[spring事务管理总结](http://blog.csdn.net/ocean1010/article/details/6745316)

标签： [spring](http://www.csdn.net/tag/spring)[service](http://www.csdn.net/tag/service)[数据库](http://www.csdn.net/tag/%e6%95%b0%e6%8d%ae%e5%ba%93)[session](http://www.csdn.net/tag/session)[nested](http://www.csdn.net/tag/nested)[exception](http://www.csdn.net/tag/exception)

2011-09-03 14:12 16975人阅读 [评论](http://blog.csdn.net/ocean1010/article/details/6745316#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ocean1010/article/details/6745316#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Spring（1） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

项目中一直用spring，事务管理这块还是遇到了挺多问题。下面把这些问题总结一下，以供大家参考和讨论。下面先提出这些问题  
  
问题一：spring中的声明式事务很方便，但有时候还是不能很好的满足需求。比如：项目中一些业务数据要从Excel中导入，一次导入可能上千条，每一条数据又涉及多个表（这几个表的数据需要在一个事务中）。这种情况可能声明式事务就不好办了。  
问题二：“PROPAGATION\_REQUIRED,readOnly”，只读事务和“PROPAGATION\_REQUIRED”区别在哪里？  
问题三：事务放在service层中，service中这个类的方法之间的调用，事务是什么样的？service层调用另外一个service层的类的方法，事务又是怎样的？  
问题四：编程式事务中，多次提交报错：Transaction is already completed do not call commit or rollback more than once per transaction，如何解决？  
首先来看一下spring中事务的基本知识  
事务的属性   
(1) 传播行为   
PROPAGATION\_MANDATORY: 方法必须在一个现存的事务中进行，否则丢出异常   
PROPAGATION\_NESTED: 在一个嵌入的事务中进行，如果不是，则同PROPAGATION\_REQUIRED   
PROPAGATION\_NEVER: 指出不应在事务中进行，如果有就丢出异常   
PROPAGATION\_NOT\_SUPPORTED: 指出不应在事务中进行，如果有就暂停现存的事务   
PROPAGATION\_REQUIRED: 在当前的事务中进行，如果没有就建立一个新的事务   
PROPAGATION\_REQUIRES\_NEW: 建立一个新的事务，如果现存一个事务就暂停它   
PROPAGATION\_SUPPORTS: 支持现在的事务，如果没有就以非事务的方式执行   
(2) 隔离层级   
ISOLATION\_DEFAULT: 使用底层数据库预设的隔离层级   
ISOLATION\_READ\_COMMITTED: 允许事务读取其他并行的事务已经送出（Commit）的数据字段，可以防止Dirty read问题   
ISOLATION\_READ\_UNCOMMITTED: 允许事务读取其他并行的事务还没送出的数据，会发生Dirty、Nonrepeatable、Phantom read等问题   
ISOLATION\_REPEATABLE\_READ: 要求多次读取的数据必须相同，除非事务本身更新数据，可防止Dirty、Nonrepeatable read问题   
ISOLATION\_SERIALIZABLE: 完整的隔离层级，可防止Dirty、Nonrepeatable、Phantom read等问题，会锁定对应的数据表格，因而有效率问题   
[具体知识请看这篇文章](http://blog.csdn.net/ocean1010/article/details/6548771)，[进一步了解spring事务请看这篇文章](http://blog.csdn.net/turkeyzhou/article/details/5202771)。

