Applications orientées données (NSY135)

13 – Dynamique des objets persistants

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

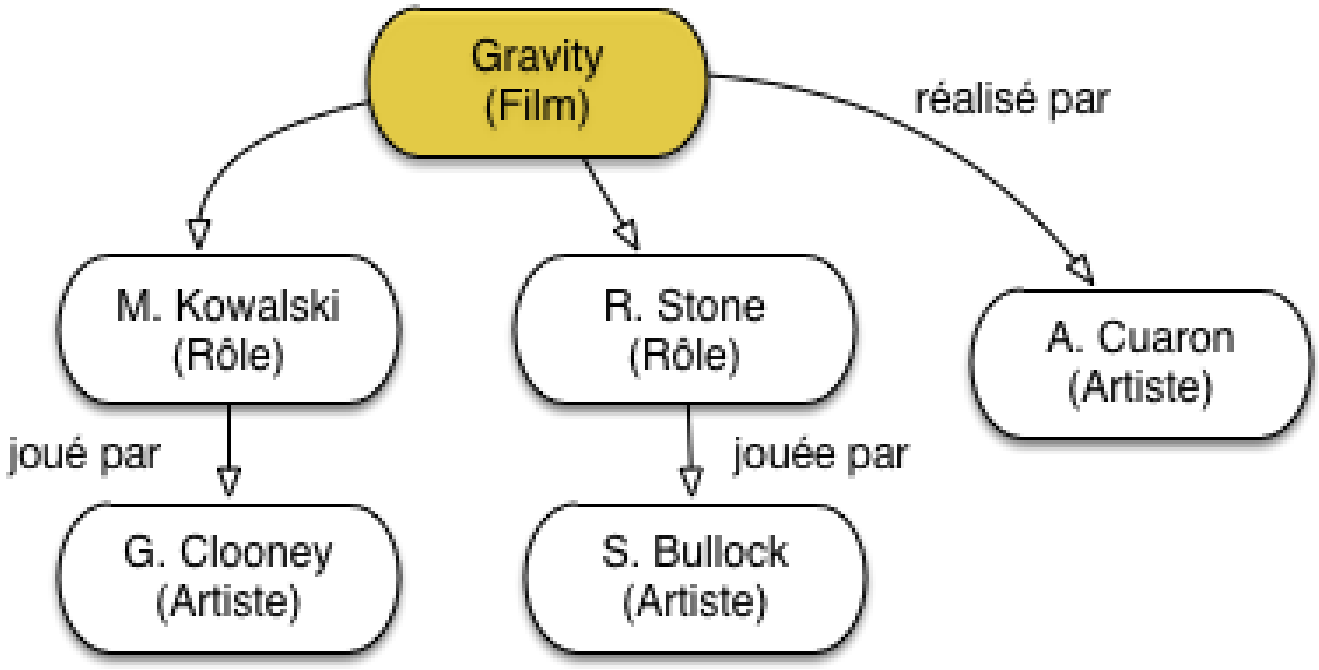
Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Plan du cours

Introduction

- Avec ce qui précède, si l'on crée un graphe d'objets connectés, les instructions rendant les objets persistants doivent préserver la **cohérence** de ce graphe
- En d'autres termes, on ne peut pas rendre certains objets persistants et d'autres transients sous peine d'aboutir à une version incohérente de la base.
- Un des rôles de la transaction est d'assurer la **cohérence (transactionnelle)** des commandes de mise à jour (A**C**ID).
- Assurer cette cohérence manuellement en gérant les situations au cas par cas est source d'erreur et de programmation répétitive.
- Avec JPA/Hibernate, on peut introduire dans le modèle de données une spécification des contraintes de cohérences, qui sont alors automatiquement appliquées.

Graphe d'objet à rendre persistants



Cohérence transactionnelle

- Si, dans ce graphe d'objet, une partie seulement est rendu persistante (par exemple le film), alors il est clair que la représentation en base de données sera incomplète.
- Au niveau de la base elle-même, les contraintes d'intégrité référentielle (décrites par les clés étrangères) ne seront pas respectées, et le SGBD rejettera la transaction.
- Mais avant même cela, Hibernate détectera que la méthode **save()** est appliquée à un objet transient qui référence d'autres objets transients, avec risque potentiel d'incohérence.
- Une exception sera donc levée avant même la tentative d'insertion en base.
- Rappel : la solution "manuelle" consiste à appliquer la méthode **save()** sur **chaque** objet du graphe,
- **dans un ordre bien défini**, pour éviter de rendre persistant un objet référençant un objet transient.

