Spring JDBC : une alternative pour Hibernate dans un contexte de développement d'application Web transactionnelle en Spring contenant du Traitement en Lot

par Joël Grira

Date de publication : 2007-09-26

Dernière mise à jour :

Cet article répresente une discussion de la performance du framework Hibernate dans le cas où il est utilisé dans une application web contenant du traitement en lots. Une alternative est présentée.

I - Abstract
    I-A - Introduction
    I-B - Architecture Commune
    I-C - Fonctionnement
    I-D - Conclusion
II - Les sources de cet article

I - Abstract

La performance est un enjeu de taille pour les applications web transactionnelles professionnelles. Plusieurs facteurs ont un impact direct sur la performance de telles applications, en l'occurrence les choix de la technologie, de l'architecture, de l'infrastructure matérielle ou logicielle #etc. La décision d'opter pour un choix plutôt qu'un autre est conditionnée, entre autres choses, par la nature des traitements exigés : ainsi, omettre de prendre en considération les détails fonctionnels affectant la nature des traitements pourrait fort probablement conduire à la production d'applications non performantes. Cet article présente la différence de performance entre deux applications web transactionnelles contenant du traitement en lots (batch processing). Chacune de ces deux applications contient une couche de persistance faisant intervenir une technologie différente. Des mesures de performances sont effectuées afin de mettre en évidence la technologie la plus appropriée à la nature du traitement.

Keywords : Performance, ORM, Struts, Spring, Hibernate, Jdbc.

I-A - Introduction

La notion de performance est devenue le centre d'intérêt de nombreuses équipes de développement d'applications web transactionnelles. Réaliser une application web transactionnelle performante n'est pas sans difficulté surtout quand on constate la large gamme de technologies disponibles. Le défi consiste à déterminer les technologies qui s'apparentent le mieux avec les besoins et d'en faire un bon usage.

Le processus de développement d'applications web transactionnelles est, de nos jours, facilité par divers outils (Les environnements de développement intégrés comme "Eclipse" et "Zend") et frameworks (Les frameworks de persistance comme "Hibernate", de développement Web comme "Struts" et "Spring"). Cet article vise à mettre en évidence la différence de performance de deux applications web transactionnelles développées en utilisant les frameworks " **Struts** " et " **Spring** " ainsi que l'IDE (Environnement de Développement Intégré) " Eclipse ".

La première application comporte une couche de persistance basée sur le principe d'ORM (Object Relational Mapping) via le framework **Hibernate** alors que la deuxième application comporte une couche de persistance utilisant la couche JDBC (Java Database Connectivity) offerte par Spring [voir **la documentation de Spring**].

I-B - Architecture Commune

Les deux applications développées respectent le patron de conception MVC (Modèle-Vue-Contrôleur). Elles sont composées de plusieurs couches telles qu'illustré par la Figure 1 - Architecture des deux applications.

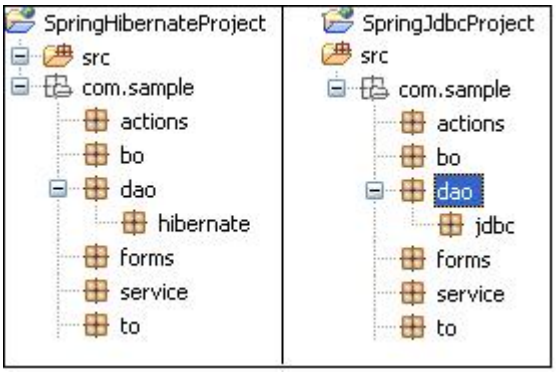

Figure 1 - Architecture des deux applications

Légende

- actions : classes d'actions Struts.
- bo (business Object): objets d'affaires.
- dao (Data Access Object) : objets d'accès aux données.
- forms : classes des formulaires Struts.
- service : classes des services Spring.
- to (Transfert Object): objets de transfert.

La seule différence entre les deux applications réside au niveau des packages " dao.hibernate " et " dao.jdbc " qui contiennent deux implémentations différentes de la couche d'accès aux données et ce, en utilisant, respectivement, " Hibernate " et " springJdbc ".