(3) 只读提示（Read-only hints）   
如果事务只进行读取的动作，则可以利用底层数据库在只读操作时发生的一些最佳化动作，由于这个动作利用到数据库在只读的事务操作最佳化，因而必须在事务中才有效，也就是说要搭配传播行为PROPAGATION\_REQUIRED、PROPAGATION\_REQUIRES\_NEW、PROPAGATION\_NESTED来设置。   
(4)事务超时期间（The transaction timeout period）   
有的事务操作可能延续很长一段的时间，事务本身可能关联到数据表的锁定，因而长时间的事务操作会有效率上的问题，对于过长的事务操作，考虑Roll back事务并要求重新操作，而不是无限时的等待事务完成。   
可以设置事务超时期间，计时是从事务开始时，所以这个设置必须搭配传播行为PROPAGATION\_REQUIRED、PROPAGATION\_REQUIRES\_NEW、PROPAGATION\_NESTED来设置。   
事务的超时属性以timeout\_为前缀和一个整型数字定义，例如： <prop key="query\*">PROPAGATION\_REGUIRED,timeout\_5,readOnly</prop>   
Spring中对异常的回滚，默认是在[抛出运行时异常(RuntimeException)](http://blog.csdn.net/ocean1010/article/details/6114348)时才回滚，对非运行时异常不回滚。如果使用-Exception,意思是对所有的异常异常都回滚。Exception前面加上 "-" 时，表示发生指定异常时撤消操作(rollback)，如果前面加上 "+"，表示发生异常时立即提交（commit）。   
要想用Spring的事务管理机制，就需要把数据库的连接交给Spring来管理，（JDBC,SESSION道理一样），如果使用Hibernate框架，要把Session交给Spring管理。在整个Service方法调用中，虽然Sevice调用了多个Dao，但是整个过程中Session只有一个。也就是说你对数据库的DML操作，都会先保存在这个Session中，包括update,insert,delete。当发生异常（这个异常可以是数据库的，也可以是程序的），Spring会把这个Session中对应的DML操作回滚。  
  
问题一解决的方法，显然声明式事务是不好满足这种需求的，那就采用编程式事务。  
spring参考文档中说最好用TransactionTemplate，而不推荐使用PlatformTransactionManager，而我认为恰恰相反，实际上TransactionTemplate还是不能很满足我们的需求，使用TransactionTemplate时当再次进行事务提交时就报Transaction is already completed do not call commit or rollback more than once per transaction。好像使用TransactionTemplate不能进行提交多次，当然采用PlatformTransactionManager是可以的（TransactionTemplate实际上提供了编程式事务的模板，TransactionTemplate处理事务方便，但可能不能满足需求）。下面贴出使用PlatformTransactionManager的代码  
第一：关于数据源和事务物理器的配置文件(TransactionTemplate也贴上了)

**[html]** [view plain](http://blog.csdn.net/ocean1010/article/details/6745316) [copy](http://blog.csdn.net/ocean1010/article/details/6745316)

1. **<bean** id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"**>**
2. **<property** name="driverClassName"**>**
3. **<value>**com.mysql.jdbc.Driver**</value>**
4. **</property>**
5. **<property** name="url"**>**
6. **<value>**jdbc:mysql://127.0.0.1:3306/db?useUnicode=true&characterEncoding=utf-8**</value>**
7. **</property>**
8. **<property** name="username"**>**
9. **<value>**root**</value>**
10. **</property>**
11. **<property** name="password"**>**
12. **<value>**root**</value>**
13. **</property>**
14. **</bean>**
15. **<bean** id="transactionManager"
16. class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>**
17. **<property** name="dataSource" ref="dataSource" **/>**
18. **</bean>**
19. **<bean** id="transactionTemplate"
20. class="org.springframework.transaction.support.TransactionTemplate"**>**
21. **<property** name="transactionManager"**>**
22. **<ref** bean="transactionManager" **/>**
23. **</property>**
24. **</bean>**

第二：使用PlatformTransactionManager进行多次提交的service类代码片断：

**[java]** [view plain](http://blog.csdn.net/ocean1010/article/details/6745316) [copy](http://blog.csdn.net/ocean1010/article/details/6745316)

