Applications orientées données (NSY135)

8 – JPA : le mapping

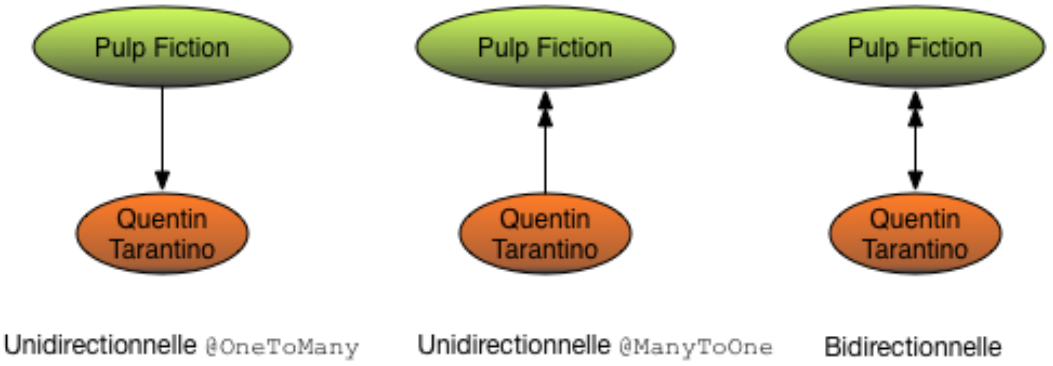
Auteurs: Raphaël Fournier-S'niehotta et Philippe Rigaux
(philippe.rigaux@cnam.fr,fournier@cnam.fr)

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

Plan du cours

S2 Associations un-à-plusieurs

Introduction



- les associations un à un sont peu fréquentes
- on va se concentrer sur les associations “un à plusieurs”
- exemple : relation entre un film et son (unique) réalisateur
- en java, cette association peut se représenter de 3 manières :
  - dans un film, on place un lien vers le réalisateur (unidirectionnel, à gauche);
  - dans un artiste, on place des liens vers les films qu'il a réalisés (unidirectionnel, centre);
  - on représente les liens des deux côtés (bidirectionnel, droite).
- dans une base relationnelle, le problème ne se pose pas: une association représentée par une clé étrangère est par nature bidirectionnelle

Annotation @ManyToOne

- Dans la classe **Film**, nous indiquons qu'un objet lié nommé **réalisateur**, de la classe **Artiste**, doit être cherché dans la base par Hibernate.

```
@ManyToOne
@JoinColumn (name="id_realisateur")
private Artiste realisateur;
public void setRealisateur(Artiste a) {realisateur = a;}
public Artiste getRealisateur() {return realisateur;}
```

- L'annotation **@JoinColumn** indique quelle est la **clé étrangère** dans **Film** qui permet de rechercher l'artiste concerné.
- Donc, quand on appelle **getRealisateur()**, Hibernate doit engendrer et exécuter la requête suivante (**?film** désigne l'objet-film courant)

```
select * from Artiste where id = ?film.id_realisateur
```

- avec le comportement simpliste décrit ci-dessus, on effectue une requête SQL pour rechercher **un** objet, ce qui constitue une utilisation totalement sous-optimale de SQL et risque d'engendrer de très gros problèmes de performance pour des bases importantes.

Représentation avec Hibernate

- Hibernate sait bien entendu représenter l'association (clé primaire / clé étrangère) dans la base, en transposant la référence objet d'un artiste par un film. Créons l'action **insertFilm** ci-dessous.

```
public void insertFilm() {
  session.beginTransaction();
  Film gravity = new Film();
  gravity.setTitre("Gravity");
  gravity.setAnnee(2013);

  Genre genre = new Genre();
  genre.setCode("Science-fiction");
  gravity.setGenre(genre);

  Artiste cuaron = new Artiste();
  cuaron.setPrenom("Alfonso");
  cuaron.setNom("Cuaron");

  // Le réalisateur de Gravity est Alfonso Cuaron
  gravity.setRealisateur(cuaron);

  // Sauvegardons dans la base
  session.save(gravity);
  session.save(cuaron);
  session.getTransaction().commit();
}
```

