Cours Spring JPA, JTA,

DBConnectionPool

1 Présentation Générale

- JPA (Java Persistance API) est le standard J2EE normalisant les services de persistance, et offrant une couche d'abstraction structurelle et fonctionnelle pour designer des services de DAO.
- JPA comme la quasi totalité des API de Java repose sur un principe de spécifications d'interfaces de services, repris et implémentés par des « *Solution Provider* » comme Hibernate.
- JPA propose une grammaire par *@nnotation* pour définir la structuration des propriétés de persistance (champs, relations, ...) des entités dont elle a la charge ; ainsi que des mécanismes passant par un *EntityManager* pour effectuer les opérations transactionnelles invoquées par les classes de services DAO.
- JPA reprend le concept d'EJB *Entity*, qui est un des 3 concepts forts des EJB (« Enterprise Java Bean », avec les *Session* et *Message*), mais permet d'en disposer en dehors d'un Container d'EJB, (qualifié de « lourd »), dans des Containers comme celui de Spring considérés comme légers (logé dans un Container de Servlet, type Tomcat).
- En J2EE (JPA, Hibernate, ...) toutes les requêtes effectuées vers la base de données nécessite un support Transactionnel sous-jacent : on ne peut effectuer de requête simple en dehors d'une Transaction, toutefois l'on peut mentionner si celle ci sera en « *readOnly* ».

  Cela permet d'améliorer quelque peu les performances, puisque les requêtes de lecture ne nécessite pas le même contexte d'exécution que celle d'écriture (notamment en terme de *Commit*/*Rollback*, et donc de conservation de l'état, et de la pile, des données manipulées).

- Rappel : une transaction est définie selon le paradigme/acronyme « A.C.I.D. »
  - Atomicité
    Toutes les requêtes de la Transaction s'exécutent, ou bien aucune d'elle - si une erreur/exception survient durant le processus.
  - Cohérence
    Les transactions prennent le contexte de bases de données dans un état stable/intègre, et doivent le rendre dans ce même état.
  - Isolation
    Les transactions son isolées, et n'influent pas les unes sur les autres.
  - Durabilité
    En sortie de transaction, les modifications effectuées sont persistées dans le SGBD.

- En J2EE/JPA/JTA/Spring Transaction, il suffit qu'une méthode, qualifiée de Transactionnelle, jette une *RuntimeException* pour que le *RollBack* soit invoqué, invalidant les précédentes modifications effectuées lors de la transaction.
- Spring ne fournit pas de *ConnectionPool* (Design Pattern « *Singleton* », famille « *Création* »), afin d'optimiser Connection JDBC à destination de la base de données.

  Avant Tomcat 7, il était d'usage d'utiliser Apache DBCP (*DataBase ConnectionPool*), ou bien C3PO, mais depuis sa version 7, Tomcat intègre un pool de Connection qui a été remanié par rapport à DBCP, et offre une meilleur qualité de services, notamment en terme de modularité et d'options techniques (configuration et d'interception par AOP)

  http://people.apache.org/~fhanik/jdbc-pool/jdbc-pool.html
  http://www.tomcatexpert.com/blog/2012/01/24/using-tomcat-7-jdbc-connection-pool-production

  Exemple d'externalisation de la DataSource en JNDI, avec ConnectionPool Tomcat7

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean" lazy-init="true">
   <property name="jndiName" value="java:comp/env/jdbc/ProductDS" />
</bean>
```

Déclaration dans le context XML Spring

```
<ResourceLink name="jdbc/ProductDS" global="jdbc/ProductDS" type="javax.sql.DataSource"/>
```

context.xml du serveur Tomcat 7

