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

1 Objets transients, persistants, et détachés
- L'interface Transaction
- Statut des objets Hibernate
- Objets persistants
- Objets détachés

Notion de transaction

Une transaction est une séquence d'accès (lectures ou mises à jour) à la base de données qui satisfait 4 propriétés, souvent résumées par l'acronyme **ACID**.

- A comme **Atomicité** : les accès d'une même transaction forment un tout solidaire; ils sont validés ensemble ou annulés ensemble.
- C comme **Cohérence** : une transaction doit permettre de passer d'une état cohérent de la base à un autre état cohérent.
- I comme **Isolation** : une transaction est totalement isolée des transaction concurrentes qui s'exécutent en même temps.
- D comme **Durabilité** : avant la validation (**commit**), les mises à jour d'une transaction sont invisibles par toute autre transaction, après, ils deviennent visibles et définitifs.

Notion de transaction

- Hibernate fournit une interface **Transaction** qui encapsule la communication avec deux types de systèmes transactionnels : les transactions JDBC, et les transactions JTA (**Java Transaction API**).
- Les JTA sont réservées à des applications complexes ayant besoin de recourir à un gestionnaire de transactions réparties.
- Hibernate fonctionne toujours en mode **autocommit=off**, pour éviter de valider aveuglément toutes les requêtes.
- toute série de requêtes / mises à jour effectuée avec Hibernate devrait être intégrée à une **transaction** bien identifiée.

- L'interface **Transaction** est en charge d'interagir avec la base pour initier, soumettre et valider des transactions.
- Cette interface peut lever des exceptions
- Bien les traiter permet d'éviter de se retrouver dans des états incohérents après l'éventuel échec d'une opération.

Interface Transaction

```
    // 1 - Instanciation d'un objet
Pays pays = new Pays();
pays.setCode("is");
pays.setLangue("Islandais");
pays.setNom("Islande");

    // 2 - cet objet devient persistant
Transaction tx = null;
try {
  tx = session.beginTransaction();
  session.save(pays);
  tx.commit();
} catch (RuntimeException e) {
  if (tx != null)
    tx.rollback();
  throw e; // Gérer le message (log, affichage, etc.)
} finally {
  session.close();
}

    // 3 - Ici, l'objet 'pays' est détaché de la session !
```

Interface Transaction

- On marque le début d'une transaction avec **beginTransaction()**
- La fin soit avec **commit()**, soit avec **rollback()**.
- Entre les deux, Hibernate charge des objets persistants dans le cache, et surveille toute modification qui leur est apportée par l'application.
- Un objet modifié est marqué comme **dirty** et sera mis à jour dans la base par une requête **update** (ou **insert** pour une création) le moment venu, c'est-à-dire :
  - à la fin de la transaction, sur un **commit()**;
  - au moment d'un appel explicite à la méthode **flush()** de la session ;
  - quand Hibernate estime nécessaire de synchroniser le cache avec la base de données.
- Le dernier cas correspond à une situation où des données ont été modifiées dans le cache, et où une requête ramenant ces données depuis la base est soumise.
- Hibernate peut estimer nécessaire de synchroniser au préalable le cache avec la base (avec un **flush()**) pour assurer la cohérence entre le résultat de la requête et le cache.
- Après avoir **synchronisé** (effectué les requêtes de mise à jour) il reste possible d'effectuer un **rollback()** ramenant la base à son état initial si on n'a pas validé.

Gérer les exceptions

- Dans la gestion d'exceptions, il y a deux choses à faire impérativement :
  - annuler l'ensemble de la transaction en cours par un **rollback**;
  - fermer la session : une exception peut signifier que le cache n'est plus en phase avec la base de données, et toute poursuite de l'activité peut mener à des résultats imprévisibles.
- La fermeture de la session implique celle de la connexion JDBC, et la suppression de tous les objets persistants situés dans le cache.
- La base elle-même est dans un état cohérent garanti par le **rollback**.

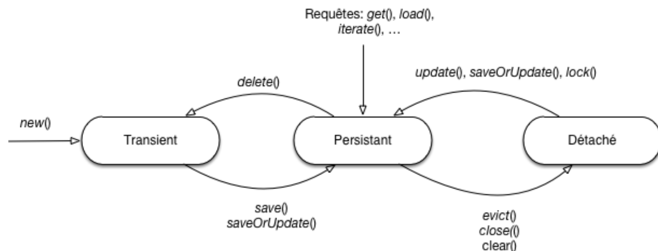
Statut des objets Hibernate

- Dans le code qui précède, on constate qu'un même objet (**pays**) n'est pas uniformément **persistant**.
- Il passe en fait par trois statuts successifs.
  - jusqu'au début de la transaction (et très précisément jusqu'à l'appel à **save()**), l'objet est **transient** : c'est un objet java standard géré par le **garbage manager**;
  - après l'appel à **save()**, l'objet est placé sous le contrôle de la session qui va observer (par des mécanismes d'inspection java) tous les événements qui affectent son état, et synchroniser cet état avec la base : le statut est **persistant**;
  - enfin, après la fermeture de la session, l'objet **pays** existe toujours (puisque l'application maintient une référence) mais il n'est plus synchronisé avec la base : son statut est dit **détaché** (sous-entendu, de la session).

