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

Plan du cours

2 Configuration des stratégies de chargement
- Les stratégies Hibernate/JPA
- **Eager**, le chargement glouton
- Le mode **Lazy**
- Le problème des 1+n requêtes
- Le chargement par lot, une variante de **Lazy**

Les stratégies Hibernate/JPA

- Étant donné un objet **x** chargé par Hibernate, désigné par **entité principale**, l'expression **entité (secondaire)** désigne un objet **y** lié à **x** via une assocation **@ToOne**
- L'expression **collection** désigne un ensemble d'objets, liés à **x** donc via une association **@ToMany**.
- Si, par exemple un objet instance de **Film** est l'entité principale, le réalisateur (instance de **Artiste**) est une entité (secondaire) et les rôles du film constituent une collection.
- Nous avons donc essentiellement deux stratégies de chargement, s'appliquant aux entités et collections:
  - **Eager** ("glouton", "gourmand" en anglais) consiste à charger une entité ou une collection le plus tôt possible, indépendamment du fait que l'application les utilise ou non par la suite.
  - **Lazy** ("paresseux" en anglais) indique au contraire qu'une entité ou une collection est chargée le plus tard possible, au moment où l'application cherche à y accéder.

Stratégies de chargement

- Par défaut, JPA charge les entités avec une stratégie **Eager**, et les collections avec une stratégie **Lazy**.
- C'est cette stratégie qui est appliquée quand on utilise les annotations JPA, même quand le moteur implantant JPA est Hibernate.

- Avec la stratégie par défaut JPA, le réalisateur, le pays et le genre sont des entités chargées en mode **Eager** quand un film est chargé dans la session;
- les rôles ne sont chargés que quand l'application tente de les récupérer avec la méthode **getRoles()**
- Vérifions sur l'étude de cas

♠ Hibernate applique une stratégie par défaut différente de JPA: tout est chargé en mode **Lazy**.

Eager, le chargement glouton

- Dans ce mode, Hibernate tente d'associer le plus tôt possible les entités secondaires à l'entité principale
- En pratique, deux méthodes sont utilisées :
  - **Par des jointures externes**. C'est notamment le cas quand on charge un objet par **load()** ou **get()**. Hibernate engendre alors une requête comprenant des **outer join**.
  - **Par des requêtes SQL complémentaires**. Si Hibernate n'a pas pu anticiper en plaçant des **outer join**, des requêtes SQL sont effectuées, une par entité secondaire. C'est le cas par exemple quand l'entité principale est obtenue par une requête HQL, ou quand elle fait partie d'une collection chargée de manière paresseuse.

Eager, le chargement glouton (suite)

- Le mode **Eager** ne se justifie que dans deux cas :
  - on est sûr que l'application accèdera toujours ou presque toujours aux entités secondaires quand une entité principale est chargée;
  - on est sûr que les entités secondaires sont dans un cache de premier ou de second niveau.
- Il semble difficile d'assurer qu'une application, pour toujours et en toutes circonstances, satisfaira une des deux conditions ci-dessus.
- Cette stragégie a le grave inconvénient d'engendrer parfois en grande quantité des requêtes SQL élémentaires, ramenant chacune une seule ligne.
- Il est donc recommandé de **toujours adopter le mode Lazy par défaut**
- Cela revient à compléter les annotations **@OneToOne** ou **@ManyToOne** comme suit :

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

- Pour les associations de type **@ToMany**, le mode par défaut est toujours **Lazy**
- il est donc inutile de compléter (mais on peut le faire, pour la clarté).

Mode Lazy

- Le mode **Lazy** est le mode par défaut pour les collections.
- Cela semble indispensable car la taille de la collection n'est pas connue a priori
- se mettre en mode **Eager** ferait courir le risque de charger un très grand nombre d'objets, sans garantie sur leur utilisation, et un coût d'initialisation et de stockage en mémoire élevé.
- Exemple : le chargement de **tous** les films d'un pays comme les USA si on associait une collection **films** en mode **Eager** à la classe **Pays**.
- Par défaut, toutes les collections devraient donc être en mode **Lazy**.
- Aucune solution n'étant idéale en toutes circonstances, ce mode peut donner lieu à un chargement inefficace, caractérisé par l'expression **1+n requêtes**.

