[Spring分布式事务实现](http://blog.csdn.net/ithomer/article/details/10859235)

 分布式事务是指操作多个数据库之间的事务，spring的org.springframework.transaction.jta.JtaTransactionManager，提供了分布式事务支持。如果使用WAS的JTA支持，把它的属性改为WebSphere对应的TransactionManager。   
       在tomcat下，是没有分布式事务的，不过可以借助于第三方软件jotm（Java Open Transaction Manager ）和AtomikosTransactionsEssentials实现，在spring中分布式事务是通过jta（jotm，atomikos）来进行实现。   
1、http://jotm.objectweb.org/   
2、http://www.atomikos.com/Main/TransactionsEssentials   
  
**一、使用JOTM例子**  
(1) Dao及实现

GenericDao接口：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ithomer/article/details/10859235)

1. **public** **interface** GenericDao {
2. **public** **int** save(String ds, String sql, Object[] obj) **throws** Exception;
3. **public** **int** findRowCount(String ds, String sql);
4. }

GenericDaoImpl 实现：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ithomer/article/details/10859235)

1. **public** **class** GenericDaoImpl **implements** GenericDao{
3. **private**  JdbcTemplate jdbcTemplateA;
4. **private**  JdbcTemplate jdbcTemplateB;
6. **public** **void** setJdbcTemplateA(JdbcTemplate jdbcTemplate) {
7. **this**.jdbcTemplateA = jdbcTemplate;
8. }
10. **public** **void** setJdbcTemplateB(JdbcTemplate jdbcTemplate) {
11. **this**.jdbcTemplateB = jdbcTemplate;
12. }
14. **public** **int** save(String ds, String sql, Object[] obj) **throws** Exception{
15. **if**(**null** == ds || "".equals(ds)) **return** -1;
16. **try**{
17. **if**(ds.equals("A")){
18. **return** **this**.jdbcTemplateA.update(sql, obj);
19. }**else**{
20. **return** **this**.jdbcTemplateB.update(sql, obj);
21. }
22. }**catch**(Exception e){
23. e.printStackTrace();
24. **throw** **new** Exception("执行" + ds + "数据库时失败！");
25. }
26. }
28. **public** **int** findRowCount(String ds, String sql) {
29. **if**(**null** == ds || "".equals(ds)) **return** -1;
31. **if**(ds.equals("A")){
32. **return** **this**.jdbcTemplateA.queryForInt(sql);
33. }**else**{
34. **return** **this**.jdbcTemplateB.queryForInt(sql);
35. }
36. }
37. }

(2) Service及实现

UserService 接口：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ithomer/article/details/10859235)

1. **public** **interface** UserService {
2. **public** **void** saveUser() **throws** Exception;
3. }

UserServiceImpl 实现：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ithomer/article/details/10859235)

1. **public** **class** UserServiceImpl **implements** UserService{
3. **private** GenericDao genericDao;
5. **public** **void** setGenericDao(GenericDao genericDao) {
6. **this**.genericDao = genericDao;
7. }
9. **public** **void** saveUser() **throws** Exception {
10. String userName = "user\_" + Math.round(Math.random()\*10000);
11. System.out.println(userName);
13. StringBuilder sql = **new** StringBuilder();
14. sql.append(" insert into t\_user(username, gender) values(?,?); ");
15. Object[] objs = **new** Object[]{userName,"1"};
17. genericDao.save("A", sql.toString(), objs);
19. sql.delete(0, sql.length());
20. sql.append(" insert into t\_user(name, sex) values(?,?); ");
21. objs = **new** Object[]{userName,"男的"};//值超出范围
22. genericDao.save("B", sql.toString(), objs);
23. }
24. }

