1、Spring框架中提供的事务处理方案

1、--适合中小项目使用：注解方案

Spring框架使用AOP实现（切面的方式）给存在的业务方法增加事务的功能，就是使用注解@Transactional增加事务。

@Transactional注解是Spring自己的注解。

--使用方式：

添加在public修饰的业务方法上面，使该业务方法具有事务，从而实现事务的管理。如果添加在不是public修饰的

方法上，Spring不会报错，只是不会发挥作用

注解中有属性，属性功能包括决定这个事务的隔离级别、传播行为、异常信息等，具体如下：

1）、propagation：设置事务传播行为。属性类型是名为Propagation的枚举类型，默认值Propagation.RQUIRED

(枚举中的属性和值使用都是"枚举名.属性",也可以理解为"类名."),具体该枚举还有什么可选项，可以Ctrl+点击

枚举类名字，一共有七个枚举属性

2）、isolation：设置事务的隔离级别。属性类型是Isolation枚举，默认值Isolation.DEFAULT。具体默认什么级

别，要看你使用的是什么类型数据库。

3）、readOnly：用于设置事务是否可重复读。属性类型为boolean，默认false。当我们的因为方法只是涉及到数据库的

查询操作时，一般设置为true，这样可以再一定程度上加快运行效率

4）、timeout：设置本操作与数据库的连接时间。单位秒，类型int，默认值为-1，没有时限。一般不设置

5）、rollbackFor：指定业务方法抛出什么异常时执行回滚操作，指定的是异常类的Class对象（"异常类类名.class"）。类型是Class[],默认为空数组，指定一个异常时可以不使用数组

6）、rollbackForClassName：指定业务方法抛出什么异常时执行回滚操作，指定的是异常类的类名。所以类型是字符串

数组，指定一个不需要使用数组

7）、noRollbackFor：指定不需要回滚的异常类的Class对象。其余和上面的rollbackFor一样

8）、noRollollbackForClassName：指定不需要回滚的异常类类名

--@Transactional的使用步骤：

1、需要声明事务管理器对象，如使用的是mybatis的方式进行数据库的访问

```
<bean id="xxx" class="DataSourceTransactionManager"/>
```

2、开启事务注解驱动，告诉Spring框架我要使用注解的方式管理事务。

开启后，Spring就会使用AOP的机制，创建注解@Transactional所修饰的业务方法的类的代理对象，给方法加入事务功

能。

--就是使用之前学过的AOP环绕通知方式给业务方法添加业务功能：

业务方法执行前添加功能：开启事务

业务方法执行

业务方法执行后添加功能：提交或者回滚事务

```
@Around(...)
Object myAround(...){
    try{
```

```
                buy(...);
                spring事务管理器.commit();
            }catch(Exception e){
                if(e instanceof RuntimeException){
                    spring的事务管理器.rollback();
                }
            }
        }
    3、在public修饰的业务方法上添加@Transactional
```

2、下面新建一个g-spring-anno-trans-07的项目中，使用f-spring-not-trans-06项目的代码实例，通过添加注解@Transactional来给业务方法添加事务功能

1）、在Spring配置文件中声明事务管理器和开启事务驱动，applicationContext.xml文件中添加如下：

```
  <!--使用Spring处理事务-->
<!--    1、声明事务管理器对象-->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
<!--    要管理哪个连接对象，需要把对象注入到管理器对象的属性中-->
        <property name="dataSource" ref="myDataSource"/>
    </bean>

<!--    2、开启事务注解驱动，告诉Spring使用的是注解方式管理事务，让Spring使用AOP方式创建代理对象-->
<!--    注意使用的是tx标签的annotation-driven，tx表示事务-->
    <tx:annotation-driven transaction-manager="transactionManager"/>
```

```
<?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:util="http://www.springframework.org/schema/util"
       xmlns:context="http://www.springframework.org/schema/context"
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/util
https://www.springframework.org/schema/util/spring-util.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!--让spring知道jdbc.properties文件位置-->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!--声明数据源DataSource对象，作用是连接数据库-->
    <bean id="myDataSource" class="com.alibaba.druid.pool.DruidDataSource"
          init-method="init" destroy-method="close">
<!--        使用set注入赋值给DruidDataSource提供连接数据库的信息-->
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
```

```
        <property name="password" value="${jdbc.password}"/>

        <property name="maxActive" value="${jdbc.maxActive}"/>
     </bean>

<!--    声明mybatis提供的SqlSessionFactoryBean类对象，这个类内部有一个方法是创建
SqlSessionFactory-->
<!--    对象的，这样我们声明SqlSessionFactoryBean类对象就是声明SqlSessionFactory对象
了。就像之前-->
<!--    使用的SqlSessionFactoryBuilder类中的build方法一样-->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <!--我们知道之前创建SqlSessionFactory对象时需要使用到mybatis的核心配置文件，而
            配置文件中的核心配置中没有了。改为上面的声明对象了，所以SqlSessionFactoryBean
类中
            有一个属性存储数据库信息，这时我们就把上面的数据源对象放到该属性中-->

        <!--通过set注入把数据库连接池赋给了dataSource属性,引用类型属性注意用ref-->
        <property name="dataSource" ref="myDataSource"/>

        <!--下面就是把核心配置文件读取到这个类中的configLocation属性中，属性是Resource类
型
            的，就像之前用mybatis中获取核心配置文件时Resources.getResourceAsStream一
样，是
            读取配置文件的,注意注入值是文件时要在前面添加路径标识：classpath-->
        <property name="configLocation" value="classpath:mybatis.xml"/>

        <!--最后创建的SqlSessionFactory对象类名是DefaultSqlSessionFactory,这是
SqlSessionFactory接口的实现类-->
    </bean>

<!--    这里就是创建dao对象，或者dao的代理对象，使用SqlSession的getMapper(接口的Class对
象)
        我们不能为每一个dao接口去一个一个调用getMapper方法生成代理对象，所以声明某个类对象
        一次性把所有的dao接口都生成代理对象，如下
        MapperScannerConfigurer这个类：可以一次性把符合条件的dao接口都生成其代理对象
        其中每个代理对象存储在Spring容器集合中的key是接口的首字母小写-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <!--之前我们要调用getMapper方法需要用到sqlSessionFactory对象和dao接口的Class对
象
        所以MapperScannerConfigurer中有两个属性要获取这些数据-->

        <!--sqlSessionFactory对象使用上面的声明的bean，用set注入-,属性是String类型，用
value-->
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
        <!--这里直接把dao接口的包名赋值给获取dao接口Class对象的属性，一次性创建所有dao接口
的代理对象
            MapperScannerConfigurer会扫描这个包装的所有接口，为每个接口都执行一次
getMapper方法，得到
            的每个接口的代理对象都放到Spring容器中
             -->
        <property name="basePackage" value="com.studymyself.dao"/>
    </bean>

<!--    声明service类对象-->
    <bean id="buyGoodsService"
class="com.studymyself.service.impl.BuyGoodsServiceImpl">
        <property name="goodsDao" ref="goodsDao"/>
```

```
        <property name="saleDao" ref="saleDao"/>
    </bean>

    <!--使用Spring处理事务-->
<!--    1、声明事务管理器对象-->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
<!--    要管理哪个连接对象，需要把对象注入到管理器对象的属性中-->
        <property name="dataSource" ref="myDataSource"/>
    </bean>

<!--    2、开启事务注解驱动，告诉Spring使用的是注解方式管理事务，让Spring使用AOP方式创建代理
对象-->
<!--    注意使用的是tx标签的annotation-driven，tx表示事务-->
    <tx:annotation-driven transaction-manager="transactionManager"/>

</beans>
```

