分布式事务操作之Spring+JTA - 勇者归来

什么是分布式事务？在网上找了一段比较容易理解的"定义".

分布式事务是指事务的参与者、支持事务的服务器、资源管理器以及事务管理器分别位于分布系统的不同节点之上，在两个或多个网络计算机资源上访问并且更新数据，将两个或多个网络计算机的数据进行的多次操作作为一个整体进行处理。如不同银行账户之间的转账。

对于在项目中接触到JTA，大部分的原因是因为在项目中需要操作多个数据库，同时，可以保证操作的原子性，保证对多个数据库的操作一致性。

在正式的项目中应该用springMVC（struts）+spring+hibernate（jpa）+jta，目前，先用spring+jta来完成基本的测试框架。下面我们看看代码

applicationContext-jta.xml

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <beans xmlns="http://www.springframework.org/schema/beans"
 3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
 4     xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
 5
xsi:schemaLocation="http://www.springframework.org/schema/beans
 6            http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd
 7            http://www.springframework.org/schema/context
 8
http://www.springframework.org/schema/context/spring-context-
2.5.xsd
 9            http://www.springframework.org/schema/aop
10            http://www.springframework.org/schema/aop/spring-
aop-2.5.xsd
11            http://www.springframework.org/schema/tx
12            http://www.springframework.org/schema/tx/spring-tx-
```

```
2.5.xsd">
13
14     <!-- jotm 本地实例 -->
15     <bean id="jotm"
class="org.springframework.transaction.jta.JotmFactoryBean" />
16
17
18     <!-- JTA事务管理器 -->
19     <bean id="txManager"
20
class="org.springframework.transaction.jta.JtaTransactionManager"
>
21         <property name="userTransaction" ref="jotm">
</property>
22     </bean>
23
24     <!-- XAPool配置，内部包含了一个XA数据源，对应sshdb数据库 -->
25     <bean id="db1"
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource"
26         destroy-method="shutdown">
27         <property name="dataSource">
28             <!-- 内部XA数据源 -->
29             <bean
class="org.enhydra.jdbc.standard.StandardXADataSource"
30                 destroy-method="shutdown">
31                 <property name="transactionManager" ref="jotm"
/>
32                 <property name="driverName"
value="com.mysql.jdbc.Driver" />
33                 <property name="url"
34
value="jdbc:mysql://192.168.1.28:3306/sshdb?
useUnicode=true&amp;characterEncoding=UTF-8" />
35             </bean>
36         </property>
37         <property name="user" value="root" />
38         <property name="password" value="123456" />
39     </bean>
```

```
40
41     <!-- 另一个XAPool配置，内部包含另一个XA数据源，对应babasport数据
库 -->
42     <bean id="db2"
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource"
43         destroy-method="shutdown">
44         <property name="dataSource">
45             <bean
class="org.enhydra.jdbc.standard.StandardXADataSource"
46                 destroy-method="shutdown">
47                 <property name="transactionManager" ref="jotm"
/>
48                 <property name="driverName"
value="com.mysql.jdbc.Driver" />
49                 <property name="url"
50
value="jdbc:mysql://192.168.1.28:3306/babasport?
useUnicode=true&amp;characterEncoding=UTF-8" />
51             </bean>
52         </property>
53         <property name="user" value="root" />
54         <property name="password" value="123456" />
55     </bean>
56
57     <!-- 配置访问sshdb数据源的Spring JDBC模板 -->
58     <bean id="sshdbTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
59         <property name="dataSource" ref="db1"></property>
60     </bean>
61
62     <!-- 配置访问babasport数据源的Spring JDBC模板 -->
63     <bean id="babasportTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
64         <property name="dataSource" ref="db2"></property>
65     </bean>
66 </beans>
```

小注一下：spring-tx-3.2.4.jar里面竟然没有org.springframework.transaction.jta.JotmFactoryBean类，如果你可以选择spring-tx-2.5.6.jar,或者自己建立一下这个类。

接下来，看下dao层测试的代码

```
 1  1     @Resource(name = "txManager")
 2  2     private JtaTransactionManager txManager;
 3  9
 4 10     protected JdbcTemplate babasport_jdbcTemplate;
 5 11
 6 12     /**
 7 13      * sshdb sql jdbcTemplate
 8 14      */
 9 15     protected JdbcTemplate ssh_jdbcTemplate;
10 16
11 17     /**
12 18      * babasport sql jdbcTemplate
13 19      *
14 20      * @return
15 21      */
16 22     public JdbcTemplate getBabasport_jdbcTemplate() {
17 23         return babasport_jdbcTemplate;
18 24     }
19 25
20 26     public JdbcTemplate getSsh_jdbcTemplate() {
21 27         return ssh_jdbcTemplate;
22 28     }
23 29
24 30     @Resource(name = "babasportTemplate")
25 31     public void setBabasport_jdbcTemplate(JdbcTemplate
babasport_jdbcTemplate) {
26 32         this.babasport_jdbcTemplate =
babasport_jdbcTemplate;
27 33     }
28 34
29 35     @Resource(name = "sshdbTemplate")
```

```
30 36     public void setSsh_jdbcTemplate(JdbcTemplate
ssh_jdbcTemplate) {
31 37         this.ssh_jdbcTemplate = ssh_jdbcTemplate;
32 38     }
33 39
34 40     /**
35 41      * 同时修改两个数据库的表中内容
36 42      *
37 43      * @throws RollbackException
38 44      */
39 45     public void updateMultiple() {
40 46
41 47         if (null == this.txManager) {
42 48             System.out.println("txManager为空");
43 49             return;
44 50         }
45 51
46 52         UserTransaction userTx =
this.txManager.getUserTransaction();
47 53         if (null == userTx) {
48 54             System.out.println("userTx为空");
49 55             return;
50 56         }
51 57
52 58         try {
53 59
54 60             userTx.begin();
55 61
56 62             this.ssh_jdbcTemplate
57 63                     .execute("update wyuser set
password='wangyong1' where id=8");
58 64
59 65
60 66             this.babasport_jdbcTemplate
61 67                     .execute("update brand set
name='wangyong28' where code='14ac8d5b-d19c-40e9-97ea-
d82dfbcd84c6'");
62 68
```

```
63 69             userTx.commit();
64 70         } catch (Exception e) {
65 71             System.out.println("捕获到异常，进行回滚" +
e.getMessage());
66 72             e.printStackTrace();
67 73             try {
68 74                 userTx.rollback();
69 75             } catch (IllegalStateException e1) {
70 76                 System.out.println("IllegalStateException:"
+ e1.getMessage());
71 77             } catch (SecurityException e1) {
72 78                 System.out.println("SecurityException:" +
e1.getMessage());
73 79             } catch (SystemException e1) {
74 80                 System.out.println("SystemException:" +
e1.getMessage());
75 81             }
76 82             // System.out.println("sql语句操作失败");
77 83         }
78 84     }
```

如果，将后一条update语句故意写错，就会发现会执行rollback，同时，对上面一个语句的操作也不会生效。基本的简单框架就是这样。

其实，之前也测试了下spring+jpa+jta的框架模式，却发现，在建立model层实体类的时候会有问题，建立的entity类映射到所有的数据库中了，于是在jpa中利用属性<property name="packagesToScan" value="包名" />这种方式的确可以解决实体类entity的映射问题，不过貌似又出现其他问题，待研究.....