Applications orientées données (NSY135)

12 – Optimisation des lectures

Auteurs: Raphaël Fournier-S'niehotta et Philippe Rigaux
(philippe.rigaux@cnam.fr,fournier@cnam.fr)

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Introduction

- On va maintenant chercher à contrôler les requêtes SQL engendrées par Hibernate
- Le but est d'éviter de se retrouver dans la situation défavorable où de nombreuses requêtes sont exécutées pour charger, petit à petit, des parties minimes du graphe de données.
- Cette situation apparaît typiquement quand le graphe à matérialiser est découvert par la couche Hibernate au fur et à mesure des navigations de l'application.
- Comme on sait souvent **à l'avance** à quelles données elle va accéder, on peut donc anticiper le chargement grâce à des requêtes SQL (idéalement, une seule)
- et ainsi obtenir facilement l'intégralité la partie du graphe contenant ces données visitées (et minimiser les requêtes nécessaires)

Plan du cours

Stratégie de chargement

- Pour effectuer cette optimisation, on définit une **stratégie de chargement**
- Elle indique quel est le voisinage d'un objet persistant qui doit être matérialisé quand cet objet est lui même chargé
- On peut paramétrer cette stratégie à deux niveaux :
  - **dans la configuration du mapping**, autrement dit sous forme d'une annotation JPA;
  - **à l'exécution de requêtes HQL**, grâce au mot-clé **fetch**, déjà évoqué.
- Ces deux niveaux sont complémentaires :
  - La spécification au niveau de la configuration a l'inconvénient d'être insensible aux différents contextes dans lesquels la base de données est interrogée
  - dans certains cas, quand on accède à un film, on voudra charger les acteurs, dans d'autres, ce seront les notes, dans d'autres, ni l'un ni l'autre

Stratégies de chargement

- La bonne méthode :
  - On indique une stratégie de chargement **par défaut** au niveau de la configuration
  - On la surcharge grâce à des requêtes HQL spécifiques dans les contextes où c'est nécessaire

- L'étude des stratégies de chargement est rendue compliquée par plusieurs facteurs défavorables :
  - la stratégie par défaut (si on ne spécifie rien) tend à changer d'une version Hibernate à une autre;
  - la stratégie par défaut n'est pas la même en Hibernate et en JPA;
  - la documentation est assez obscure

- Il est donc préférable de ne pas se fier à la stratégie par défaut mais de toujours en donner une explicitement

- Et il est sans doute nécessaire de vérifier le comportement d'Hibernate sur les requêtes les plus importantes de notre application, pour éviter des accès sous-optimaux à la base sous-jacente.

Petite étude de cas : le problème

- Nous reprenons les requêtes du contrôleur **Requeteur**
- Il y a déjà une action **lectureParCle** dans le contrôleur, elle se contente de charger un film par son id avec **load()** ou **get()**
- On lui associe la vue minimale suivante :

```
  <%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

  <html>
   <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Accès à un film par la clé, sans navigation</title>
  </head>
  <body>

<h2>Le film no 1</h2>

Nous avons bien ramené le film ${film.titre}. Et c'est tout.

   </body>
  </html>
```

Petite étude de cas : le problème

- Nous n'accédons donc à aucun objet associé. Seul le titre est affiché. Exécutez cette action et examinez la requête SQL engendrée par Hibernate.
- Si vous avez utilisé les annotations JPA, cette requête devrait être la suivante (sans la liste du **select** pour ne conserver que les jointures).

```
select [...]
from Film film0_ left outer join Genre genre1_ on film0_.genre=genre1_.code
   left outer join Pays pays2_ on film0_.code_pays=pays2_.code
     left outer join Artiste artiste3_ on film0_.id_realisateur=artiste3_.id
where film0_.id=?
```

- Hibernate a donc engendré une requête qui effectue une jointure (externe) pour toutes les associations de type **@ManyToOne**.
- C'est clairement inutile ici puisque nous n'accédons ni au genre, ni au pays, ni au metteur en scène.