(3) applicationContext-jotm.xml

**[java]** [view plaincopyprint?](http://blog.csdn.net/ithomer/article/details/10859235)

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"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:aop="http://www.springframework.org/schema/aop"
6. xmlns:tx="http://www.springframework.org/schema/tx"
7. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
8. http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd
9. http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
10. http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
12. <description>springJTA</description>
14. <!--指定Spring配置中用到的属性文件-->
15. <bean id="propertyConfig"
16. **class**="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
17. <property name="locations">
18. <list>
19. <value>classpath:jdbc.properties</value>
20. </list>
21. </property>
22. </bean>
24. <!-- JOTM实例 -->
25. <bean id="jotm" **class**="org.springframework.transaction.jta.JotmFactoryBean">
26. <property name="defaultTimeout" value="500000"/>
27. </bean>
29. <!-- JTA事务管理器 -->
30. <bean id="jtaTransactionManager" **class**="org.springframework.transaction.jta.JtaTransactionManager">
31. <property name="userTransaction" ref="jotm" />
32. </bean>
34. <!-- 数据源A -->
35. <bean id="dataSourceA" **class**="org.enhydra.jdbc.pool.StandardXAPoolDataSource" destroy-method="shutdown">
36. <property name="dataSource">
37. <bean **class**="org.enhydra.jdbc.standard.StandardXADataSource" destroy-method="shutdown">
38. <property name="transactionManager" ref="jotm"/>
39. <property name="driverName" value="${jdbc.driver}"/>
40. <property name="url" value="${jdbc.url}"/>
41. </bean>
42. </property>
43. <property name="user" value="${jdbc.username}"/>
44. <property name="password" value="${jdbc.password}"/>
45. </bean>
47. <!-- 数据源B -->
48. <bean id="dataSourceB" **class**="org.enhydra.jdbc.pool.StandardXAPoolDataSource" destroy-method="shutdown">
49. <property name="dataSource">
50. <bean **class**="org.enhydra.jdbc.standard.StandardXADataSource" destroy-method="shutdown">
51. <property name="transactionManager" ref="jotm"/>
52. <property name="driverName" value="${jdbc2.driver}"/>
53. <property name="url" value="${jdbc2.url}"/>
54. </bean>
55. </property>
56. <property name="user" value="${jdbc2.username}"/>
57. <property name="password" value="${jdbc2.password}"/>
58. </bean>
60. <bean id = "jdbcTemplateA"
61. **class** = "org.springframework.jdbc.core.JdbcTemplate">
62. <property name = "dataSource" ref="dataSourceA"/>
63. </bean>
65. <bean id = "jdbcTemplateB"
66. **class** = "org.springframework.jdbc.core.JdbcTemplate">
67. <property name = "dataSource" ref="dataSourceB"/>
68. </bean>
70. <!-- 事务切面配置 -->
71. <aop:config>
72. <aop:pointcut id="pointCut"
73. expression="execution(\* com.logcd.service..\*.\*(..))"/><!-- 包及其子包下的所有方法 -->
74. <aop:advisor pointcut-ref="pointCut" advice-ref="txAdvice"/>
76. <aop:advisor pointcut="execution(\* \*..common.service..\*.\*(..))" advice-ref="txAdvice"/>
77. </aop:config>
79. <!-- 通知配置 -->
80. <tx:advice id="txAdvice" transaction-manager="jtaTransactionManager">
81. <tx:attributes>
82. <tx:method name="delete\*" rollback-**for**="Exception"/>
83. <tx:method name="save\*" rollback-**for**="Exception"/>
84. <tx:method name="update\*" rollback-**for**="Exception"/>
85. <tx:method name="find\*" read-only="true" rollback-**for**="Exception"/>
86. </tx:attributes>
87. </tx:advice>
89. <bean id="genericDao"  **class**="com.logcd.dao.impl.GenericDaoImpl" autowire="byName"> </bean>
90. <bean id="userService"  **class**="com.logcd.service.impl.UserServiceImpl" autowire="byName"> </bean>
91. </beans>

