Spring AOP Transaction Management In Hibernate

A database transaction is a sequence of actions that are treated as a single unit of work. These actions should either complete entirely or take no effect at all. Transaction management is an important part of and RDBMS oriented enterprise applications to ensure data integrity and consistency. The concept of transactions can be described with following four key properties described as **ACID**:

1. **Atomicity:** A transaction should be treated as a single unit of operation which means either the entire sequence of operations is successful or unsuccessful.
2. **Consistency:** This represents the consistency of the referential integrity of the database, unique primary keys in tables etc.
3. **Isolation:** There may be many transactions processing with the same data set at the same time, each transaction should be isolated from others to prevent data corruption.
4. **Durability:** Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.

Transaction management is required to ensure the data integrity and consistency in database. Spring's AOP technique is allow developers to manage the transaction declarative.

Here's an example to show how to manage the Hibernate transaction with Spring AOP.

```
//this method need to be transactional
public void save(Product product, int qoh){

        productDao.save(product);
        System.out.println("Product Inserted");

        ProductQoh productQoh = new ProductQoh();
        productQoh.setProductId(product.getProductId());
        productQoh.setQty(qoh);

        productQohBo.save(productQoh);
        System.out.println("ProductQoh Inserted");
}
```

Assume the **save()** does not has the transactional feature, if an Exception throw by **productQohBo.save()**, you will insert a record into `**product**` table only, no record will be insert into the `**productQoh**` table. This is a serious problem and break the data consistency in your database.

Transaction Management :

Declared a 'TransactionInterceptor' bean, and a 'HibernateTransactionManager' for the Hibernate transaction, and passing the necessary property.

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="transactionInterceptor"
       class="org.springframework.transaction.interceptor.TransactionInterceptor">
        <property name="transactionManager" ref="transactionManager" />
        <property name="transactionAttributes">
           <props>
                <prop key="save">PROPAGATION_REQUIRED</prop>
           </props>
        </property>
    </bean>

    <bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
          <property name="dataSource" ref="dataSource" />
          <property name="sessionFactory" ref="sessionFactory" />
    </bean>

</beans>
```

```
<!-- Transaction Manager -->
<bean id="transactionManager"
    class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="dataSource" ref="cypersDataSource" />
    <property name="sessionFactory" ref="mySessionFactory" />
</bean>

<bean id="transactionInterceptor"
    class="org.springframework.transaction.interceptor.TransactionInterceptor">
    <property name="transactionManager" ref="transactionManager" />
    <property name="transactionAttributeSource">
        <value>
            com.ignis.cypers.service.adtr.IPatientService.*=PROPAGATION_REQUIRED
            com.ignis.cypers.service.admin.IStaffService.*=PROPAGATION_REQUIRED
            com.ignis.cypers.service.admin.IGeneralService.*=PROPAGATION_REQUIRED
            com.ignis.cypers.service.admin.IWardService.*=PROPAGATION_REQUIRED
            com.ignis.cypers.service.masterdata.IMasterDataRepositoryService.*=PROPAGATION_REQUIRED
            com.ignis.cypers.service.dis.IDISService.*=PROPAGATION_REQUIRED
            com.ignis.cypers.service.cms.InternalCareProviderService.*=PROPAGATION_REQUIRED
            com.ignis.cypers.service.cms.ICMSService.*=PROPAGATION_REQUIRED
            com.ignis.cypers.service.admin.IAdminService.*=PROPAGATION_REQUIRED
            com.ignis.cypers.service.crs.ICRSService.*=PROPAGATION_REQUIRED
        </value>
    </property>
</bean>
```

Transaction Attributes :

In transaction interceptor, you have to define which transaction's attributes '**propagation behavior**' should be use. It means if a transactional '**ProductBoImpl.save()**' method is called another method '**productQohBo.save()**', how the transaction should be propagated? Should it continue to run within the existing transaction? or start a new transaction for its own.

There are 7 types of propagation supported by Spring :

- **PROPAGATION_REQUIRED** – Support a current transaction; create a new one if none exists.
- **PROPAGATION_SUPPORTS** – Support a current transaction; execute non-transactionally if none exists.
- **PROPAGATION_MANDATORY** – Support a current transaction; throw an exception if no current transaction exists.
- **PROPAGATION_REQUIRES_NEW** – Create a new transaction, suspending the current transaction if one exists.
- **PROPAGATION_NOT_SUPPORTED** – Do not support a current transaction; rather always execute non-transactionally.
- **PROPAGATION_NEVER** – Do not support a current transaction; throw an exception if a current transaction exists.
- **PROPAGATION_NESTED** – Execute within a nested transaction if a current transaction exists, behave like PROPAGATION_REQUIRED else.

In most cases, you may just need to use the PROPAGATION_REQUIRED.

In addition, you have to define the method to support this transaction attributes as well. The method name is supported wild card format, a **save*** will match all method name start with save(…).

Transaction Manager:

In Hibernate transaction, you need to use **HibernateTransactionManager**. If you only deal with pure JDBC, use**DataSourceTransactionManager**; while JTA, use **JtaTransactionManager**.

Proxy Factory Bean:

Create a new proxy factory bean for **ProductBo**, and set the '**interceptorNames**' property.

```
<!-- Product business object -->
<bean id="productBo" class="com.mkyong.product.bo.impl.ProductBoImpl" >
     <property name="productDao" ref="productDao" />
     <property name="productQohBo" ref="productQohBo" />
</bean>

<!-- Product Data Access Object -->
<bean id="productDao" class="com.mkyong.product.dao.impl.ProductDaoImpl" >
     <property name="sessionFactory" ref="sessionFactory"></property>
</bean>

<bean id="productBoProxy"
     class="org.springframework.aop.framework.ProxyFactoryBean">
     <property name="target" ref="productBo" />
     <property name="interceptorNames">
             <list>
                     <value>transactionInterceptor</value>
             </list>
     </property>
</bean>
```

Get your proxy bean `**productBoProxy**`, and your **save()** method is support transactional now, any exceptions inside**productBo.save()** method will cause the whole transaction to rollback, no data will be insert into the database.

```
<!-- Service -->
<bean id="patientService" class="com.ignis.cypers.service.adtr.PatientService" ></bean>


<bean id="myPatientService" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="target" ref="patientService" />
    <property name="interceptorNames">
        <list>
            <value>transactionInterceptor</value>
        </list>
    </property>
</bean>

<bean id="adtrServiceFactory" class="com.ignis.cypers.service.adtr.ADTRServiceFactory">
    <property name="patientService" ref="myPatientService"></property>

</bean>
```