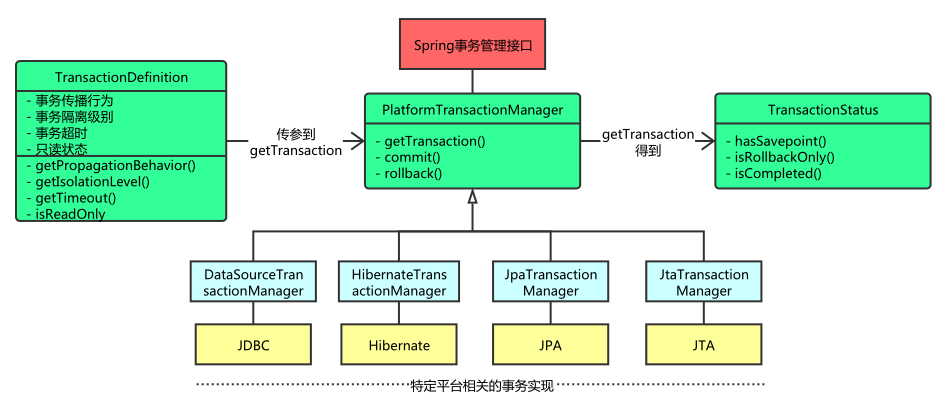


一、事务的作用

  将若干的数据库操作作为一个整体控制,一起成功或一起失败。  
  **原子性**：指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。  
  **一致性**：指事务前后数据的完整性必须保持一致。  
  **隔离性**：指多个用户并发访问数据库时，一个用户的事务不能被其他用户的事务所干扰，多个并发事务之间数据要相互隔离。  
  **持久性**：指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，即时数据库发生故障也不应该对其有任何影响。



二、Spring事务管理高层抽象主要包括3个接口

  --**Platform TransactionManager** 事务管理器（提交、回滚事务）  
     Spring为不同的持久化框架提供了不同的Platform TransactionManager接口实现。如：  
        使用Spring JDBC或iBatis进行持久化数据时使用DataSourceTransactionManager  
        使用Hibernate3.0版本进行持久化数据时使用HibernateTransactionManager

  --**TransactionDefinition** 事务定义信息（隔离、传播、超时、只读）  
        脏读：一个事务读取了另一个事务改写但还未提交的数据，如果这些数据被回滚，则读到的数据是无效的。  
        不可重复读：在同一事务中，多次读取同一数据返回的结果有所不同。  
        幻读：一个事务读取了几行记录后，另一个事务插入一些记录，幻读就发生了。再后来的查询中，第一个事务就会发现有些原来没有的记录。

事务隔离级别：（五种）

    DEFAULT--使用后端数据库默认的隔离级别（Spring中的选择项）

    READ\_UNCOMMITED--允许你读取还未提交的改变了的数据。可能导致脏、幻、不可重复读

    READ\_COMMITTED--允许在并发事务已经提交后读取。可防止脏读，但幻读和不可重复读仍可发生

    REPEATABLE\_READ--对相同字段的多次读取是一致的，除非数据被事务本身改变。可防止脏、不可重复读，但幻读仍可能发生

    SERIALIZABLE--完全服从ACID的隔离级别，确保不发生脏、幻、不可重复读。这在所有的隔离级别中是最慢的，它是典型的通过完全锁定在事务中涉及的数据表来完成的

**其中，MySQL默认采用REPEATABLE\_READ隔离级别；Oracle默认采用READ\_COMMITTED隔离级别**  
  
事务传播行为：（七种）

    REQUIRED--支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

    SUPPORTS--支持当前事务，如果当前没有事务，就以非事务方式执行。

    MANDATORY--支持当前事务，如果当前没有事务，就抛出异常。

    REQUIRES\_NEW--新建事务，如果当前存在事务，把当前事务挂起。

    NOT\_SUPPORTED--以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

    NEVER--以非事务方式执行，如果当前存在事务，则抛出异常。

    NESTED--如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则进行与REQUIRED类似的操作。拥有多个可以回滚的保存点，内部回滚不会对外部事务产生影响。只对DataSourceTransactionManager有效