Représentation avec Hibernate (suite)

- On a donc créé le film **Gravity**, de genre **Science-Fiction** et mis en scène par Alfonso Cuaron. L'association est créée, **au niveau du graphe d'objets java** par l'instruction :

```
gravity.setRealisateur(cuaron);
```

- On a ensuite sauvegardé les deux nouvelles entités dans la base. Ce qui s'affiche dans la console après exécution : Hibernate a engendré des requêtes SQL.

```
Hibernate: select genre_.code from Genre genre_ where genre_.code=?
Hibernate: insert into Film (annee, genre, code_pays,
           d_realisateur, resume, titre) values (?, ?, ?, ?, ?, ?)
Hibernate: insert into Artiste (annee_naissance, nom, prenom) values (?, ?, ?)
Hibernate: update Film set annee=?, genre=?, code_pays=?, id_realisateur=?,
           resume=?, titre=? where id=?
```

- La première sélectionne le genre **Science-Fiction** : comme nous avons indiqué la valeur de la clé primaire dans l'instance Java, Hibernate vérifie automatiquement si cet objet **Genre** existe dans la base et va le lire si c'est le cas pour le lier par référence à l'objet **Film** en cours de création.
- Hibernate cherche toujours à synchroniser le graphe des objets et la base de données quand c'est nécessaire.

Représentation avec Hibernate (suite)

- On a donc créé le film **Gravity**, de genre **Science-Fiction** et mis en scène par Alfonso Cuaron. L'association est créée, **au niveau du graphe d'objets java** par l'instruction :

```
gravity.setRealisateur(cuaron);
```

- On a ensuite sauvegardé les deux nouvelles entités dans la base. Ce qui s'affiche dans la console après exécution : Hibernate a engendré des requêtes SQL.

```
Hibernate: select genre_.code from Genre genre_ where genre_.code=?
Hibernate: insert into Film (annee, genre, code_pays,
            d_realisateur, resume, titre) values (?, ?, ?, ?, ?, ?)
Hibernate: insert into Artiste (annee_naissance, nom, prenom) values (?, ?, ?)
Hibernate: update Film set annee=?, genre=?, code_pays=?, id_realisateur=?,
            resume=?, titre=? where id=?
```

- Le premier **insert** est déclenché par l'instruction **session.save(gravity)**. Il insère le film. À ce stade l'identifiant du metteur en scène dans la base est à **NULL** puisque Cuaron n'a pas encore été sauvegardé. Nous pouvons donc, à un moment donné, associer un objet persistant (le film ici) à un objet transient (l'artiste).

Représentation avec Hibernate (suite)

- On a donc créé le film **Gravity**, de genre **Science-Fiction** et mis en scène par Alfonso Cuaron. L'association est créée, **au niveau du graphe d'objets java** par l'instruction :

```
gravity.setRealisateur(cuaron);
```

- On a ensuite sauvegardé les deux nouvelles entités dans la base. Ce qui s'affiche dans la console après exécution : Hibernate a engendré des requêtes SQL.

```
Hibernate: select genre_.code from Genre genre_ where genre_.code=?
Hibernate: insert into Film (annee, genre, code_pays,
            d_realisateur, resume, titre) values (?, ?, ?, ?, ?, ?)
Hibernate: insert into Artiste (annee_naissance, nom, prenom) values (?, ?, ?)
Hibernate: update Film set annee=?, genre=?, code_pays=?, id_realisateur=?,
            resume=?, titre=? where id=?
```

- Le second **insert** correspond à **session.save(cuaron)**. Il insère l'artiste, qui obtient alors un identifiant dans la base de données. Et du coup Hibernate effectue un **update** sur le film pour affecter cet identifiant à la colonne **id_realisateur**.

