JDBC + JPA + Hibernate – Sans maitrise la puissance n'est rien

Date : 10 avril 2015
Format : Conférence
Speakers : Brice Leporini, indépendant
Blog : http://the-babel-tower.github.io

Brice commence par nous rappeler que la mise en œuvre de JPA est rapide :

1. Configuration XML
2. Annotations sur les entités
3. Hibernate dans le classpath

Hibernate et JPA sont fortement ancrées dans les applications de gestion. Mais ces technos sont rarement maitrisées. Et les problèmes de performance viennent souvent de là.

Depuis son IDE, Brice nous montre une succession d'exemples utilisant JPA avec Hibernate pour implémentation. Codés sous forme de tests unitaires, ces exemples et n'utilisent pas de transaction manager. A chaque, correspond une problématique et la manière d'y remédier.

Exemple 1 : occupation mémoire du cache de niveau 1

Insertion de 1000 User et Bibliography avec un Xmx à 64Mo, le tout dans une même transaction.
OOME de type Heap Space déclenchée.
Réponse dans la doc : il faut périodiquement se synchroniser et vider le cache de niveau 1.

Exemple 2 : problèmes de concurrence avec le cache de niveau 1

Le scénario est le suivant :

1. Création d'un User depuis un 1er contexte de persistance.

2. Mise à jour du nom de ce User dans un 2ième contexte de persistance.
3. Chargement du même User depuis le 1er contexte de persistance.

Dans les traces, on voit que la ligne est remontée dans la base de données mais que le nombre d'objets hydraté est de 0. Hibernate récupère l'entité dont l'ID est 1 depuis son cache de niveau 1.
=> Attention au cache obsolète entre les sessions, cas typiquement possible dans une transaction imbriquée.

Exemple 3 : utilisation du bach_size JDBC

Permet de réduire le nombre d'A/R avec la base de données.
Utilisation de JProfiler pour comprendre pourquoi le temps d'exécution est identique.
Dans son exemple, le batch_size ne semble pas fonctionner.
Une 1ière explication vient de la stratégie de génération de clés primaires qui est à AUTO => c'est la base qui donne l'identifiant. Avec AUTO, à chaque persist(client) il faut communiquer avec la base.
L'utilisation de séquences est censé corriger ce problème, mais les perfs ne sont toujours pas terribles.
En effet, par défaut, Hibernate insère les entités dans l'ordre qu'on lui a donné : il alterne entre User et Bibliography.
Le paramètre **hibernate.order_inserts** indique à Hibernate de réordonner les entités avant insertion. On passe alors de 18 à 8 secondes.

Exemple 4 : requêtage OneToOne

Exemple d'insertion d'un User et d'une Biography.
Un SqlCounter est placé sur la datasource ds.addListener.
Au lieu d'avoir 1 seule requête SQL comme on pourrait s'y attendre, 2 requêtes SQL sont exécutées.
1ière tentative de correction : @OneToOne passé en lazy. Toujours 2 requêtes avec le même comportement.
La spéc JPA précise que l'utilisation de LAZY est un hint alors que le EAGER est une contrainte. Pourquoi Hibernate ne suit pas cette recommandation ? Et bien parce que l'attribut *mappedBy* est dans le User. C'est donc Bibliography qui est propriétaire de la relation.
Solution : changer le propriétaire + ne pas rendre bidirectionnelle la relation.
Java ne peut pas placer de proxy pour la valeur null : le choix de la propriété (ownership) de la relation est important.

Exemple 5 : méthodes d'interrogation

Brice rappelle les différentes techniques proposées par JPA pour requêter la base de données :

1. Utilisation d'un createQuery(« requête HQL ») => détection d'une erreur de typo au runtime
2. Requête nommée : définie dans le mapping, elle est interprétée et validée au démarrage de l'unité de persistance. Cas d'usage limité car requête statique.
3. API Criteria : une version typé et une autre non typée. L'utilisation du méta-modèle est à privilégier. Le méta-modèle est généré à la compilation par un processeur d'annotation (Java 6 nécessaire).
4. Spring Data JPA: les paramètres étant passés dans la signature des méthodes, on renforce l'utilisation des requêtes nommées

Exemple 6 : utilisation des paramètres nommés

Comparaison entre l'utilisation ou non d'un paramètre nommé sur une requête ne remontant aucune entité et en utilisant un pool de threads.

Problèmes rencontrés :

- Plans d'exécution non réutilisé (DBA mécontents)
- HibernateQueryPlanCache prend 60 Mo
- Blocage dans le pool C3PO qui a lui même un cache

Exemple 7 : stratégie d'héritage

Brice expose les 3 stratégies de mapping permettant de mapper l'héritage :

1. Une seule table : problème des contraintes not null
2. Une table par classe concrète : union entre les tables et pas de contraintes d'intégrité référentielle
3. Une table par classe (stratégie par jointure) : a la préférence de Brice

Lors d'une requête polymorphique sur les Véhicules avec de nombreux outer join (jointures index), les SGBD ne peuvent pas utiliser leurs index.

Solutions proposées par Brice :

1. Possibilité d'utiliser un VehicleDto avec un constructeur à N paramètres => cela fonctionne.
2. Possibilité d'utiliser un VehicleDto avec un constructeur avec le véhicule remonté. N+1 requêtes. Hibernate commence par une shallow request (une requête ne remontant que les ID des 10 véhicules). Pourquoi Hibernate prend il cette décision ? Jira sans suite.