Statut des objets Hibernate

- On peut constater que le statut Hibernate d'un objet peut être géré de manière **complètement indépendante** des services fonctionnels qu'il fournit à l'application.
- En d'autres termes, cette dernière n'a pas à se soucier de savoir si l'objet qu'elle manipule est persistant, transient ou détaché.
- Pensez à l'inconvénient qu'il y aurait à devoir insérer en base un objet alors que l'on veut simplement faire appel à ses méthodes.
- Pensez inversement inversement à l'intérêt de pouvoir utiliser (par exemple dans des tests unitaires) des objets sans les synchroniser avec la base de données.
- Le statut d'un objet change par l'intermédiaire d'interactions avec la session Hibernate.

Cycle de vie des objets Hibernate



- Le statut "persistant" est central dans la figure.
- On constate qu'il existe essentiellement trois méthodes pour qu'un objet **devienne** persistant :
  - un objet transient, instancié avec **new()**, est associé à la session par **save()** ou **saveOrUpdate()**, comme on vient de le voir
  - une ligne de la base de données, sélectionnée par une requête ou une opération de navigation, est **mappée** en objet persistant;
  - un objet **détaché** est "ré-attaché" à une session.

Objets persistants

- Un objet persistant est une instance d'une classe **mappée** qui est associée à une session par l'une des méthodes mentionnées précédemment.
- Par définition, un objet persistant est **synchronisé** avec la base de données : il existe une ligne dans une table qui stocke les propriétés.
- La session surveille l'objet, détecte tout changement dans son état, et ces changements sont reportés automatiquement sur la ligne associée par des requêtes **insert**, **delete** ou **update** selon le cas.
- Le moment où ce report s'effectue est choisi par Hibernate. Au plus tard, c'est à l'appel du **commit()**.

Objets persistants

- L'association a une ligne signifie qu'un objet persistant a également la propriété de disposer d'un identifiant de base de données (celui, donc, de la ligne correspondante).
- Le mode d'acquisition de la valeur pour cet identifiant explique les différentes méthodes disponibles.
- Dans le cas le plus simple, l'identifiant est engendré par une séquence.
- Quand on instancie un objet avec **new()** et que l'on appelle la méthode **save()**, Hibernate va détecter que l'objet est nouvellement instancié et ne dispose par d'identifiant.
- Un appel **insert** à la base va créer la ligne correspondante, et lui affecter une valeur d'identifiant auto-générée.

Objets persistants

- Si l'identifiant n'est pas auto-généré par une séquence, il doit être fourni par l'application (c'est le cas dans notre exemple pour le pays, identifié par son code).
- L'exécution de l'**insert** risque alors d'être rejetée si une ligne avec le même identifiant existe déjà.
- Si, donc, on veut rendre persistant un objet **dont l'identifiant est déjà connu**, il est préférable d'appeler **saveOrUpdate()**. Hibernate déclenchera alors, selon le case un **insert** ou un **update**.
- La destruction d'un objet persistant avec la méthode **delete()** implique d'une part son passage au statut d'objet transient, et d'autre part l'effacement de la ligne correspondante dans la base (par un **delete**).

Objets détachés

- La différence entre un objet **transient** et un objet **détaché** est que ce dernier a été, à un moment donné, associé à une ligne de la base.
- On peut donc **affirmer** que l'objet transient dispose d'une valeur d'identifiant.
- On peut aussi **supposer** que la ligne dans la base existe toujours, mais ce n'est pas garanti puisqu'un objet détaché, du fait de la fermeture de la session, n'est plus synchronisé à la base.
- Dans ces conditions, la méthode **saveOrUpdate()** s'impose naturellement pour les objets détachés qui sont ré-injectés dans une session.
- Hibernate effectue un **update** de la ligne existante, ou un **insert** si la ligne n'existe pas.

Objets détachés

- Les deux autres méthodes pour rendre persistant un objet détaché se comportent différemment.
  - la méthode **update()** indique à Hibernate qu'une synchronisation immédiate avec la commande SQL **update** doit être effectuée; c'est nécessaire quand l'objet détaché a été modifié hors de toute session, et que son état n'a donc pas été synchronisé avec la base.
  - la méthode **lock()** associe l'objet détaché à la session (il devient donc persistant) mais les changements d'état intervenus **pendant** le détachement ne sont pas reportés dans la base.
- Voilà l'essentiel de ce que vous devez savoir pour comprendre et gérer **individuellement** le statut des objets.
- Dans la prochaine session, on va se pencher sur le cas ou des modifications affectent des ensembles d'objets, et plus particulièrement des objets connexes dans le graphe, comme par exemple un film et l'ensemble de ses acteurs.
- En suivant l'approche présentée ici, il faudrait appeler **save()** ou **saveOrUpdate()** sur **chaque** objet créé ou modifié, ce qui n'est ni naturel, ni élégant.