```
<Resource
   auth="Container"
   driverClassName="com.mysql.jdbc.Driver"
   factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
   initialSize="5"
   jdbcInterceptors="org.apache.tomcat.jdbc.pool.interceptor.ConnectionState;
                     org.apache.tomcat.jdbc.pool.interceptor.StatementFinalizer"
   jmxEnabled="true"
   logAbandoned="true"
   maxActive="50"
   maxWait="10000"
   minEvictableIdleTimeMillis="30000"
   minIdle="10"
   name="jdbc/ProductDS"
   password="nbuser"
   removeAbandoned="true"
   removeAbandonedTimeout="60"
   testOnBorrow="true"
   testOnReturn="false"
   testWhileIdle="true"
   timeBetweenEvictionRunsMillis="30000"
   type="javax.sql.DataSource"
   url="jdbc:mysql://localhost:3306/products"
   username="root"
   validationInterval="30000"
   validationQuery="SELECT 1"/>
```

server.xml du serveur Tomcat 7

Options supplémentaires disponibles :

maxIdle, minIdle, validatorClassName, numTestsPerEvictionRun,
accessToUnderlyingConnectionAllowed, connectionProperties,
poolPreparedStatements, maxOpenPreparedStatements
initSQL, fairQueue, abandonWhenPercentageFull, maxAge, useEquals,
suspectTimeout, alternateUsernameAllowed, dataSource, dataSourceJNDI

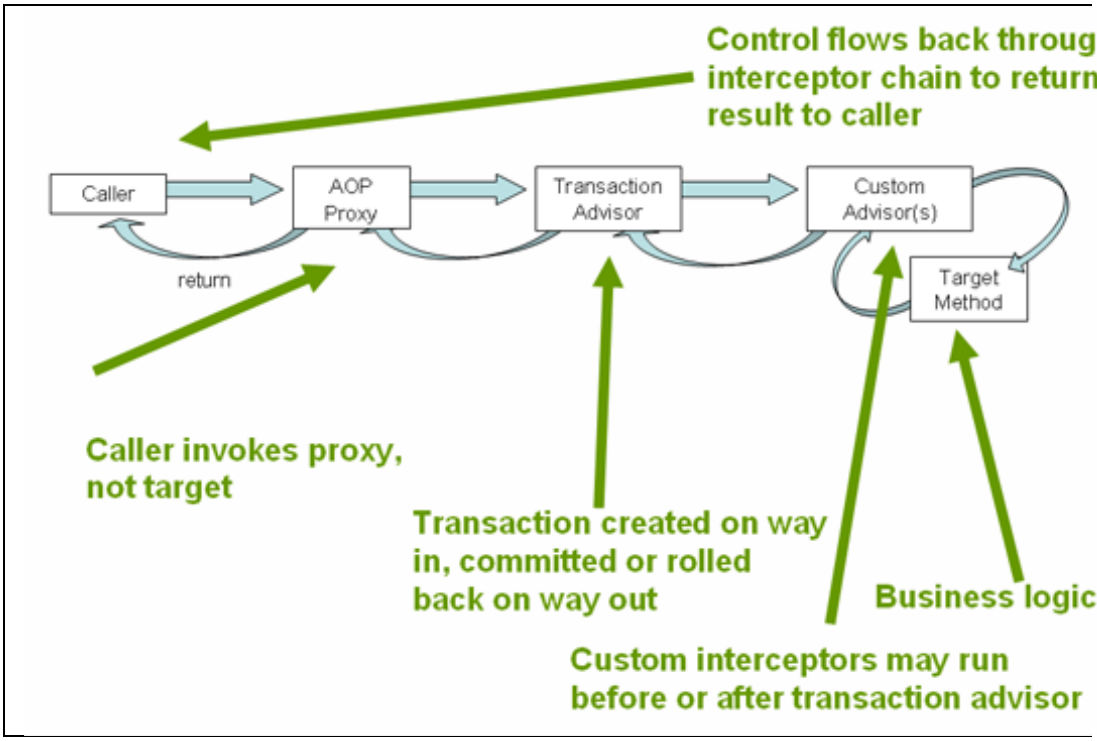
- JTA (Java Transaction API) est le standard J2EE associé aux mécanismes « Transactionnels ».

  Spring offre en ensemble cohérent de « best practice » et d'outils pour mettre en place JTA dans un environnement JPA ; pour ce faire, Spring Transaction propose deux modes de fonctionnement :

  - Les Transaction dites « *programmatives* », qui imposent l'implémentation du corps de transaction à l'intérieur du code des DAO, ou bien par un mécanisme de *callback* (CF TP précédent), en encapsulant les appels de méthodes de la transaction dans une inner classe.
    Cette partie transverse venant indéniablement polluer le fonctionnement des classes, les transactions « programmatives » sont à délaisser au profit de celles « déclaratives ».
  - Les Transaction dites « *déclaratives* », qui utilisant la technologie *AOP*, « *Programmation Orienté Aspect* », et l'@nnotations *@Transactionnal* permettant de déclarer les méthodes/classes sujettes à être remaniées par Spring Transaction lors de leurs appels de services.

    L'utilisation de l'AOP est parfaitement adaptée pour ce cas d'utilisation, et est recommandée puisque simplifiant très largement l'implémentation et le fonctionnement du code.

    Elle est activé par déclaration dans la configuration XML de la ligne :
    <tx:annotation-driven transaction-manager="transactionManager"/>



Mécanisme d'Aspect *@Transactionnal* mise en place par Spring

http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/transaction.html