  --**TransactionStatus** 事务具体运行状态

三、Spring提供了以下方法控制事务

  a.编程式事务管理(基于Java编程控制，很少使用)--见demo1包  
       利用TransactionTemplate将多个DAO操作封装起来  
  \*b.声明式事务管理(基于Spring的AOP配置控制)  
       -基于TransactionProxyFactoryBean的方式.(很少使用)--见demo2包  
            需要为每个进行事务管理的类,配置一个TransactionProxyFactoryBean进行增强.  
       -**基于XML配置(经常使用)**--见demo3包  
            一旦配置好之后,类上不需要添加任何东西。  
            如果Action作为目标对象切入事务，需要在<aop:config>元素里添加proxy-target-class="true"属性。原因是通知Spring框架采用CGLIB技术生成具有事务管理功能的Action类。  
       -**基于注解(配置简单，经常使用)**--见demo4包  
            在applicationContext.xml中开启事务注解配置。(applicationContext.xml中只需定义Bean并追加以下元素)  
<bean id="txManager" class="...">  
  <property name="sessionFactory">  
  </property>  
<tx:annotation-driven transaction-manager="txManager"/>

            在目标组件类中使用@Transactional，该标记可定义在类前或方法前。

* a.编程式事务管理(基于Java编程控制，很少使用)
* b.声明式事务管理(基于Spring的AOP配置控制)

-基于TransactionProxyFactoryBean的方式.(很少使用)

-**基于XML配置(经常使用)**

-**基于注解(配置简单，经常使用)**

四、示例（银行转账）

 --编程式

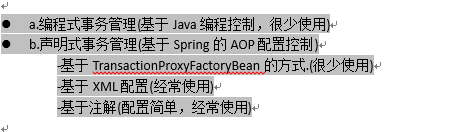
1. /\*\*
2. \* @Description:转账案例的DAO层接口
3. \*
4. \*/
5. **public** **interface** AccountDao {
6. /\*\*
7. \* @param out
8. \*            :转出账号
9. \* @param money
10. \*            :转账金额
11. \*/
12. **public** **void** outMoney(String out, Double money);
14. /\*\*
15. \*
16. \* @param in
17. \*            :转入账号
18. \* @param money
19. \*            :转账金额
20. \*/
21. **public** **void** inMoney(String in, Double money);
22. }
23. /\*\*
24. \* @Description:转账案例的DAO层实现类
25. \*/
26. **public** **class** AccountDaoImpl **extends** JdbcDaoSupport **implements** AccountDao {
27. /\*\*
28. \* @param out
29. \*            :转出账号
30. \* @param money
31. \*            :转账金额
32. \*/
33. @Override
34. **public** **void** outMoney(String out, Double money) {
35. String sql = "update account set money = money-? where name = ?";
36. **this**.getJdbcTemplate().update(sql, money, out);
37. }
38. /\*\*
39. \* @param in
40. \*            :转入账号
41. \* @param money
42. \*            :转账金额
43. \*/
44. @Override
45. **public** **void** inMoney(String in, Double money) {
46. String sql = "update account set money = money+? where name = ?";
47. **this**.getJdbcTemplate().update(sql, money, in);
48. }
49. }
50. /\*\*
51. \* @Description:转账案例的业务接口
52. \*
53. \*/
54. **public** **interface** AccountService {
55. /\*\*
56. \* @param out   :转出账号
57. \* @param in    :转入账号
58. \* @param money :转账金额
59. \*/
60. **public** **void** transfer(String out,String in,Double money);
61. }
62. /\*\*
63. \* @Description:转账案例的业务层实现类
64. \*/
65. **public** **class** AccountServiceImpl **implements** AccountService {
66. // 注入转账的DAO
67. **private** AccountDao accountDao;
69. // 注入事务管理的模板
70. **private** TransactionTemplate transactionTemplate;
72. /\*\*
73. \* @param out
74. \*            :转出账号
75. \* @param in
76. \*            :转入账号
77. \* @param money
78. \*            :转账金额
79. \*/
80. @Override
81. **public** **void** transfer(**final** String out, **final** String in, **final** Double money) {
83. // 未经事务控制的业务处理操作，如果过程中出异常，则导致前面的操作能完成，后面的不能，即转账成功但未收到转账款
84. // accountDao.outMoney(out, money);
85. // int i = 1/0;
86. // accountDao.inMoney(in, money);
88. transactionTemplate.execute(**new** TransactionCallbackWithoutResult() {
90. @Override
91. **protected** **void** doInTransactionWithoutResult(
92. TransactionStatus transactionStatus) {
93. accountDao.outMoney(out, money);
94. // int i = 1 / 0;//事务控制，即出现异常，该段内代码都执行失效
95. accountDao.inMoney(in, money);
96. }
97. });
98. }
100. **public** **void** setAccountDao(AccountDao accountDao) {
101. **this**.accountDao = accountDao;
102. }
104. **public** **void** setTransactionTemplate(TransactionTemplate transactionTemplate) {
105. **this**.transactionTemplate = transactionTemplate;
106. }
107. }

