# Spring框架的第三天 -- 拓薪教育

## 第一章：AOP概念的引入

### 1. 编写入门案例，完成AOP概念的引入

创建maven的项目，引入开发的坐标

<dependencies>  
        <dependency>  
            <groupId>org.springframework</groupId>  
            <artifactId>spring-context</artifactId>  
            <version>5.0.2.RELEASE</version>  
        </dependency>  
        <dependency>  
            <groupId>commons-logging</groupId>  
            <artifactId>commons-logging</artifactId>  
            <version>1.2</version>  
        </dependency>  
        <dependency>  
            <groupId>log4j</groupId>  
            <artifactId>log4j</artifactId>  
            <version>1.2.12</version>  
        </dependency>  
        <!--有单元测试的环境，Spring5版本，Junit4.12版本-->  
        <dependency>  
            <groupId>junit</groupId>  
            <artifactId>junit</artifactId>  
            <version>4.12</version>  
            <scope>test</scope>  
        </dependency>  
        <!--连接池-->  
        <dependency>  
            <groupId>com.alibaba</groupId>  
            <artifactId>druid</artifactId>  
            <version>1.1.10</version>  
        </dependency>  
        <!--mysql驱动包-->  
        <dependency>  
            <groupId>mysql</groupId>  
            <artifactId>mysql-connector-java</artifactId>  
            <version>5.1.6</version>  
        </dependency>  
        <!-- Spring整合Junit测试的jar包 -->  
        <dependency>  
            <groupId>org.springframework</groupId>  
            <artifactId>spring-test</artifactId>  
            <version>5.0.2.RELEASE</version>  
            <scope>test</scope>  
        </dependency>  
​  
    </dependencies>

注意：从资料中拷贝TxUtils类，进行事务管理

AccountServiceImpl代码如下

package cn.tx.demo1;  
​  
import java.sql.SQLException;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class AccountServiceImpl implements AccountService {  
​  
    private AccountDao accountDao;  
    public void setAccountDao(AccountDao accountDao) {  
        this.accountDao = accountDao;  
   }  
​  
    /\*\*  
     \* 对saveAll方法进行增强  
     \* @param account1  
     \* @param account2  
     \* @throws SQLException  
     \*/  
    @Override  
    public void saveAll(Account account1, Account account2) throws SQLException {  
        // 保存1账号  
        accountDao.save(account1);  
​  
        // 模拟异常  
        // int a = 1/0;  
​  
        // 保存2账号  
        accountDao.save(account2);  
   }  
​  
​  
    /\*\*  
     \* 保存2个  
     \* @param account1  
     \* @param account2  
​  
    @Override  
    public void saveAll(Account account1, Account account2) {  
        try {  
            // 开启事务  
            TxUtils.startTransaction();  
            // 保存1账号  
            accountDao.save(account1);  
​  
            // 模拟异常  
            // int a = 1/0;  
​  
            // 保存2账号  
            accountDao.save(account2);  
            // 提交事务/回滚事务  
            TxUtils.commit();  
        } catch (Exception e) {  
            // 打印异常信息  
            e.printStackTrace();  
            // 回滚事务  
            TxUtils.rollback();  
        } finally {  
            // 关闭资源  
            TxUtils.close();  
        }  
    }  
     \*/  
​  
}

AccountDaoImpl 代码如下

package cn.tx.demo1;  
​  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.SQLException;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class AccountDaoImpl implements AccountDao {  
​  
    /\*\*  
     \* 保存  
     \* @param account  
     \*/  
    @Override  
    public void save(Account account) throws SQLException {  
        // System.out.println("持久层：保存账户...");  
        // 把数据存储到数据库中  
        // 先获取到连接  
        Connection conn = TxUtils.getConnection();  
        // 编写sql语句  
        String sql = "insert into account values (null,?,?)";  
        // 预编译SQL语句  
        PreparedStatement stmt = conn.prepareStatement(sql);  
        // 设置值  
        stmt.setString(1,account.getName());  
        stmt.setDouble(2,account.getMoney());  
        // 执行操作  
        stmt.executeUpdate();  
        // 关闭资源 ，conn不能关闭  
        stmt.close();  
   }  
​  
}

