

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

Joël Grira – joel.grira.1@gmail.com

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 ^[1]. 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.

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 ^[2] et frameworks ^[3]. Cet article vise à mettre en évidence la différence de performance de deux applications web transactionnelles développées en utilisant les frameworks « Struts » [I] et « Spring » [II] ainsi que l'IDE ^[4] « Eclipse ».

¹ Traitement en lot : *batch processing en anglais*.

² Les environnements de développement intégrés (IDE) comme « Eclipse » et « Zend ».

³ Les frameworks de persistance comme « Hibernate », de développement Web comme « Struts » et « Spring ».

⁴ IDE : *Environnement de Développement Intégré*

La première application comporte une couche de persistance basée sur le principe d'ORM ^[5] via le framework Hibernate [III] alors que la deuxième application comporte une couche de persistance utilisant la couche JDBC ^[6] offerte par Spring [IV].

Architecture Commune

Les deux applications développées respectent le patron de conception MVC ^[7]. Elles sont composées de plusieurs couches telles qu'illustré par la *Figure 1 - Architecture des deux applications*.

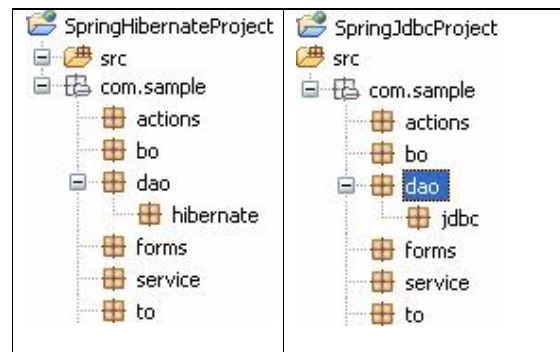


Figure 1 - Architecture des deux applications

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 de des packages « dao.hibernate » et « dao.jdbc » qui contiennent deux implémentations différentes en utilisant, respectivement, « Hibernate » et « spring.Jdbc ».

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

⁵ ORM : *Object Relational Mapping*.

⁶ JDBC : *Java Database Connectivity*.

⁷ MVC : *Modèle-Vue-Contrôleur*.

« *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

Fonctionnement

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:

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.

```
<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

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
$\sum_i T_i$	53/8 = 6,625	59/8 = 7,375

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 ne 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 Mysql qui ne contient qu'une seule table : ainsi, le mapping des relations « *many-to-one* »^[8] n'a pas été étudié. De plus, plusieurs notions telles que les « *associations bidirectionnelles* » [V] ainsi que le « *lazy loading* » [VI] ...etc., n'ont pas été abordées dans cet article pour la simplification. Même si on a testé la différence de performance entre les deux applications en n'utilisant qu'une seule table, les résultats ont été concluants en montrant que la couche Hibernate ralentit l'application même si la différence n'est pas énorme. Même en doublant le volume des données traitées, en changeant le nombre d'itérations de 100 à 200 sur l'instruction I/O, on a remarqué, comme le montre la Figure 5, que le rapport de performance entre les deux applications reste pratiquement le même.

	Spring Jdbc	Hibernate
T1	10:03:14 - 10:03:25	09:45:29 - 09:45:41
T2	10:03:25 - 10:03:37	09:45:41 - 09:45:53
T3	10:03:37 - 10:03:49	09:45:53 - 09:46:05
T4	10:03:49 - 10:04:00	09:46:05 - 09:46:18
T5	10:04:00 - 10:04:11	09:46:18 - 09:46:32
T6	10:04:11 - 10:04:22	09:46:32 - 09:46:44
T7	10:04:22 - 10:04:34	09:46:44 - 09:46:56
T8	10:04:34 - 10:04:45	09:46:56 - 09:47:09
	10:04:45 - 10:04:57	09:47:09 - 09:47:20
	10:04:57 - 10:05:08	09:47:20 - 09:47:33
$\frac{\sum T_i}{i}$	114/10 = 11,4	124/10 = 12,4

Figure 5 - Mesures de performance en ayant doublé le volume de données traitées

Néanmoins, il importe de signaler que, dans un contexte réel, l'application utilisant « springJdbc » comportera un mécanisme de construction dynamique de requête, chose que Hibernate peut s'en passer. En effet, la couche « DAO » aura besoin de connaître les noms des tables qui feront l'objet des requêtes, l'ensemble de colonnes concernées par la requête ainsi que les valeurs des paramètres à passer à la requête SQL. En revanche, la couche « DAO » de l'application utilisant « Hibernate » n'a besoin que des valeurs des

paramètres. Ainsi, il est fort possible que la performance de l'application « springJdbc » soit altérée par l'existence d'un mécanisme de construction dynamique de requêtes, tel que celui qui existe dans la méthode « *dao.jdbc.BaseJdbcDAO.update(T bo)* ».

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. Néanmoins, Hibernate permet d'éviter de se préoccuper des détails de la base données, chose que les applications utilisant JDBC doivent prendre en considération en se dotant d'un mécanisme de récupération des noms de tables, de leurs colonnes ...etc. ce qui pourrait coûter cher au niveau de la performance et ainsi, être désavantagé par rapport à Hibernate.

Références

- ^I Struts : <http://struts.apache.org/>
^{II} Spring : <http://www.springframework.org/>
^{III} Hibernate : <http://www.hibernate.org/>
^{IV} Spring JDBC : <http://www.springframework.org/docs/reference/new-in-2.html#new-in-2-middle-tier-jdbc>
^V Associations bidirectionnelles dans Hibernate : http://www.hibernate.org/hib_docs/v3/reference/en/html/tutorial.html#tutorial-associations-bidirectional
^{VI} Article «*Hibernate: Understanding Lazy Fetching*». <http://www.javalobby.org/java/forums/t20533.html>
 Aug 3, 2005, R.J. Lorimer.

⁸ Le mapping « many-to-one » et « many-to-many » font partie des bases du mapping O/R.