Hibernate - Annexes

Plan

- Cache de second niveau
- Intercepteurs et évènements
- Traitements Batch
- Monitoring

Cache de second niveau

La session fournit un cache de la base durant la transaction

Hibernate permet de configurer un cache de second niveau (niveau de la JVM)

Les objets cachés peuvent être :

- Des entités
- Des collections (relation many)
- Les résultats d'une requête

La configuration s'effectue par la propriété ***hibernate.cache.provider_class***

Options de configuration

Cache	Provider class	Type	Cluster Safe	Query Cache Supported
Hashtable (not intended for production use)	org.hibernate.cache.HashtableCacheProvider	memory		yes
EHCache	org.hibernate.cache.EhCacheProvider	memory, disk		yes
OSCache	org.hibernate.cache.OSCacheProvider	memory, disk		yes
SwarmCache	org.hibernate.cache.SwarmCacheProvider	clustered (ip multicast)	yes (clustered invalidation)	
JBoss Cache 1.x	org.hibernate.cache.TreeCacheProvider	clustered (ip multicast), transactional	yes (replication)	yes (clock sync req.)
JBoss Cache 2	org.hibernate.cache.jbc.JBossCacheRegionFactory	clustered (ip multicast), transactional	yes (replication or invalidation)	yes (clock sync req.)

Configuration

Afin qu'une entité ou une collection soit placée dans le cache de second niveau, il suffit :

- De l'annoter via ***@Cache***
- Ou d'utiliser l'élément ***<cache>*** dans le fichier de mapping

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
public class Forest { ... }
----
@OneToMany(cascade=CascadeType.ALL, fetch=FetchType.EAGER)
@JoinColumn(name="CUST_ID")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
public SortedSet<Ticket> getTickets() {
    return tickets;
}
```

Attributs de *@Cache*

L'annotation *@Cache* propose 3 attributs :

- ***usage (obligatoire)*** : Permet d'optimiser les performances de contrôle de la concurrence d'accès au cache
- ***region (optionnelle)*** : La région est utilisée lorsque l'on veut obtenir des statistiques d'utilisation du cache
- ***include (optionnel)*** : all ou non-lazy, cache toutes les propriétés ou seulement les propriétés non-lazy

Stratégie de concurrence

L'attribut *usage* peut prendre 5 valeurs :

- **NONE :**
- **READ_ONLY** : A utiliser si seules des opérations de lecture sont effectuées
- **READ_WRITE** : Lecture et écriture
- **NONSTRICT_READ_WRITE** : Les opérations d'écriture sont rares et il y a peu de chances que 2 transactions écrivent des données simultanément
- **TRANSACTIONAL** : Seulement dans un environnement JTA

Compatibilité

Cache	read-only	nonstrict-read-write	read-write	transactional
Hashtable (not intended for production use)	yes	yes	yes	
EHCache	yes	yes	yes	
OSCache	yes	yes	yes	
SwarmCache	yes	yes		
JBoss Cache 1.x	yes			yes
JBoss Cache 2	yes			yes

Interactions avec le cache de 2nd niveau

L'attribut ***CacheMode*** permet de contrôler comment une session interagit avec le cache de second niveau. Plusieurs valeurs sont possibles :

- ***CacheMode.NORMAL*** : La session lit et écrit dans le cache de second niveau
- ***CacheMode.GET*** : La session lit à partir du cache de second niveau, n'y écrit pas et l'invalide lors de la mise à jour des données
- ***CacheMode.PUT*** : La session ne lit pas dans le 2nd cache mais y écrit lorsqu'il charge des données de la base

Gestion du cache

Les opérations proposées par l'interface *SessionFactory* permettent de supprimer une entité ou une collection du cache :

```
//Une entité particulière
sessionFactory.evict(Cat.class, catId);
// Toutes les entités de type Cat
sessionFactory.evict(Cat.class);
//Une collection particulière
sessionFactory.evictCollection("Cat.kittens", catId);
//Toutes les collections kittens des entités Cat
sessionFactory.evictCollection("Cat.kittens");
```

Cache de requête

Les résultats des requêtes peuvent être également cachées (même si les cas d'utilisation sont assez rare)

Le cache requête est désactivé par défaut, il faut donc configurer hibernate pour l'activer :
hibernate.cache.use_query_cache true

Ensuite une requête peut être cachée :

- programmatiquement : `org.hibernate.Query.setCacheable(true)`
- Une requête nommée peut également être annotée :