(4) 测试

**[java]** [view plaincopyprint?](http://blog.csdn.net/ithomer/article/details/10859235)

1. **public** **class** TestUserService{
3. **private** **static** UserService userService;
5. @BeforeClass
6. **public** **static** **void** init(){
7. ApplicationContext app = **new** ClassPathXmlApplicationContext("applicationContext-jotm.xml");
8. userService = (UserService)app.getBean("userService");
9. }
11. @Test
12. **public** **void** save(){
13. System.out.println("begin...");
14. **try**{
15. userService.saveUser();
16. }**catch**(Exception e){
17. System.out.println(e.getMessage());
18. }
19. System.out.println("finish...");
20. }
21. }

二、关于使用atomikos实现   
(1) 数据源配置

**[java]** [view plaincopyprint?](http://blog.csdn.net/ithomer/article/details/10859235)

1. <bean id="dataSourceA" **class**="com.atomikos.jdbc.SimpleDataSourceBean" init-method="init" destroy-method="close">
2. <property name="uniqueResourceName">
3. <value>${datasource.uniqueResourceName}</value>
4. </property>
5. <property name="xaDataSourceClassName">
6. <value>${database.driver\_class}</value>
7. </property>
8. <property name="xaDataSourceProperties">
9. <value>URL=${database.url};user=${database.username};password=${database.password}</value>
10. </property>
11. <property name="exclusiveConnectionMode">
12. <value>${connection.exclusive.mode}</value>
13. </property>
14. <property name="connectionPoolSize">
15. <value>${connection.pool.size}</value>
16. </property>
17. <property name="connectionTimeout">
18. <value>${connection.timeout}</value>
19. </property>
20. <property name="validatingQuery">
21. <value>SELECT 1</value>
22. </property>
23. </bean>

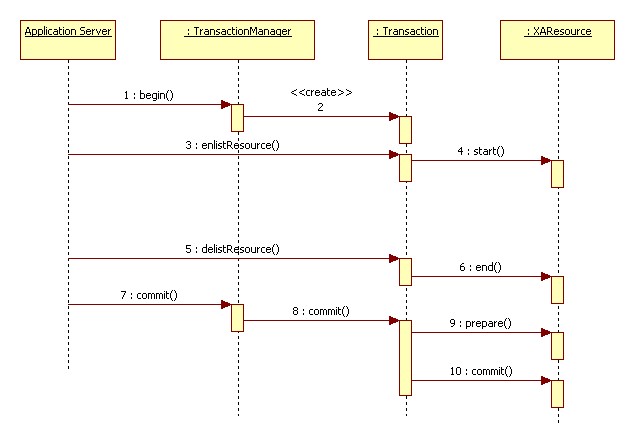
(2)、事务配置 

**[java]** [view plaincopyprint?](http://blog.csdn.net/ithomer/article/details/10859235)

1. <bean id="atomikosTransactionManager" **class**="com.atomikos.icatch.jta.UserTransactionManager"
2. init-method="init" destroy-method="close">
3. <property name="forceShutdown" value="true"/>
4. </bean>
6. <bean id="atomikosUserTransaction" **class**="com.atomikos.icatch.jta.UserTransactionImp">
7. <property name="transactionTimeout" value="${transaction.timeout}"/>
8. </bean>
10. <!-- JTA事务管理器 -->
11. <bean id="springTransactionManager" **class**="org.springframework.transaction.jta.JtaTransactionManager">
12. <property name="transactionManager" ref="atomikosTransactionManager"/>
13. <property name="userTransaction" ref="atomikosUserTransaction"/>
14. </bean>
16. <!-- 事务切面配置 -->
17. <aop:config>
18. <aop:pointcut id="serviceOperation"  expression="execution(\* \*..service\*..\*(..))"/>
19. <aop:advisor pointcut-ref="serviceOperation" advice-ref="txAdvice"/>
20. </aop:config>
22. <!-- 通知配置 -->
23. <tx:advice id="txAdvice" transaction-manager="springTransactionManager">
24. <tx:attributes>
25. <tx:method name="\*" rollback-**for**="Exception"/>
26. </tx:attributes>
27. </tx:advice>