Petite étude de cas : le problème

- Faisons la même expérience, cette fois en utilisant HQL.
- on a écrit une action **parTitre()** qui exécute la méthode suivante

```
public List<Film> parTitre(String titre)
{
Query q = session.createQuery("from Film f where f.titre= :titre");
q.setString ("titre", titre);
return q.list();
}
```

- Associons lui la vue suivante, qui affiche pour chaque film son titre et le nom du réalisateur.

```
<h2>Les films</h2>

Voici les films ramenés:

<ul>
<c:forEach items="${films}" var="film">
<li>Nous avons bien ramené le film ${film.titre} dont le
réalisateur est ${film.realisateur.nom}</li>
</c:forEach>
</ul>
```

Petite étude de cas : le problème

- En exécutant l'action, il y a 4 requêtes SQL :
  - la première correspond à l'exécution de la requête HQL.
  - les trois suivantes correspondent, respectivement, à la recherche du genre, du pays et du réalisateur; soit :

```
select * from Genre genre0_ where genre0_.id=?
select * from Pays pays0_ where pays0_.id=?
select * from Artiste artiste0_ where artiste0_.id=?
```

  - Les id recherchés sont les valeurs des clés étrangères trouvées dans **Film**.

- Les seconde et troisième requêtes sont ici inutiles.

- La quatrième ne l'est que parce que nous affichons le réalisateur, mais en fait Hibernate effectue "aveuglément" et systématiquement la recherche des objets liés au film par une association **@ManyToOne**, dès que ce film est placé dans le cache, sans se soucier de savoir si ces objets vont être utilisés.

- Il s'agit d'une stratégie dite de **chargement immédiat** (ou "glouton", **eager** en anglais), en général peu efficace.

Petite étude de cas : la solution

- Pour remédier à ça, on indique explicitement la stratégie de chargement à appliquer
- On édite la classe **Film.java** en complètant toutes les annotations **@ManyToOne** comme suit :

```
@ManyToOne(fetch=FetchType.LAZY)
```

- On souhaite un chargement “paresseux” (**lazy**)
- En exécutant à nouveau l’action de recherche par clé, la requête devient :

```
select [...] from Film film0_ where film0_.id=?
```

- On ne charge que ce qui est nécessaire, sans jointure superflue.

Solution (suite)

- Considérons la seconde action, celle qui effectue une recherche HQL.
- Si vous la ré-exécutez avec la stratégie indiquée, on a deux requêtes SQL :

```
select [...] from Film film0_ where film0_.id=?
select * from Artiste artiste0_ where artiste0_.id=?
```

- La première requête est toujours celle correspondant à la requête HQL
- La seconde est déclenchée **par navigation**
- quand on veut afficher le nom du réalisateur, Hibernate déclenche l'accès à l'artiste et son chargement
- Cet accès est nécessaire car l'objet n'est pas présent dans le cache, puisque Hibernate est maintenant en mode **Lazy**

Solution (suite)

- Cette stratégie est donc parfaite pour notre première action
- mais pas pour la seconde où une jointure avec la table **Artiste** quand on recherche le film serait plus appropriée
- ici, aucune stratégie générale indiquée au niveau de la configuration, ne peut être optimale dans tous les cas
- Il nous reste une solution, qui est d'adapter la stratégie de chargement au contexte particulier d'utilisation
- dans la méthode **parTitre()**, on remplace la requête HQL par :

```
select film from Film as film join fetch film.realisateur
where film.titre= :titre
```

Solution (suite)

- il n'y a plus qu'une seule requête SQL effectuant la jointure entre **Film** et **Artiste**
- Cette requête charge le film et l'artiste
- surtout, elle **matérialise le graphe d'association entre les deux objets**
- ainsi, quand on navigue du film vers son réalisateur, l'objet **Artiste** est trouvé dans le cache et il n'est plus nécessaire d'effectuer une requête SQL supplémentaire
- C'est le mot-clé **fetch** dans la requête HQL qui entraine une surcharge de la stratégie de chargement par défaut

Récapitulons

- Hibernate (ou JPA) applique une stratégie par défaut qui tend à multiplier les requêtes SQL ou les jointures externes inutilement.
- Cette stratégie par défaut peut être remplacée par une configuration explicite dans les annotations de **mapping**.
- Une stratégie au niveau de la configuration ne peut être optimale dans tous les cas: on peut alors, **dans un contexte donnée**, la remplacer par un chargement adapté au contexte, exprimé en HQL avec l'option **fetch**.

- Limitation :
  - on a pris l'hypothèse simple que l'on cherche à minimiser le nombre de requêtes SQL
  - c'est une bonne approche en général, car elle laisse l'optimiseur du SGBD déterminer la bonne méthode d'exécution (en fonction de ses ressources)
  - bien sûr, il se peut qu'une requête SQL engendrée soit trop complexe et pénalise au contraire les performances
  - dans ce cas, il faut expertiser les plans d'exécution de requêtes, cf autre cours (`http://sys.bdpedia.fr`)