Les deux applications utilisent la même source de données (dataSource) comme le montre la Figure 2 - Configuration du DataSource mysql. Néanmoins, l'application " SpringHibernateProject " injecte cette source de données en tant que propriété d'un SessionFactory de Hibernate (voir le bean dont le id="hibernateSessionFactory" dans le fichier " applicationContext.xml "). C'est exactement à cet emplacement que le principe d'ORM se met en application : en effet, le mapping de la table " client " est effectué via le fichier " Client.hbm.xml " qui est référencé dans le fichier " applicationContext.xml ".

```
<bean id="sampleDataSource"
      class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close">
      <property name="driverClassName">
            <value>com.mysql.jdbc.Driver</value>
      </property>
      <property name="url">
            <value>jdbc:mysql://127.0.0.1:3306/sample</value>
      </property>
      <property name="username">
            <value>root</value>
      </property>
      <property name="password">
            <value>root</value>
      </property>
</bean>
```

Figure 2 - Configuration du DataSource mysql

I-C - Fonctionnement

Les deux applications sont fonctionnellement identiques: elles affichent un formulaire dans lequel on peut saisir des informations concernant un client. Les deux applications tenteront d'enregistrer ces mêmes informations plusieurs fois. Les deux applications comportent les mêmes actions Struts telles que déclarées dans le fichier " Struts-config.xml " qui apparaît à la Figure 3 - Configuration des actions Struts:

```
<action path="/loginForm"
        type="com.sample.actions.ClientFormAction"
        parameter="clientForm"
        name="clientForm"
        scope="request">
        <forward name="success" path="/WEB-INF/clientForm.jsp" />
</action>

<action path="/inscriptionAction"
        type="com.sample.actions.ClientAjouterModifierAction"
        parameter="clientForm"
        name="clientForm"
        scope="request">
        <forward  name="fail" path="/loginForm.do" />
        <forward  name="success" path="/WEB-INF/success.jsp" />
</action>
```

Figure 3 - Configuration des actions Struts

La première action est responsable de l'affichage d'un formulaire de saisie. La seconde action traite les données saisies au niveau de ce formulaire.

Afin de simuler un traitement en lot, on itère une centaine de fois sur une instruction I/O : ainsi, on récupère une référence vers le service de l'entité " client " et on invoque la méthode " saveOrUpdate() " de ce service tout en lui passant en paramètre l'entité " client " précédente.

La différence entre les performances notées pour chaque application lors de l'exécution de l'action " inscriptionAction " montre que la vitesse d'exécution est plus élevée pour l'application configurée avec " Hibernate " par rapport à celle configurée avec " springJdbc ". En effet, l'application affiche à la console le temps avant et après le traitement itératif. En moyenne, il s'agit d'une différence de 10% de différence de performance tel que le montre la figure ci-contre :

	Spring Jdbc	Hibernate
T1	13:19:28 - 13:19:35	11:54:52 - 11:55:00
T2	13:19:38 - 13:19:44	11:56:04 - 11:56:11
T3	13:19:46 - 13:19:54	11:56:22 - 11:56:29
T4	13:19:56 - 13:20:02	11:56:42 - 11:56:49
T5	13:20:04 - 13:20:12	11:57:08 - 11:57:15
T6	13:20:14 - 13:20:19	11:57:18 - 11:57:25
T7	13:20:47 - 13:20:54	11:57:26 - 11:57:34
T8	13:20:56 - 13:21:02	11:57:35 - 11:57:43
$\frac{\sum T_i}{i}$	53/8 = 6,625 secondes	59/8 = 7,375 secondes

Figure 4 - Mesures de performance

Il est important de noter que les données quantitatives recueillies suites aux essais de performance sont dépendantes de plusieurs facteurs tels que l'infrastructure matérielle et logicielle. Ainsi, il est possible que l'exécution des mêmes applications dans d'autres environnements donne d'autres mesures. Dans cet article, les essais ont été effectués

sur le même poste : ainsi, toutes les variables desquelles la performance est dépendante sont égales et n'ont aucun impact dans le cadre des tests puisqu'elles ont demeuré constantes

Dans cet article, les applications développées se connectent à une base de données qui ne contient qu'une seule table : ainsi, le mapping des relations " many-to-one " (Le mapping " many-to-one " et " many-to-many " font partie des bases du mapping O/R) n'a pas été étudié. De plus, plusieurs notions telles que les " associations bidirectionnelles " [voir la **documentation de hibernate**] ainsi que le " lazy loading " [voir **cet article**] #etc., n'ont pas été abordées dans cet article pour la simplification.

I-D - Conclusion

Dans cet article, on a mis en évidence que la performance d'une application web transactionnelle développée avec springJdbc est relativement supérieure à une autre développée avec une couche de persistance O/R telle que Hibernate. Les frameworks d'ORM sont certainement avantageux dans la mesure où ils fournissent des interfaces obéissant au paradigme orienté objet et par conséquent, facilitent le processus du développement. Néanmoins, on constate que les applications y recourant sont pénalisées au niveau de la performance.

II - Les sources de cet article

- **springjdbc.zip**
- **springhibernate.zip**