1. //采用注入的方式得到PlatformTransactionManager
2. **private** PlatformTransactionManager platformTransactionManager=**null**;
4. **public** PlatformTransactionManager getPlatformTransactionManager()
5. {
6. **return** platformTransactionManager;
7. }
9. **public** **void** setPlatformTransactionManager(
10. PlatformTransactionManager platformTransactionManager)
11. {
12. **this**.platformTransactionManager = platformTransactionManager;
13. }
14. //循环提交示例
15. **public** **void** drExcel()
16. {
17. TransactionDefinition definition = **new** DefaultTransactionDefinition(TransactionDefinition.PROPAGATION\_REQUIRED);
18. TransactionStatus status=**null**;
19. **for**(**int** i = 0;i<10;i++)
20. {
21. **try**
22. {
23. //重置事务状态。没这句话会报错。这就是问题四的解决方法
24. status=platformTransactionManager.getTransaction(definition);
25. //以下代码是操作数据
26. //......
27. //提交
28. platformTransactionManager.commit(status);
29. }**catch**(Exception e)
30. {
31. e.printStackTrace();
32. platformTransactionManager.rollback(status);
33. }
35. }
36. }

下面贴出transactionTemplate使用片断，以供以后用到时参考：

**[java]** [view plain](http://blog.csdn.net/ocean1010/article/details/6745316) [copy](http://blog.csdn.net/ocean1010/article/details/6745316)

1. **public** **void** createPerson(**final** Person person) {
2. transactionTemplate.setPropagationBehavior(TransactionDefinition.PROPAGATION\_REQUIRED);
3. transactionTemplate.execute(**new** TransactionCallbackWithoutResult(){    // 使用无返回值的事务回调接口
4. @Override
5. **protected** **void** doInTransactionWithoutResult(TransactionStatus arg0) {
6. getHibernateTemplate().save(person);
7. }
8. });
9. }
11. **public** Person queryOnePerson(**final** String hql) {
12. transactionTemplate.setPropagationBehavior(TransactionDefinition.PROPAGATION\_REQUIRED);
13. **return** (Person)transactionTemplate.execute(**new** TransactionCallback(){     // 使用带返回值的事务回调接口
15. **public** Object doInTransaction(TransactionStatus arg0) {
16. **return** getHibernateTemplate().find(hql).get(0);
17. }
18. });
19. }

以上问题一和问题四基本上解决。  
再看问题二，我在oracle数据库试验了下，“PROPAGATION\_REQUIRED,readOnly”下进行事务操作是可以的，和“PROPAGATION\_REQUIRED”没有区别，可能是要告诉数据库我只进行只读操作，数据库可以进行对sql进行一些优化吧？如果进行insert update delete操作也还是放在一个事务中进行的。我认为对于不需要事务的操作，应该配置成“PROPAGATION\_SUPPORTS”还是更合理一些。有待大家讨论。  
问题三，service中这个类的方法之间的调用，你比如我现在事务配置成如下所示:

**[html]** [view plain](http://blog.csdn.net/ocean1010/article/details/6745316) [copy](http://blog.csdn.net/ocean1010/article/details/6745316)

1. **<bean** id="baseTransactionProxy"
2. class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"
3. abstract="true"**>**
4. **<property** name="transactionManager"**>**
5. **<ref** bean="transactionManager" **/>**
6. **</property>**
7. **<property** name="transactionAttributes"**>**
8. **<props>**
9. **<prop** key="insert\*"**>**PROPAGATION\_REQUIRED**</prop>**
10. **<prop** key="update\*"**>**PROPAGATION\_REQUIRED**</prop>**
11. **<prop** key="delete\*"**>**PROPAGATION\_REQUIRED**</prop>**
12. **<prop** key="\*"**>**PROPAGATION\_SUPPORTS**</prop>**
13. **</props>**
14. **</property>**
15. **</bean>**

某service1类中有方法两个方法importData（有一个对数据库操A）、insertData（有一个对数据库的操作B）方法。  
试验一importData方法中调用了insertData的方法  
试验结果：action中调用service1中的importData方法，这时候是没有事务的。  
试验二insertData调用了importData方法  
试验结果：action中调用service1中的insertData方法是有事务的。  
这就说明service方法中事务是根据该service层中最开始方法的事务属性来的。  
那么service类之间方法调用会是什么样的？我想也应该是这样的。