**Spring事物与传播行为**

# 课程目标:

Spring事物概念、理解事物的传播行为

# 一、事物的概述

## ⑴ 原子性（Atomicity）

　　原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

## ⑵ 一致性（Consistency）

　　一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

　　拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

## ⑶ 隔离性（Isolation）

　　隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

　　即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

　　关于事务的隔离性数据库提供了多种隔离级别，稍后会介绍到。

## ⑷ 持久性（Durability）

　　持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

　　例如我们在使用JDBC操作数据库时，在提交事务方法后，提示用户事务操作完成，当我们程序执行完成直到看到提示后，就可以认定事务以及正确提交，即使这时候数据库出现了问题，也必须要将我们的事务完全执行完成，否则就会造成我们看到提示事务处理完毕，但是数据库因为故障而没有执行事务的重大错误。

# 二、程序中事务控制

## 2.1 环境准备

用户访问—》C--》 Service---》Dao

一个业务的成功： 调用的service是执行成功的，意味着service中调用的所有的dao是执行成功的。  **事务应该在Service层统一控制。**

1）没有应用事务的代码：

2）模拟：

在service中调用2次dao， 希望其中一个dao执行失败，整个操作要回滚。

开发步骤：

1. 后台环境准备

数据库、表/entity/dao/service

2. dao 的实现用JdbcTemplate

3. 对象创建都有Spring容器完成

|  |
| --- |
|  |

# 三、程序中事务控制

## 3.1事务控制概述

**编程式事务控制**

自己手动控制事务，就叫做编程式事务控制。

Jdbc代码：

Conn.setAutoCommite(false); // 设置手动控制事务

Hibernate代码：

Session.beginTransaction(); // 开启一个事务

【细粒度的事务控制： 可以对指定的方法、指定的方法的某几行添加事务控制】

(比较灵活，但开发起来比较繁琐： 每次都要开启、提交、回滚.)

**声明式事务控制**

Spring提供了对事务的管理, 这个就叫声明式事务管理。

Spring提供了对事务控制的实现。用户如果想用Spring的声明式事务管理，只需要在配置文件中配置即可； 不想使用时直接移除配置。这个实现了对事务控制的最大程度的解耦。

Spring声明式事务管理，核心实现就是基于Aop。

【粗粒度的事务控制： 只能给整个方法应用事务，不可以对方法的某几行应用事务。】

(因为aop拦截的是方法。)

Spring声明式事务管理器类：

Jdbc技术：**DataSourceTransactionManager**

Hibernate技术：**HibernateTransactionManager**

## 3.2编程事物管理

### 3.2.1手动管理事物

|  |
| --- |
| 1. UserDao.java |
| **@Repository**  **public class UserDao {**  **@Autowired**  **private JdbcTemplate jdbcTemplate;**  **public void add(String name, Integer age) {**  **String sql = "INSERT INTO users(NAME, age) VALUES(?,?);";**  **int update = jdbcTemplate.update(sql, name, age);**  **System.*out*.println("updateResult:" + update);**  **}**  **}** |
| 2. UserService |
| **@Service**  **public class UserService {**  **@Autowired**  **private UserDao userDao;**  **public void add() {**  **userDao.add("lisi", 18);**  **int i=1/0;//可能会发生异常**  **userDao.add("yushengjun", 19);**  **}**  **}** |
| 3. App 测试类 |
| **public** **class** UserTest {  **public** **static** **void** main(String[] args) {  ClassPathXmlApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("bean.xml");  UserService userService = (UserService) applicationContext.getBean("userService");  userService.add();  }  } |
| 4.手动事物管理类 |
| @Component  **public** **class** TransactionUtils {  // 事物管理器  @Autowired  **private** DataSourceTransactionManager dataSourceTransactionManager;  **public** TransactionStatus begin() {  TransactionStatus transaction = dataSourceTransactionManager.getTransaction(**new** DefaultTransactionDefinition());  **return** transaction;  }  **public** **void** commit(TransactionStatus transaction) {  dataSourceTransactionManager.commit(transaction);  }  **public** **void** rollback(TransactionStatus transaction) {  dataSourceTransactionManager.rollback(transaction);  }  } |
| 5. bean.xml (Spring务管理配置) |
| **<?xml version=*"1.0"* encoding=*"UTF-8"*?>**  **<beans xmlns=*"http://www.springframework.org/schema/beans"***  **xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"***  **xmlns:context=*"http://www.springframework.org/schema/context"***  **xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"***  **xsi:schemaLocation=*"http://www.springframework.org/schema/beans***  ***http://www.springframework.org/schema/beans/spring-beans.xsd***  ***http://www.springframework.org/schema/context***  ***http://www.springframework.org/schema/context/spring-context.xsd***  ***http://www.springframework.org/schema/aop***  ***http://www.springframework.org/schema/aop/spring-aop.xsd***  ***http://www.springframework.org/schema/tx***  ***http://www.springframework.org/schema/tx/spring-tx.xsd"*>**  **<!-- 开启注解 -->**  **<context:component-scan base-package=*"com.itmayiedu"*></context:component-scan>**  **<!-- 1. 数据源对象: C3P0连接池 -->**  **<bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>**  **<property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"*></property>**  **<property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost:3306/test"*></property>**  **<property name=*"user"* value=*"root"*></property>**  **<property name=*"password"* value=*"root"*></property>**  **</bean>**  **<!-- 2. JdbcTemplate工具类实例 -->**  **<bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>**  **<property name=*"dataSource"* ref=*"dataSource"*></property>**  **</bean>**  **<!-- 配置事物 -->**  **<bean id=*"DataSourceTransactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>**  **<property name=*"dataSource"* ref=*"dataSource"*></property>**  **</bean>**  **</beans>** |