Représentation avec Hibernate (suite)

- On remarque que tous les champs de **Film** sont modifiés par l'**update** alors que seul l'identifiant du metteur en scène a changé.
- Hibernate ne gère pas les modifications au niveau des attributs mais se contente - c'est déjà beaucoup - de détecter toute modification de l'objet java pour déclencher une synchronisation avec la base. (cf plus tard avec transactions)
- Pour un effort minime nous obtenons donc une gestion complète de l'association entre un film et son metteur en scène.
- Hibernate synchronise les actions effectuées sur le graphe des objets persistants en java avec la base de données, que ce soit en lecture ou en écriture.
- Remarquons une différence entre la représentation relationnelle et le modèle de données en java :
  - Il n'est pour l'instant pas possible de récupérer la liste des films réalisés par Alfonson Cuaron car nous n'avons pas représenté dans le modèle objet (Java) ce côté de l'association.
  - En revanche, une simple jointure en SQL nous donnerait cette information.

Annotation **@OneToMany**

- De l'autre côté de l'association, on utilise l'annotation **@OneToMany**: à un objet correspondent plusieurs objets associés.
- Dans notre exemple, à un artiste correspondent de 0 à plusieurs films réalisés.

```
    /* @ManyToOne
   @JoinColumn (name="id_realisateur")
   private Artiste realisateur;
   public void setRealisateur(Artiste a) {realisateur = a;}
   public Artiste getRealisateur() {return realisateur;}
*/
```

- Mettons en commentaires, pour l'instant, la clause **@ManyToOne** définissant le réalisateur dans la classe **Film**, puisque nous étudions les associations unidirectionnelles : nous verrons dans la prochaine session ce qui se passe quand on combine les deux.

Annotation **@OneToMany**

- Spécifions l'association **@OneToMany** dans la classe **Artiste**.

```
@OneToMany
@JoinColumn(name="id_realisateur")
private Set<Film> filmsRealises = new HashSet<Film>();
public void addFilmsRealise(Film f) {filmsRealises.add(f) ;}
public Set<Film> getFilmsRealises() {return filmsRealises;}
```

- Il n'y a pratiquement pas de différence avec la représentation **@ManyToOne**.
- On indique, comme précédemment, que la clé étrangère est **id_realisateur** et Hibernate comprend qu'il s'agit d'un attribut de **Film**, ce qui lui permet d'engendrer la même requête SQL que précédemment.
- L'association est représentée par une collection, ici un **Set** (notez qu'on l'initialise avec un ensemble vide).
- On fournit deux méthodes, **add** (au lieu de **set** pour le côté **@ManyToOne**) et **get**.

Annotation **@OneToMany**

- Tentons à nouveau l'insertion d'un nouveau film et de son metteur en scène.
- Il suffit d'exécuter la même méthode **insertFilm** que précédemment, en remplaçant la ligne :

```
gravity.setRealisateur(cuaron);
```

par la ligne :

```
cuaron.addFilmsRealise(gravity);
```

- Supprimez éventuellement les données insérées précédemment, dans phpMyAdmin, avec:

```
DELETE FROM Film where titre='Gravity';
DELETE FROM Artiste WHERE nom='Cuaron'
```

Annotation **@OneToMany**

- Après exécution, Hibernate devrait afficher les requêtes suivantes dans la console:

```
Hibernate: select genre_.code from Genre genre_ where genre_.code=?
Hibernate: insert into Film (annee, genre, code_pays, id_realisateur,
               resume, titre) values (?, ?, ?, ?, ?, ?)
Hibernate: insert into Artiste (annee_naissance, nom, prenom) values (?, ?, ?)
Hibernate: update Film set id_realisateur=? where id=?
```

- Hibernate a effectué les insertions et les mises à jour identiques à celles déjà vues pour l'association **@ManyToOne**.
- Là aussi, c'est normal puisque nous avons changé l'association en java, mais au niveau de la base elle reste représentée de la même manière.
- Cette fois il n'est plus possible de connaître le metteur en scène d'un film puisque nous avons supprimé un des côtés de l'association.
- Il va donc falloir représenter une association bidirectionnelle.
- C'est ce que nous étudions dans la prochaine session. Faites au préalable l'exercice.