applicationContext1.xml

1. <!-- 引入外部的属性文件 -->
2. **<context:property-placeholder** location="classpath:jdbc.properties"**/>**
4. <!-- 配置c3p0连接池 -->
5. **<bean** id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"**>**
6. **<property** name="driverClass" value="${jdbc.driverClass}" **/>**
7. **<property** name="jdbcUrl" value="${jdbc.url}" **/>**
8. **<property** name="user" value="${jdbc.username}" **/>**
9. **<property** name="password" value="${jdbc.password}" **/>**
10. **</bean>**
12. <!-- 配置业务层类 -->
13. **<bean** id="accountService" class="com.zs.spring.demo1.AccountServiceImpl"**>**
14. **<property** name="accountDao" ref="accountDao" **/>**
15. <!-- 注入事务管理的模板 -->
16. **<property** name="transactionTemplate" ref="transactionTemplate" **/>**
17. **</bean>**
19. <!-- 配置DAO类(简化，会自动配置JdbcTemplate) -->
20. **<bean** id="accountDao" class="com.zs.spring.demo1.AccountDaoImpl"**>**
21. **<property** name="dataSource" ref="dataSource" **/>**
22. **</bean>**
24. <!-- 配置DAO类(未简化) -->
25. <!-- **<bean** id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate"**>**
26. **<property** name="dataSource" ref="dataSource" **/>**
27. **</bean>**
28. **<bean** id="accountDao" class="com.zs.spring.demo1.AccountDaoImpl"**>**
29. **<property** name="jdbcTemplate" ref="jdbcTemplate" **/>**
30. **</bean>** --**>**
32. <!-- ==================================1.编程式的事务管理=============================================== -->
33. <!-- 配置事务管理器 -->
34. **<bean** id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>**
35. **<property** name="dataSource" ref="dataSource" **/>**
36. **</bean>**
38. <!-- 配置事务管理的模板:Spring为了简化事务管理的代码而提供的类 -->
39. **<bean** id="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate"**>**
40. **<property** name="transactionManager" ref="transactionManager"**/>**
41. **</bean>**

测试：

1. @RunWith(SpringJUnit4ClassRunner.**class**)
2. @ContextConfiguration("classpath:applicationContext1.xml")
3. **public** **class** TransactionTest {
4. @Resource(name = "accountService")
5. **private** AccountService accountService;
7. @Test
8. **public** **void** demo1() {
9. accountService.transfer("aaa", "bbb", 200d);
10. }
11. }



   --基于TransactionProxyFactoryBean的方式

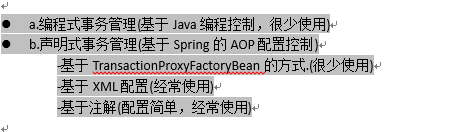
1. **public** **class** AccountServiceImpl **implements** AccountService {
2. // 注入转账的DAO
3. **private** AccountDao accountDao;
5. /\*\*
6. \* @param out
7. \*            :转出账号
8. \* @param in
9. \*            :转入账号
10. \* @param money
11. \*            :转账金额
12. \*/
13. @Override
14. **public** **void** transfer(String out, String in, Double money) {
15. accountDao.outMoney(out, money);
16. // int i = 1/0;
17. accountDao.inMoney(in, money);
18. }
20. **public** **void** setAccountDao(AccountDao accountDao) {
21. **this**.accountDao = accountDao;
22. }
23. }