## 3.3声明式事务管理

### 3.3.1 XML方式实现

|  |
| --- |
| 1. UserDao.java |
| **@Repository**  **public class UserDao {**  **@Autowired**  **private JdbcTemplate jdbcTemplate;**  **public void add(String name, Integer age) {**  **String sql = "INSERT INTO users(NAME, age) VALUES(?,?);";**  **int update = jdbcTemplate.update(sql, name, age);**  **System.*out*.println("updateResult:" + update);**  **}**  **}** |
| 2. UserService |
| **@Service**  **public class UserService {**  **@Autowired**  **private UserDao userDao;**  **public void add() {**  **userDao.add("lisi", 18);**  **int i=1/0;//可能会发生异常**  **userDao.add("yushengjun", 19);**  **}**  **}** |
| 3. App 测试类 |
| **public** **class** UserTest {  **public** **static** **void** main(String[] args) {  ClassPathXmlApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("bean.xml");  UserService userService = (UserService) applicationContext.getBean("userService");  userService.add();  }  } |
| 4.手动事物管理类 |
| @Component  **public** **class** TransactionUtils {  // 事物管理器  @Autowired  **private** DataSourceTransactionManager dataSourceTransactionManager;  **public** TransactionStatus begin() {  TransactionStatus transaction = dataSourceTransactionManager.getTransaction(**new** DefaultTransactionDefinition());  **return** transaction;  }  **public** **void** commit(TransactionStatus transaction) {  dataSourceTransactionManager.commit(transaction);  }  **public** **void** rollback(TransactionStatus transaction) {  dataSourceTransactionManager.rollback(transaction);  }  } |
| 5. bean.xml (Spring务管理配置) |
| **<!-- 开启注解 -->**  **<context:component-scan base-package=*"com.itmayiedu"*></context:component-scan>**  **<!-- 1. 数据源对象: C3P0连接池 -->**  **<bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>**  **<property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"*></property>**  **<property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost:3306/test"*></property>**  **<property name=*"user"* value=*"root"*></property>**  **<property name=*"password"* value=*"root"*></property>**  **</bean>**  **<!-- 2. JdbcTemplate工具类实例 -->**  **<bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>**  **<property name=*"dataSource"* ref=*"dataSource"*></property>**  **</bean>**  **<!-- 配置事物 -->**  **<bean id=*"dataSourceTransactionManager"***  **class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>**  **<property name=*"dataSource"* ref=*"dataSource"*></property>**  **</bean>**  **<!—配置事物增强-->**  **<tx:advice id=*"txAdvice"* transaction-manager=*"dataSourceTransactionManager"*>**  **<tx:attributes>**  **<tx:method name=*"get\*"* read-only=*"true"* />**  **<tx:method name=*"find\*"* read-only=*"true"* />**  **<tx:method name=*"\*"* read-only=*"false"* />**  **</tx:attributes>**  **</tx:advice>**  **<!-- Aop配置： 拦截哪些方法(切入点表表达式) + 应用上面的事务增强配置 -->**  **<aop:config>**  **<aop:pointcut expression=*"execution(\* com.itmayiedu.service.\*.\*(..))"***  **id=*"pt"* />**  **<aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"pt"* />**  **</aop:config>** |

#### 使用事物注意事项

事物是程序运行如果没有错误,会自动提交事物,如果程序运行发生异常,则会自动回滚。

如果使用了try捕获异常时.一定要在catch里面手动回滚。

事物手动回滚代码

TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();

### 3.3.2 注解方式实现

使用注解实现Spring的声明式事务管理，更加简单！

步骤：

1） 必须引入Aop相关的jar文件

2） bean.xml中指定注解方式实现声明式事务管理以及应用的事务管理器类

3）在需要添加事务控制的地方，写上: @Transactional

@Transactional注解：

1）应用事务的注解

2）定义到方法上： 当前方法应用spring的声明式事务

3）定义到类上： 当前类的所有的方法都应用Spring声明式事务管理;

