity.
 */
public Iterable<CarBean> find(final String color) {
    return new ValueIterator<CarBean>(new ValueHandler<CarBean>() {
        @SuppressWarnings("unchecked")
        public List feed(final int pageSize, final int pageNumber) {
            return (List) getHibernateTemplate().execute(new HibernateCallback() {
                public Object doInHibernate(Session session) throws HibernateException, SQLException {
                    Criteria criteria = session.createCriteria(getEntityClass());

                    if (color != null && !"".equals(color))
                        criteria.add(Restrictions.eq("m_color", color));

                    criteria.addOrder(Order.asc("m_identity"));
                    criteria.setFirstResult(pageSize * pageNumber);
                    criteria.setMaxResults(pageSize);
                    return criteria.list();
                }
            });
        }
    }, 10);
}
```

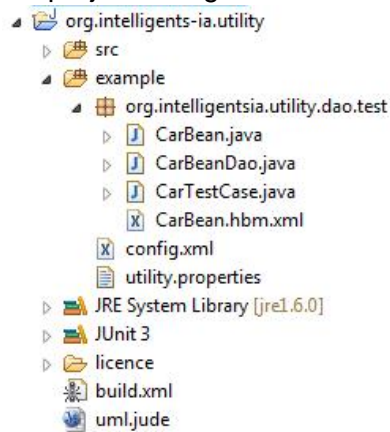
Vous vous occupez uniquement de créer votre requête de façon normale, sans les soucis de pagination...

Pour les curieux, voici un diagramme de séquence qui explicite de façon plus détaillée ce qui se passe lors de l'appel à la méthode `find` :



Le projet. Je vous ai rassemblé toutes les sources nécessaires (les classes `ValueXXXX`, les classes d'exemple et le test unitaire) dans un projet eclipse.

Ce projet est organisé comme suit:



Pour le tester vous devez mettre à jour le fichier `utility.properties` qui contient la chaîne de connexion à la base de données, et la classe `path` de façon à référencer les projets Spring, hibernate...

Si vous rencontrez des problèmes, ou avez des suggestions n'hésitez pas :-)