**有关JTA**  
JTA全称为Java Transaction API，顾名思义JTA定义了一组统一的事务编程的接口，这些接口如下：  
   
**XAResource**  
XAResource接口是对实现了X/Open CAE规范的资源管理器 (Resource Manager，数据库就是典型的资源管理器) 的抽象，它由资源适配器 (Resource Apdater) 提供实现。XAResource是支持事务控制的核心。  
  
**Transaction**  
Transaction接口是一个事务实例的抽象，通过它可以控制事务内多个资源的提交或者回滚。二阶段提交过程也是由Transaction接口的实现者来完成的。

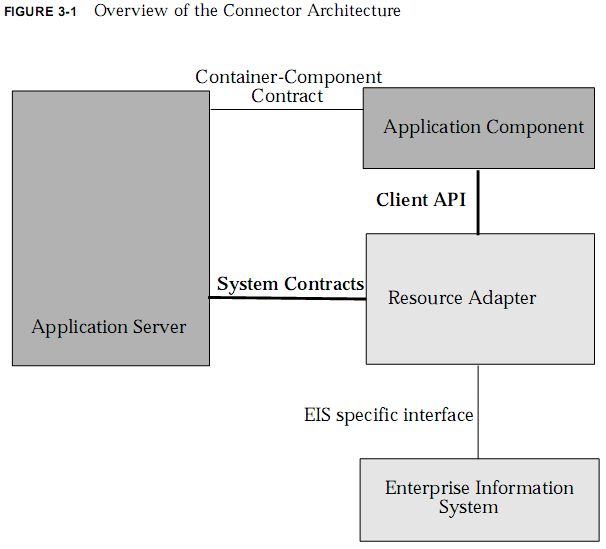
TransactionManager  
托管模式 (managed mode) 下，TransactionManager接口是被应用服务器调用，以控制事务的边界的。  
   
UserTransaction  
非托管模式 (non-managed mode) 下，应用程序可以通过UserTransaction接口控制事务的边界  
   
托管模式下的事务提交场景



**注意**：在上图中3和5的步骤之间省略了应用程序对资源的操作 (如CRUD)。另外，应用服务器什么时机 enlistResource，又是什么时候delistResource呢？这在后文中会解释。

**有关JCA**

下图为JCA的架构图



中间涉及元素说明如下：  
1）Enterprise Information System  
简称EIS，在JTA中它又被称为资源管理器。典型的EIS有数据库，事务处理系统(Transaction Processing System)，ERP系统。  
   
2）Resource Adapter  
资源适配器(Resource Adaper)是JCA的关键。要想把不同的EIS整合(或者连接)到J2EE运行环境中，就必须为每个EIS提供资源适配器，它会将将EIS适配为一个具备统一编程接口的资源 (Resource) 。这个统一编程接口就是上图中的System Contracts和Client API。下面的UML类图将完美诠释资源适配器。

3）Application Server  
应用服务器 (Application Server) 通过System Contracts来管理对EIS的安全、事务、连接等。典型的应用服务器有JBoss、JOnAS、Geronimo、GlassFish等。  
   
4）Application Component   
应用组件 (Application Component) ，它封装了应用业务逻辑，像对资源的访问和修改。典型的应用组件就是EJB。  
   
更多细节请参见：  
Sun Microsystems Inc.J2EE Connector Architecture 1.5

**参考推荐：**

[Spring分布式事务实现](http://log-cd.iteye.com/blog/807607)

[JTA与JCA分布式事务](http://zhongl.iteye.com/blog/317041)

[理解 JCA 事务](http://www.ibm.com/developerworks/cn/java/j-jca/)（IBM）

[Jencks实现Hibernate与Jackrabbit的分布式事务](http://zhongl.iteye.com/blog/315430)