生成代理对象的代码如下

package cn.tx.demo1;  
​  
import java.lang.reflect.InvocationHandler;  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;  
import java.lang.reflect.Proxy;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*  
 \* 传入目标对象，生成该对象的代理对象，返回。对目标对象的方法进行增强  
 \*/  
public class JdkProxy {  
​  
    /\*\*  
     \* 获取代理对象，返回，增强目标对象的方法  
     \* @param accountService  
     \* @return  
     \*/  
    public static Object getProxy(AccountService accountService){  
​  
        /\*\*  
         \* 使用Jdk的动态代理生成代理对象  
         \*/  
        Object proxy = Proxy.newProxyInstance(JdkProxy.class.getClassLoader(), accountService.getClass().getInterfaces(), new InvocationHandler() {  
​  
            /\*\*  
             \* 调用代理对象的方法，invoke方法就会去执行  
             \* @param proxy  
             \* @param method  
             \* @param args  
             \* @return  
             \* @throws Throwable  
             \*/  
            @Override  
            public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
                Object result = null;  
                try {  
                    // 开启事务  
                    TxUtils.startTransaction();  
​  
                    // 对象目标对象的方法进行增强  
                    result = method.invoke(accountService, args);  
​  
                    // 提交事务  
                    TxUtils.commit();  
               } catch (Exception e) {  
                    e.printStackTrace();  
                    TxUtils.rollback();  
               } finally {  
                    // 关闭资源  
                    TxUtils.close();  
               }  
                return result;  
           }  
       });  
        return proxy;  
   }  
​  
}

测试的方法如下

package cn.tx.test;  
​  
import cn.tx.demo1.Account;  
import cn.tx.demo1.AccountService;  
import cn.tx.demo1.JdkProxy;  
import org.junit.Test;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
​  
import java.sql.SQLException;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class Demo1 {  
​  
    @Test  
    public void run1() throws SQLException {  
        ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext\_demo1.xml");  
        // 获取service对象  
        AccountService accountService = (AccountService) ac.getBean("accountService");  
​  
        Account account1 = new Account();  
        account1.setName("熊大");  
        account1.setMoney(1000d);  
​  
        Account account2 = new Account();  
        account2.setName("美羊羊");  
        account2.setMoney(1000d);  
​  
        // 生成代理对象  
        Object proxyobj = JdkProxy.getProxy(accountService);  
        // 强转  
        AccountService proxy = (AccountService) proxyobj;  
​  
        // 调用代理对象方法  
        proxy.saveAll(account1,account2);  
   }  
​  
}

## 第二章：AOP相关的概念

### 1. AOP的概述

什么是AOP的技术？

在软件业，AOP为Aspect Oriented Programming的缩写，意为：面向切面编程

* AOP是一种编程范式，隶属于软工范畴，指导开发者如何组织程序结构
* AOP最早由AOP联盟的组织提出的,制定了一套规范.Spring将AOP思想引入到框架中,必须遵守AOP联盟的规范
* 通过预编译方式或者运行期动态代理实现程序功能的统一维护的一种技术
* AOP是OOP的延续，是软件开发中的一个热点，也是Spring框架中的一个重要内容，是函数式编程的一种衍生范型
* 利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率

AOP:面向切面编程.(思想.---解决OOP遇到一些问题)

AOP采取横向抽取机制，取代了传统纵向继承体系重复性代码（性能监视、事务管理、安全检查、缓存）

为什么要学习AOP，可以在不修改源代码的前提下，对程序进行增强！！

### 2. AOP的优势

运行期间，不修改源代码的情况下对已有的方法进行增强

优势

1. 减少重复的代码
2. 提供开发的效率
3. 维护方便

### 3. AOP的底层原理

JDK的动态代理技术

​ 1、为接口创建代理类的字节码文件

​ 2、使用ClassLoader将字节码文件加载到JVM

​ 3、创建代理类实例对象，执行对象的目标方法

cglib代理技术

## 第三章：Spring的AOP技术-配置文件方式

### 1. AOP相关的术语

Joinpoint(连接点) 所谓连接点是指那些被拦截到的点。在spring中,这些点指的是方法,因为spring只支持方法类型的连接点

Pointcut(切入点) -- 所谓切入点是指我们要对哪些Joinpoint进行拦截的定义

Advice(通知/增强)-- 所谓通知是指拦截到Joinpoint之后所要做的事情就是通知.通知分为前置通知,后置通知,异常通知,最终通知,环绕通知(切面要完成的功能)

Target(目标对象)-- 代理的目标对象

Weaving(织入)-- 是指把增强应用到目标对象来创建新的代理对象的过程

Proxy（代理）-- 一个类被AOP织入增强后，就产生一个结果代理类

Aspect(切面)-- 是切入点和通知的结合，以后咱们自己来编写和配置的

### 2. AOP配置文件方式的入门

创建maven项目，坐标依赖

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.0.2.RELEASE</version>  
 </dependency>  
 <dependency>  
 <groupId>commons-logging</groupId>  
 <artifactId>commons-logging</artifactId>  
 <version>1.2</version>  
 </dependency>  
 <dependency>  
 <groupId>log4j</groupId>  
 <artifactId>log4j</artifactId>  
 <version>1.2.12</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>5.0.2.RELEASE</version>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 </dependency>  
    