2）、在业务方法中添加@Transactional注解，这里是BuyGoodsServiceImpl类中的buy方法上面，如下

```
package com.studymyself.service.impl;

import com.studymyself.dao.GoodsDao;
import com.studymyself.dao.SaleDao;
import com.studymyself.entity.Goods;
import com.studymyself.entity.Sale;
import com.studymyself.exceptions.NotEnoughException;
import com.studymyself.service.BuyGoodsService;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

public class BuyGoodsServiceImpl implements BuyGoodsService {

    GoodsDao goodsDao;
    SaleDao saleDao;

      /**
     *  rollbackFor:表示抛出指定异常进行回滚
     *  处理逻辑：
     *      1、Spring框架会首先检查方法抛出的异常是不是rollbackFor属性值列表的值，只要是
     *      不管是运行时还是编译时异常，都会执行回滚操作
     *      2、如果抛出的不是该属性值列表中的异常，就会判断该异常是运行时异常还是编译时异常
     *      前者的话就执行回滚，后者不执行回滚
     *      所以默认的该属性是抛出运行时异常就执行回滚操作
     *
     */

    //购买商品的方法
    @Transactional(
            //这里的属性自己依据业务需求设置值，一般来说都不设置，使用默认的
            propagation = Propagation.REQUIRED,
            isolation = Isolation.DEFAULT,
            timeout = -1,
            readOnly = false,
```

```
            rollbackFor = {NotEnoughException.class,NullPointerException.class}
            //或
            //rollbackForClassName = {"NotEnoughException","NullPointException"}

    )
    //一般我们都是直接加个@Transactional就行了
    //@Transactional
    @Override
    public void buy(Integer goodId, String goodName, Integer nums) {

        System.out.println("==执行buy方法，开始购买商品==");
        //添加商品销售记录（向t_sale表中插入数据）
        //创建Sale类，并且向其中属性赋值
        Sale sale1 = new Sale();
        sale1.setGid(goodId);
        sale1.setGname(goodName);
        sale1.setNums(nums);
        int count = saleDao.insertSale(sale1);

        //根据提供的信息数据查询商品信息
        Goods good = goodsDao.selectByIdAndName(sale1);

        //根据查询结果判断是否要更新商品库存
        if (good == null){
            count -= 1;
            System.out.println("购买失败，请重新购买");
            throw new NullPointerException("商品:"+sale1.getGname()+" 编
号:"+sale1.getGid()+"不存在！！！");
        }else if (good.getAmount()<nums){
            count -= 1;
            System.out.println("购买失败，请重新购买");
            throw new NotEnoughException("商品:"+sale1.getGname()+" 编
号:"+sale1.getGid()+" 库存不足！！！");

        }
        //更新商品表
        goodsDao.updateAmount(sale1);
        if (count>0){
            System.out.println("已购买商品："+goodName);
        }

    }

    public void setGoodsDao(GoodsDao goodsDao) {
        this.goodsDao = goodsDao;
    }

    public void setSaleDao(SaleDao saleDao) {
        this.saleDao = saleDao;
    }
}
```

测试代码如下：

```
package com.studymyself;
```

```
import com.studymyself.service.BuyGoodsService;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyTest {

    /**
     * 测试购买商品的方法
     */
    @Test
    public void testBuy(){

        //定义Spring配置文件路径
        String config = "applicationContext.xml";

        //获取Spring容器对象
        ApplicationContext ac = new ClassPathXmlApplicationContext(config);

        //获取service对象
        BuyGoodsService buyGoodsService = (BuyGoodsService)
ac.getBean("buyGoodsService");
        System.out.println(buyGoodsService.getClass().getName());

        buyGoodsService.buy(1002,"手机",20);

    }

}
```

我们可以发现service对象已经是JDK动态代理对象了