applicationContext2.xml

1. <!-- 引入外部的属性文件 -->
2. **<context:property-placeholder** location="classpath:jdbc.properties"**/>**
4. <!-- 配置c3p0连接池 -->
5. **<bean** id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"**>**
6. **<property** name="driverClass" value="${jdbc.driverClass}" **/>**
7. **<property** name="jdbcUrl" value="${jdbc.url}" **/>**
8. **<property** name="user" value="${jdbc.username}" **/>**
9. **<property** name="password" value="${jdbc.password}" **/>**
10. **</bean>**
12. <!-- 配置业务层类 -->
13. **<bean** id="accountService" class="com.zs.spring.demo2.AccountServiceImpl"**>**
14. **<property** name="accountDao" ref="accountDao" **/>**
15. **</bean>**
17. <!-- 配置DAO类(简化，会自动配置JdbcTemplate) -->
18. **<bean** id="accountDao" class="com.zs.spring.demo2.AccountDaoImpl"**>**
19. **<property** name="dataSource" ref="dataSource" **/>**
20. **</bean>**
22. <!-- ==================================2.使用XML配置声明式的事务管理(原始方式)=============================================== -->
24. <!-- 配置事务管理器 -->
25. **<bean** id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>**
26. **<property** name="dataSource" ref="dataSource" **/>**
27. **</bean>**
29. <!-- 配置业务层的代理 -->
30. **<bean** id="accountServiceProxy" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"**>**
31. <!-- 配置目标对象 -->
32. **<property** name="target" ref="accountService" **/>**
33. <!-- 注入事务管理器 -->
34. **<property** name="transactionManager" ref="transactionManager"**></property>**
35. <!-- 注入事务的属性 -->
36. **<property** name="transactionAttributes"**>**
37. **<props>**
38. <!--
39. prop的格式:
40. \* PROPAGATION   :事务的传播行为
41. \* ISOTATION     :事务的隔离级别
42. \* readOnly      :只读
43. \* -EXCEPTION    :发生哪些异常回滚事务
44. \* +EXCEPTION    :发生哪些异常不回滚事务
45. --**>**
46. **<prop** key="transfer"**>**PROPAGATION\_REQUIRED**</prop>**
47. <!-- <prop key="transfer">PROPAGATION\_REQUIRED,readOnly</prop> -->
48. <!-- <prop key="transfer">PROPAGATION\_REQUIRED,+java.lang.ArithmeticException</prop> -->
49. **</props>**
50. **</property>**
51. **</bean>**

测试：

1. @RunWith(SpringJUnit4ClassRunner.**class**)
2. @ContextConfiguration("classpath:applicationContext2.xml")
3. **public** **class** TransactionTest {
4. /\*\*
5. \* 一定要注入代理类:因为代理类进行增强的操作
6. \*/
7. // @Resource(name="accountService")
8. @Resource(name = "accountServiceProxy")
9. **private** AccountService accountService;
11. @Test
12. **public** **void** demo1() {
13. accountService.transfer("aaa", "bbb", 200d);
14. }
15. }



 --基于XML配置

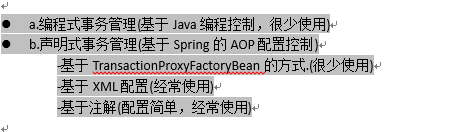
1. **public** **class** AccountServiceImpl **implements** AccountService {
2. // 注入转账的DAO
3. **private** AccountDao accountDao;
5. /\*\*
6. \* @param out
7. \*            :转出账号
8. \* @param in
9. \*            :转入账号
10. \* @param money
11. \*            :转账金额
12. \*/
13. @Override
14. **public** **void** transfer(String out, String in, Double money) {
15. accountDao.outMoney(out, money);
16. // int i = 1/0;
17. accountDao.inMoney(in, money);
19. }
21. **public** **void** setAccountDao(AccountDao accountDao) {
22. **this**.accountDao = accountDao;
23. }
24. }

applicationContext3.xml

1. <!-- 引入外部的属性文件 -->
2. **<context:property-placeholder** location="classpath:jdbc.properties"**/>**
4. <!-- 配置c3p0连接池 -->
5. **<bean** id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"**>**
6. **<property** name="driverClass" value="${jdbc.driverClass}" **/>**
7. **<property** name="jdbcUrl" value="${jdbc.url}" **/>**
8. **<property** name="user" value="${jdbc.username}" **/>**
9. **<property** name="password" value="${jdbc.password}" **/>**
10. **</bean>**
12. <!-- 配置业务层类 -->
13. **<bean** id="accountService" class="com.zs.spring.demo3.AccountServiceImpl"**>**
14. **<property** name="accountDao" ref="accountDao" **/>**
15. **</bean>**
17. <!-- 配置DAO类(简化，会自动配置JdbcTemplate) -->
18. **<bean** id="accountDao" class="com.zs.spring.demo3.AccountDaoImpl"**>**
19. **<property** name="dataSource" ref="dataSource" **/>**
20. **</bean>**
22. <!-- ==================================3.使用XML配置声明式的事务管理,基于tx/aop=============================================== -->
24. <!-- 配置事务管理器 -->
25. **<bean** id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>**
26. **<property** name="dataSource" ref="dataSource" **/>**
27. **</bean>**
29. <!-- 配置事务的通知 -->
30. **<tx:advice** id="txAdvice" transaction-manager="transactionManager"**>**
31. **<tx:attributes>**
32. <!--
33. propagation :事务传播行为
34. isolation   :事务的隔离级别
35. read-only   :只读
36. rollback-for:发生哪些异常回滚
37. no-rollback-for :发生哪些异常不回滚
38. timeout     :过期信息
39. --**>**
40. **<tx:method** name="transfer" propagation="REQUIRED"**/>**
41. **</tx:attributes>**
42. **</tx:advice>**
44. <!-- 配置切面 -->
45. **<aop:config>**
46. <!-- 配置切入点 -->
47. **<aop:pointcut** expression="execution(\* com.zs.spring.demo3.AccountService+.\*(..))" id="pointcut1"**/>**
48. <!-- 配置切面 -->
49. **<aop:advisor** advice-ref="txAdvice" pointcut-ref="pointcut1"**/>**
50. **</aop:config>**