Les 1+n requêtes

- Supposions que l'on souhaite parcourir tous les films de la base et analyser les notes de chacun
- La structure de notre action serait la suivante :

```
// On recherche les films
Set<Film> films = session.createQuery ("from Film");

for (Film film: films) {
  // Pour chaque film on traite les notes
  for (Notation notation: films.getNotation()) {
    // Faire qq chose avec la notation
  }
}
```

- Nous avons deux boucles imbriquées, engendrant une requête SQL chacune.
- La première sélectionne tous les films et ressemble simplement à:

```
select * from Film
```

- La seconde sélectionne les notes d'un film donné :

```
select * from Notation where id_film=:film.id
```

Les 1+n requêtes (suite)

- La première est exécutée **une fois**, la seconde **autant de fois qu'il y a de films**, d'où la caractérisation par l'expression '1+n' requêtes.
- Le problème est que cette stratégie est inefficace car elle soumet potentiellement beaucoup de requêtes au SGBD.
- Si vous avez 10 000 films, vous exécuterez 10 000 fois la même requête, alors que les SGBD sont conçus, grâce à l'opération de jointure, pour ramener tous les objets en une seule requête.
- On peut envisager plusieurs solutions au problème :
  - en changeant la configuration pour accéder à la collection en mode **Eager**, mais utiliser une configuration générale pour résoudre un problème survenant dans un contexte spécifique ne fait que transposer le problème ailleurs
  - en utilisant le **chargement par lot** (suite de cette section)
  - en utilisant une requête HQL de chargement (la meilleure, cf section suivante).

Le chargement par lot, une variante de **Lazy**

- Le chargement par lot (**batch fetching**) permet de factoriser les requêtes effectuées pour obtenir les collections associées aux objets de la boucle extérieure (celle sur les films dans notre exemple).
- Il est caractérisé par la taille d'un lot, **k**, exprimant le nombre de collections recherchées en une seule fois.
- La syntaxe de l'annotation est la suivante:

```
@BatchSize(size=k)
```

- Avec nos 10 000 films, et 10 000 requêtes cherchant la collection **roles** de chaque film, en faisant un chargement par lot de taille 10, on va grouper la recherche des collections 10 par 10.

Chargement par lot

- Supposons pour simplifier que les identifiants des films sont séquentiels (1, 2, 3)
- Quand l'application cherche à accéder à la collection **roles** du **premier** film, la requête suivante sera effectuée:

```
select * from Notation where id_film in (1, 2, 3, 4, 5, 6, 7, 8 , 9, 10)
```

- L'accès aux **roles** pour les films 2, 3, 4, ..., 10 se fera donc dans le cache, sans avoir besoin d'effectuer une nouvelle requête SQL.
- L'accès aux roles du film 10 déclenchera la requête (et l'initialisation des collections des premiers films) :

```
select * from Notation where id_film in (11, 12, 13, 14, 15, 16, 17, 18 , 19)
```

- idem pour la suite
- Avec ce paramétrage, on est passé du problème des 1+n requêtes au problème des 1+n/10 requêtes!

Chargement par lot

- C'est donc un mode intermédiaire entre **Eager** et **Lazy**.
- Incontestablement, il va amener une amélioration des performances de l'application.
- Cela peut être une solution simple et satisfaisante, même si elle repose sur un paramétrage dont la valeur n'est pas évidente à fixer.
- Cela dit, on peut aisément faire la même critique : le paramétrage du lot dans la configuration n'est sans doute pas adapté à tous les contextes présents dans l'application.
- Dans la mesure où on peut, **localement**, dans un contexte donné, opter pour une stratégie **Eager**, pourquoi se priver de le faire, ce qui est à la fois simple, efficace et compréhensible ?
- Cela passe par l'utilisation de requêtes de chargement HQL.