Cohérence transactionnelle

- Dans les bases objets (peu répandues), cette cohérence transactionnelle est prise en charge par le SGBD qui applique un principe dit de **persistance par atteignabilité**.
- Ce principe n'existe pas dans les SGBD relationnelles, la notion la plus proche étant celle d'intégrité référentielle.
- L'option **cascade** sert à spécifier la prise en charge par Hibernate de la cohérence transactionnelle.

L'option Cascade

- La propagation "en cascade" des instructions de persistance dans un graphe d'objet est spécifiée par une annotation **@Cascade**.
- **Dans ce qui suit nous utilisons les options de JPA, et pas celles d'Hibernate**.
- De plus, la méthode **save()** de la session Hibernate semble ne pas reconnaitre les annotations JPA, alors que cela fonctionne bien pour la méthode **persist()** que nous utilisons donc. Ces problèmes seront probablement réglés dans une prochaine version.

Persistance du réalisateur

- Voici donc simplement comment on indique que le réalisateur doit être rendu persistant quand le film est lui-même rendu persistant (classe **Film.java**).

```
1        @ManyToOne (fetch=FetchType.LAZY, cascade=CascadeType.PERSIST)
2        @JoinColumn(name = "id_realisateur")
3        private Artiste realisateur;
```

- Si vous appelez maintenant la méthode **session.persist(gravity)**, la persistance s'appliquera par transitivité au metteur en scène

Opérations en cascade

- Les opérations de persistance à "cascader" sont les créations, modifications et destructions, et peuvent se placer des deux côtés de l'association.
- Voici les annotations (JPA) correspondantes, énumérées par **CascadeType** :
  - **PERSIST**. Quand on appelle la méthode **persist()** sur l'objet référençant (le film dans notre exemple), l'objet référencé (l'artiste dans notre exemple) devient également persistant par appel à la méthode **persist()**, qui est donc éventuellement propagée transitivement à son tour.
  - **REMOVE**. Un appel à **remove()** sur l'objet référençant est propagé à l'objet référencé. Attention à bien considérer cette option : Sur notre schéma, la suppression d'un film **ne devrait pas** entraîner la suppression du réalisateur. En revanche la suppression d'un artiste peut entraîner celle des films qu'il/elle a réalisés, et entraîne certainement celle des rôles qu'il/elle a joués.
  - **MERGE**. La méthode **merge()** JPA est équivalente à **saveOrUpdate()** en Hibernate. Cette option de cascade indique donc une propagation des opérations d'insertion **ou** de mise à jour sur les objets référencés.
  - **REFRESH**. La méthode **refresh()** syncrhonise l'objet par lecture de son état dans la base. Cette option de cascade indique donc une propagation de l'opération sur les objets référencés.
  - **ALL**. Couvre l'ensemble des opérations précédentes.

Opérations en cascade

- N'utilisez surtout pas **CascadeType.ALL** aveuglément !
- La persistance transitive complète s'impose pour les associations de nature compositionnelle par exemple une commande et ses lignes de commande, dans lesquelles le sort d'un composé est étroitement lié à celui du composant (mais l'inverse n'est pas vrai !)
- Elle est aussi normale pour les entités qui représentent (conceptuellement) une association plusieurs-plusieurs dans le modèle de l'application.
- Dans notre schéma, c'est le cas pour les rôles par exemple, dont la préservation n'a pas de sens dès lors que le film ou l'acteur est supprimé.
- Dans toutes les autres situations, une réfléxion au cas par cas s'impose pour éviter de déclencher automatiquement des opérations non souhaitées (et surtout des destructions).
- Il est possible de combiner plusieurs options en les plaçant entre accolades :

```
cascade = {CascadeType.PERSIST, CASCADE.MERGE}
```

Plan du cours

Résumé

- La question des mises à jour est souvent plus délicate que celle des recherches car son impact sur la base de données est plus grand.
- Toute mise à jour s'effectue dans le contexte d'une **transaction**, qui s'exécute de manière **atomique** (tout est validé, ou rien);
- Une transaction peut lever des exceptions qui risquent de laisser la couche ORM dans un état incohérent (synchronisation incomplète entre la base et la session): toute levée d'exception doit entraîner la fermeture de la session courante;
- Les objets passent par divers statuts, distingués techniquement par la présence d'une valeur pour l'identifiant; le passage au statut **persistant** assure la synchronisation avec la base, il faut tâcher d'avoir un code dans lequel ce statut est clair à chaque instant;
- la persistance d'un graphe d'objet peut s'obtenir avec des annotations **@Cascade**, dont l'option plus simple et la plus sûre à manipuler est **PERSIST**;
- les autres options sont à considérer avec précaution et en connaissance de cause.

Nous avons considéré jusqu'à présent les transactions de manière isolée, alors qu'elles s'exécutent souvent en **concurrence**, c'est le sujet du prochain chapitre.