使用AspectJ的AOP基于xml配置和管理事务

2、--适合大型项目，有很多的类，方法。需要大量的配置事务，使用AspectJ框架的功能，在Spring配置文件中声明类和方法的事务。
    --这种方式业务方法和事务配置完全分离
实现步骤：--都是在Spring的xml配置文件中实现的
    1）、首先需要加入AspectJ框架的依赖
    2）、声明事务管理器对象的类型
    使用的是mybatis方式访问数据库
    <bean id="xxx" class="DataSourceTransactionManager"/>
    3）、声明方法需要的事务类型（就是配置方法所加事务的属性：隔离级别、传播方式、超时时限）

复制f-spring-not-trans-06项目的代码到新建的项目模块h-spring-aspectj-trans-08

1）、在pom文件中添加AspectJ依赖

```
 <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aspects</artifactId>
      <version>5.2.5.RELEASE</version>
    </dependency>
```

2）、在Spring配置文件中声明事务的相关配置

首先在service的实现类中添加如下示例代码

```
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;

    //开发者将方法归类，前面添加相同前缀
    public void addGoods(){};
    public void addSale(){};

    public void modifyGoods(){};
    public void modifySale(){};
```

```
    public void queryGoods(){};
    public void querySale(){};



    //购买商品的方法
    @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;
    }
}
```

Spring配置文件如下

```
<?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"
       xmlns:aop="http://www.springframework.org/schema/aop"
       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
http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop.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>

<!--   声明式事务处理：与源代码是完全分离的-->
<!--   1、声明事务管理器对象-->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="myDataSource"/>
    </bean>

<!--   2、声明业务方法所加事务的属性（隔离级别、传播方式、超时时间）使用的还是tx标签的属性
       id:自定义名称，表示tx-advice标签的配置的唯一标识id
       transaction-manager：是配置的事务管理器对象的唯一标识id
-->
    <tx:advice id="myAdvice" transaction-manager="transactionManager">
<!--    tx:attributes:配置事务的属性-->
        <tx:attributes>
            <!--tx:method：给具体的业务方法配置事务属性，method标签可以有多个，给不同的方
法配置不同属性
                                的事务。
                    name属性：填方法名称
                      1、简洁的方法名称，不带有包名和类名
                      2、方法名可以使用通配符*，表示任意字符
                      所以类中业务方法是同一类型的操作时加个前缀 ，如都是修改表的操作方法都加
个modifyXxx
                      插入数据的方法都加addYyy，查询的方法都加queryZzz等等，这样我们就可以
用通配符*代替后
                      面的部分，将前缀一致的方法都配置事务属性了
                    propagation：事务的传播方式
                    isolation：事务的隔离级别
                    rollback-for：指定异常类的全限定名
            -->
            <tx:method name="buy" propagation="REQUIRED" isolation="DEFAULT"
```

```
                        rollback-
for="java.lang.NullPointerException,com.studymyself.exceptions.NotEnoughExceptio
n"/>
            <!--使用通配符*，指定多个相同事务属性的方法，-->
            <tx:method name="add*" propagation="REQUIRED"/>
            <!--modify前缀的方法添加默认属性的事务-->
            <tx:method name="modify*"/>
            <!--query前缀的查询类方法添加只读属性的事务-->
            <tx:method name="*" read-only="true"/>
            <!--上面中Spring寻找添加事务的方法顺序是完整类名的，再是有通配符的，再是全是通配
符的-->
        </tx:attributes>
    </tx:advice>

    <!--配置AOP，指定哪些包装的类需要使用事务-->
    <aop:config>
        <!--配置切入点表达式：指定哪些包装的类要添加事务
            id：切入点表达式配置的唯一标识id
            expression：切入点表达式，指定哪些包装的类药添加事务功能，AspectJ会为其创建代
理对象
            因为项目中service包是有多个的，路径都不相同，还有就是service实现类也不在
service包下
            所需要合理使用切入点表达式的通配符
            com.studymyself.service.impl
            com.service.impl
            com.crm.service.impl
        -->
        <aop:pointcut id="servicePt" expression="execution(* *..servicce..*
(..))"/>

        <!--这里配置增强器：目的是关联上面配置的advice和pointcut配置的方法和包类名-->
        <aop:advisor advice-ref="myAdvice" pointcut-ref="servicePt"/>
    </aop:config>

</beans>
```

在测试文件中测试

在测试中出现错误，导致AspectJ框架并没有为service类创建代理实现类，导致没有将事务加入到业务方法中。

主要是错在切入点表达式中的，忘记写了方法名的通配符和service包写错了