```
@NamedQuery(name = "products.getProducts",
        query = "SELECT product FROM Product product",
        hints = { @QueryHint(name =
"org.hibernate.cacheable", value = "true"),
@QueryHint(name="org.hibernate.cacheMode",
value="NORMAL") })
```

Intercepteurs et évènements
Introduction

- Hibernate permet à l'application de réagir en fonction de certains évènements session.
  - L'interface *Interceptor* permet d'implémenter des méthodes de callback lors de l'ajout, modification ou suppression des entités.
  - Des listeners d'évènements session peuvent être définis dans le fichier de configuration ou via des annotations.

Intercepteurs

Les intercepteurs peuvent être positionnés :

- Sur la *session* :

```
Session session =
sf.openSession( new AuditInterceptor() );
```

- Sur la *session-factory* (valable pour toutes les sessions ouvertes) :

```
new Configuration().
setInterceptor( new AuditInterceptor() );
```

Intercepteurs
Méthodes de callback

void afterTransactionBegin(Transaction tx) : Au démarrage d'une transaction Hibernate
void afterTransactionCompletion(Transaction tx) : Après un commit ou rollback
void beforeTransactionCompletion(Transaction tx) : Avant un commit
void onCollectionRecreate(Object collection, Serializable key) : Chargement ou recharchement d'une collecttion
void onCollectionRemove(Object collection, Serializable key) : Suppression d'une collection
void onCollectionUpdate(Object collection, Serializable key) : Mise à jour d'une collection
void onDelete(Object entity, Serializable id, Object[] state, String[] propertyNames, Type[] types) : Suppression
boolean onFlushDirty(...) : L'entité est détectée désynchronisé lors d'un flush
boolean onLoad(...) : Au chargement d'une entité
String onPrepareStatement(String sql) : Lorsqu'un ordre SQL est construit
boolean onSave(..) : Création d'objet
void postFlush(Iterator entities) : Après un flush
void preFlush(Iterator entities) : Avant le flush

Intercepteurs
Exemple

```
public class AuditInterceptor extends EmptyInterceptor {
    private int updates, creates ;
    public boolean onFlushDirty(Object entity, Serializable id, Object[] newState,
  Object[] oldState,String[] propertyNames,Type[] types) {
            updates++;
            for ( int i=0; i < propertyNames.length; i++ ) {
                if ( "lastUpdateTimestamp".equals( propertyNames[i] ) ) { newState[i] =
  new Date() ; return true ;  }
     }
  public boolean onSave(Object entity, Serializable id, Object[] state, String[]
  propertyNames,Type[] types) {
            creates++;
            for ( int i=0; i<propertyNames.length; i++ ) {
                if ( "createTimestamp".equals( propertyNames[i] ) ) { state[i] = new
  Date() ; return true ; }
            }
  }
  public void afterTransactionCompletion(Transaction tx) {
        if ( tx.wasCommitted() ) { System.out.println("Creations: " + creates + ",
  Updates: " + updates) ;  }
        updates=0; creates=0 ;
     }
```

Evènements
org.hibernate.event

Toutes les méthodes de l'interface *Session* sont associés à des évènements

Lorsqu'une méthode est appelée, Hibernate génère un événement et le passe à tous les listeners enregistrés

Hibernate utilise ce système en positionnant des listeners par défaut

Un listener est considéré comme un singleton et ne doit pas stocker des informations d'états dans ses variables d'instance

Les listeners s'enregistrent
- Par configuration (XML ou annotation)
- Programmatiquement, via l'objet *Configuration*

Exemple

```
<hibernate-configuration>

    <session-factory>

        ...