测试：

1. /\*\*
2. \* @Description:Spring的声明式事务管理的方式二：基于AspectJ的XML方式的配置
3. \*/
4. @RunWith(SpringJUnit4ClassRunner.**class**)
5. @ContextConfiguration("classpath:applicationContext3.xml")
6. **public** **class** TransactionTest {
7. /\*\*
8. \* 一定要注入代理类:因为代理类进行增强的操作
9. \*/
10. @Resource(name = "accountService")
11. **private** AccountService accountService;
13. @Test
14. **public** **void** demo1() {
15. accountService.transfer("aaa", "bbb", 200d);
16. }
17. }



  --基于注解

1. /\*\*
2. \* @Transactional中的的属性 propagation :事务的传播行为 isolation :事务的隔离级别 readOnly :只读
3. \*                     rollbackFor :发生哪些异常回滚 noRollbackFor :发生哪些异常不回滚
4. \*                     rollbackForClassName 根据异常类名回滚
5. \*/
6. @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.DEFAULT, readOnly = **false**)
7. **public** **class** AccountServiceImpl **implements** AccountService {
8. // 注入转账的DAO
9. **private** AccountDao accountDao;
11. /\*\*
12. \* @param out
13. \*            :转出账号
14. \* @param in
15. \*            :转入账号
16. \* @param money
17. \*            :转账金额
18. \*/
19. @Override
20. **public** **void** transfer(String out, String in, Double money) {
21. accountDao.outMoney(out, money);
22. // int i = 1/0;
23. accountDao.inMoney(in, money);
24. }
26. **public** **void** setAccountDao(AccountDao accountDao) {
27. **this**.accountDao = accountDao;
28. }
29. }

applicationContext4.xml

1. <!-- 引入外部的属性文件 -->
2. **<context:property-placeholder** location="classpath:jdbc.properties"**/>**
4. <!-- 配置c3p0连接池 -->
5. **<bean** id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"**>**
6. **<property** name="driverClass" value="${jdbc.driverClass}" **/>**
7. **<property** name="jdbcUrl" value="${jdbc.url}" **/>**
8. **<property** name="user" value="${jdbc.username}" **/>**
9. **<property** name="password" value="${jdbc.password}" **/>**
10. **</bean>**
12. <!-- 配置业务层类 -->
13. **<bean** id="accountService" class="com.zs.spring.demo4.AccountServiceImpl"**>**
14. **<property** name="accountDao" ref="accountDao" **/>**
15. **</bean>**
17. <!-- 配置DAO类(简化，会自动配置JdbcTemplate) -->
18. **<bean** id="accountDao" class="com.zs.spring.demo4.AccountDaoImpl"**>**
19. **<property** name="dataSource" ref="dataSource" **/>**
20. **</bean>**
22. <!-- ==================================4.使用注解配置声明式事务============================================ -->
24. <!-- 配置事务管理器 -->
25. **<bean** id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>**
26. **<property** name="dataSource" ref="dataSource" **/>**
27. **</bean>**
29. <!-- 开启注解事务 -->
30. **<tx:annotation-driven** transaction-manager="transactionManager"**/>**

测试：

1. @RunWith(SpringJUnit4ClassRunner.**class**)
2. @ContextConfiguration("classpath:applicationContext4.xml")
3. **public** **class** TransactionTest {
5. /\*\*
6. \* 一定要注入代理类:因为代理类进行增强的操作
7. \*/
8. @Resource(name = "accountService")
9. **private** AccountService accountService;
11. @Test
12. **public** **void** demo1() {
13. accountService.transfer("aaa", "bbb", 200d);
14. }
15. }