4）定义到父类上： 当执行父类的方法时候应用事务。

|  |
| --- |
| Bean.xm |
| **<?xml version=*"1.0"* encoding=*"UTF-8"*?>**  **<beans xmlns=*"http://www.springframework.org/schema/beans"***  **xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"***  **xmlns:context=*"http://www.springframework.org/schema/context"***  **xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"***  **xsi:schemaLocation=*"http://www.springframework.org/schema/beans***  ***http://www.springframework.org/schema/beans/spring-beans.xsd***  ***http://www.springframework.org/schema/context***  ***http://www.springframework.org/schema/context/spring-context.xsd***  ***http://www.springframework.org/schema/aop***  ***http://www.springframework.org/schema/aop/spring-aop.xsd***  ***http://www.springframework.org/schema/tx***  ***http://www.springframework.org/schema/tx/spring-tx.xsd"*>**  **<!-- 开启注解 -->**  **<context:component-scan base-package=*"com.itmayiedu"*></context:component-scan>**  **<!-- 1. 数据源对象: C3P0连接池 -->**  **<bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>**  **<property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"*></property>**  **<property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost:3306/test"*></property>**  **<property name=*"user"* value=*"root"*></property>**  **<property name=*"password"* value=*"root"*></property>**  **</bean>**  **<!-- 2. JdbcTemplate工具类实例 -->**  **<bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>**  **<property name=*"dataSource"* ref=*"dataSource"*></property>**  **</bean>**  **<!-- 配置事物 -->**  **<bean id=*"dataSourceTransactionManager"***  **class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>**  **<property name=*"dataSource"* ref=*"dataSource"*></property>**  **</bean>**  **<!-- 开启注解事物 -->**  **<tx:annotation-driven transaction-manager=*"dataSourceTransactionManager"*/>**  **</beans>** |
| UserService |
| **@Transactional**  **public void add() {**  **try {**  **userDao.add("lisi", 18);**  **int i = 1 / 0;**  **userDao.add("yushengjun", 19);**  **} catch (Exception e) {**  **TransactionAspectSupport.*currentTransactionStatus*().setRollbackOnly();**  **}**    **}** |
|  |

# 四、传播七种行为

Spring中事务的定义：

Propagation（key属性确定代理应该给哪个方法增加事务行为。这样的属性最重要的部份是传播行为。）有以下选项可供使用：

* PROPAGATION\_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(会统一为最外层的事物)。
* PROPAGATION\_SUPPORTS--支持当前事务，如果当前没有事务，就以非事务方式执行。(只支持当前方法的事物,比如调用了其他Service.add()方法,那么这个add方法就不会参与事物操作)
* PROPAGATION\_MANDATORY--支持当前事务，如果当前没有事务，就抛出异常。 -(调用的其他方法没有添加事物的话,抛出来异常,使用非常少)
* PROPAGATION\_REQUIRES\_NEW--新建事务，如果当前存在事务，把当前事务挂起(事物之间互相不影响)。
* PROPAGATION\_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。
* PROPAGATION\_NEVER--以非事务方式执行，如果当前存在事务，则抛出异常。

|  |
| --- |
| **@Transactional(**  **readOnly = false, // 读写事务**  **timeout = -1, // 事务的超时时间不限制**  **noRollbackFor = ArithmeticException.class, // 遇到数学异常不回滚**  **isolation = Isolation.*DEFAULT*, // 事务的隔离级别，数据库的默认**  **propagation = Propagation.*REQUIRED* // 事务的传播行为**  **)** |

事务传播行为:

Propagation.REQUIRED

指定当前的方法必须在事务的环境下执行；

如果当前运行的方法，已经存在事务， 就会加入当前的事务；

Propagation.REQUIRED\_NEW

指定当前的方法必须在事务的环境下执行；

如果当前运行的方法，已经存在事务： 事务会挂起； 会始终开启一个新的事务，执行完后； 刚才挂起的事务才继续运行。

举例：

Class Log{

Propagation.REQUIRED

insertLog();

}

Propagation.REQUIRED

Void saveDept(){

insertLog(); // 加入当前事务 注意:这里就是上面总结的**当前**

**.. 异常, 会回滚**

saveDept();

}

Class Log{

Propagation.REQUIRED\_NEW

insertLog();

}

Propagation.REQUIRED

Void saveDept(){

insertLog(); // 始终开启事务

.. 异常, 日志不会回滚

saveDept();

}

测试步骤：

1）日志表Log\_

2）LogService.java

insertLog();