        <event type="load">

            <listener class="com.eg.MyLoadListener"/>

            <listener
 class="org.hibernate.event.def.DefaultLoadEventListener"/>

        </event>

    </session-factory>

</hibernate-configuration>
```

Exemple

```
public class MyLoadListener implements LoadEventListener {

    // this is the single method defined by the LoadEventListener interface

    public void onLoad(LoadEvent event, LoadEventListener.LoadType loadType)

            throws HibernateException {

        if ( !MySecurity.isAuthorized( event.getEntityClassName(),
 event.getEntityId() ) ) {

            throw MySecurityException("Unauthorized access");

        }

    }

}
```

JPA Méthodes de callback

Ces méthodes peuvent être définies dans la classe du Bean entité
ou
dans une classe dite "listener": cette classe devra alors être indiquée
dans l'annotation *@EntityListener* sur la classe du bean entité

Annotations possibles:

@PrePersist : Avant l'appel à persist
@PostPersist : Après l'insertion en base
@PreUpdate : Avant l'appel à merge
@PostUpdate : Après la mise à jour de l'entité
@PreRemove : Avant l'appel à remove
@PostRemove : Après la suppression en base
@PostLoad : Après un chargement de l'entité

Exemple

```
@Entity
public class Client implements
  java.io.Serializable{
.
.
.
@PostPersist
public void insertion() { ... }
@PostLoad
public void chargement() { ... }
}
```

Exemple EntityListener

```
public class Logging {


  @PostPersist
  public void insertion(Object entite) {
    …
  }
  @PostLoad
  public void chargement(Object entite)
    …
  }
}
-----------
@Entity
@EntityListeners(Logging.class)
public class Personne implements java.io.Serializable {
```

EntityListeners par défaut

Des entity listeners par défaut peuvent être déclarés dans le fichier de mapping *orm.xml*

→ Ils interviennent pour tout bean entité de l'unité de persistance

```
<entity-mappings>
...
  <entity-listeners>
    <entity-listener class="com.sample.Logging">
      <post-persist method-name="insertion"/>
      <post-load method-name="chargement"/>
    </entity-listener>
    <entity-listener class="com.sample.Perform"/>
  </entity-listeners>
...
</entity-mappings>
```

Traitements par paquets

Lorsque l'on désire exécuter de nombreuses mises à jour dans une même méthode, nous sommes confrontés à des problèmes de taille mémoire.

Le code suivant provoquera certainement un *OutOfMemoryException*, car le cache session contient tous les objets Customer

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
    Customer customer = new Customer(.....);
    session.save(customer);
}
tx.commit();
session.close();
```

Conception

Lors de traitement par paquet nécessite plusieurs aspects :

- Configurer les paquets JDBC à une valeur raisonnable pour améliorer les performances

```
hibernate.jdbc.batch_size 20
```

- Désactiver le cache de second-niveau

```
hibernate.cache.use_second_level_cache false
session.setCacheMode(CacheMode.IGNORE)
```

- Effectuer régulièrement un *flush()* et un *clear()* de la session

Exemple insertion

```
Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

for ( int i=0; i<100000; i++ ) {
    Customer customer = new Customer(.....);
    session.save(customer);

    if ( i % 20 == 0 ) { //20, same as the JDBC batch size
        //flush a batch of inserts and release memory:
        session.flush();
        session.clear();
    }
}

tx.commit();
session.close();
```

Monitoring

Des métriques sur une *SessionFactory* sont disponibles via la méthode *sessionFactory.getStatistics()* ou en utilisant le MBean *StatisticsService* précédemment enregistré sur un serveur JMX

Le monitoring peut être activé ou désactivé :
Par configuration : propriété ***hibernate.generate_statistics***
A l'exécution : ***sf.getStatistics().setStatisticsEnabled(true)*** ou ***hibernateStatsBean.setStatisticsEnabled(true)***

Les statistiques peuvent être réinitialisées avec la méthode *clear()*.

Un résumé peut être envoyé à un logger (info level) avec la méthode *logSummary().*

Métriques

Tous les métriques disponibles sont décrits dans l'API ***Statistics***

Ils peuvent être classés en 3 catégories :

- Usage général des Session : nombre d'ouverture de session, de connexions JDBC, etc.
- Compteurs globaux sur les entités, les collections, les requêtes
- Compteurs détaillés sur une entité particulière, une collection, une requête ou une région du cache