- De la même façon que Spring fournit des *Template* et des *DAOSupport* pour un nombre important de solutions (JDBC, Hibernate, JDO, JPA, ...) on retrouve cette même modularité « *out-of-ze-box* » pour les *TransactionManager*, 3ème éléments fondamental et constitutif des mécanismes de DAO au sein du Framework Spring.

  CF TP : dans lequel on utilise un
  org.springframework.orm.hibernate4.HibernateTransactionManager
  référencie une *org.springframework.orm.hibernate4.LocalSessionFactoryBean*

  org.springframework.orm.jpa.JpaTransactionManager
  référencie
  org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean
  (ce dernier déléguant à Hibernate, de par sa propriété« *persistenceProviderClass* »
  ayant pour valeur « *org.hibernate.ejb.HibernatePersistence »).*

- L'implémentation de JTA dans Spring met à disposition 5
  propriétés fondamentales quant au fonctionnement optimal des Transactions :

  - Read-Only : true/false (défaut).
  - Time-Out : none (défaut), sinon une valeur en millisecondes.
  - Rollback Rules : Permet de spécifier de façon fine les règles de rollback en fonction du type d'Exception qui se produit lors du traitement. (*(no)rollbackFor(ClassName)*)
  - Isolation :
    - ISOLATION_DEFAULT (défaut)
      Se réfère au niveau d'isolation du SGBD.
    - ISOLATION_READ_UNCOMMITED
      Autorise la lecture d'entités non rafraîchies par l'*EntityManager,*
      lecture sale/« dirty read ».
    - ISOLATION_READ_COMMITED
      N'Autorise pas la lecture d'entités non rafraîchies, pas de lecture sale/
      « dirty read ».
  - Propagation :
    - PROPAGATION_REQUIRED (défaut)
      La méthode courante requière nécessairement une Transaction pour être exécutée, elle peut cependant se joindre à une transaction déjà existante.
    - PROPAGATION_NEW
      Une nouvelle Transaction est explicitement requise et créée.
    - PROPAGATION_SUPPORTS
      La méthode courante ne requière pas de Transaction, mais s'il en existe une elle peut alors la rejoindre.

2 Contenu du TP

- A l'instar du TP précédent qui proposait 3 solutions de DAO, on migre ici la solution Hibernate vers la couche d'abstraction JPA, ce qui permet de faire un comparatif entre les 2 frameworks, notamment sur les parties :
    - *Query* (déclarées en en-tête de classes par JPA)
    - L'utilisation de l'EntityManager
    - Les *Criteria*

- La configuration des politiques de mapping des entités du modèle, initialement déclarées en fichier hbm (Hibernate Mapping) est ici migrée selon la norme JPA en intégrant dans les classes du modèle des @nnotations à deux niveaux de définition :
    - Classe : *@Entity, @Table, @NamedQueries,*
    - Propriétés/Relations : @Column, @ManyToOne, ...

  Pour un détail exhaustif des @nnotations JPA d'Entity, se référer au document « JPA Cheat Sheet » disponible sur le site www.shortcircuit.fr.

- La configuration XML pour la configuration JNDI du Tomcat7 ConnectionPool, déclarant la *DataSource.*