    <!-- AOP联盟 -->  
        <dependency>  
            <groupId>aopalliance</groupId>  
            <artifactId>aopalliance</artifactId>  
            <version>1.0</version>  
        </dependency>  
        <!-- Spring Aspects -->  
        <dependency>  
            <groupId>org.springframework</groupId>  
            <artifactId>spring-aspects</artifactId>  
            <version>5.0.2.RELEASE</version>  
        </dependency>  
        <!-- aspectj -->  
        <dependency>  
            <groupId>org.aspectj</groupId>  
            <artifactId>aspectjweaver</artifactId>  
            <version>1.8.3</version>  
        </dependency>  
  </dependencies>

创建Spring的配置文件，引入具体的AOP的schema约束

<?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:context="http://www.springframework.org/schema/context"  
       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/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">  
   
</beans>

创建包结构，编写具体的接口和实现类

package cn.tx.demo2;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
public class UserServiceImpl implements UserService {  
​  
    @Override  
    public void save() {  
        System.out.println("业务层：保存用户...");  
   }  
​  
}

将目标类配置到Spring中

<bean id="userService" class="cn.tx.demo2.UserServiceImpl"/>

定义切面类

package cn.tx.demo2;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*  
 \* 自定义切面类 = 切入点(表达式) + 通知(增强的代码)  
 \*/  
public class MyXmlAspect {  
​  
    /\*\*  
     \* 通知  
     \*/  
    public void log(){  
        // 发送手机短信  
        // 发送邮件/记录日志/事务管理  
​  
        System.out.println("增强的方法执行了...");  
   }  
​  
}

在配置文件中定义切面类

<bean id="myXmlAspect" class="cn.tx.demo2.MyXmlAspect"/>

在配置文件中完成aop的配置

   <!--配置AOP的增强-->  
    <aop:config>  
        <!--配置切面 = 切入点 + 通知组成-->  
        <aop:aspect ref="myXmlAspect">  
            <!--前置通知：UserServiceImpl的save方法执行前，会增强-->  
            <aop:before method="log" pointcut="execution(public void cn.tx.demo2.UserServiceImpl.save())" />  
        </aop:aspect>  
    </aop:config>

完成测试

package cn.tx.test;  
​  
import cn.tx.demo2.UserService;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.test.context.ContextConfiguration;  
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
@RunWith(SpringJUnit4ClassRunner.class)  
@ContextConfiguration("classpath:applicationContext\_demo2.xml")  
public class Demo2 {  
​  
    @Autowired  
    private UserService userService;  
​  
    /\*\*  
     \* 测试  
     \*/  
    @Test  
    public void run1(){  
        userService.save();  
   }  
​  
}

### 3. 切入点的表达式

再配置切入点的时候，需要定义表达式，具体展开如下：

切入点表达式的格式如下：

* execution([修饰符] 返回值类型 包名.类名.方法名(参数))
* 修饰符可以省略不写，不是必须要出现的。
* 返回值类型是不能省略不写的，根据你的方法来编写返回值。可以使用 \* 代替。
* 包名例如：com.tx.demo3.BookDaoImpl
  + 首先com是不能省略不写的，但是可以使用 \* 代替
  + 中间的包名可以使用 \* 号代替
  + 如果想省略中间的包名可以使用 ..
* 类名也可以使用 \* 号代替，也有类似的写法：\*DaoImpl
* 方法也可以使用 \* 号代替
* 参数如果是一个参数可以使用 \* 号代替，如果想代表任意参数使用 ..

<?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:context="http://www.springframework.org/schema/context"  
       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/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">  
​  
    <!--管理bean对象-->  
    <bean id="userService" class="cn.tx.demo2.UserServiceImpl"/>  
​  
    <!--配置切面类，把该类交给IOC容器管理-->  
    <bean id="myXmlAspect" class="cn.tx.demo2.MyXmlAspect"/>  
​  
    <!--配置AOP的增强-->  
    <aop:config>  
        <!--配置切面 = 切入点 + 通知组成-->  
        <aop:aspect ref="myXmlAspect">  
            <!--前置通知：UserServiceImpl的save方法执行前，会增强  
                <aop:before method="log" pointcut="execution(public void cn.tx.demo2.UserServiceImpl.save())" />  
            -->  
​  
            <!--  
                切入点的表达式  
                    execution() 固定的写法  
                    public         是可以省略不写的  
                   方法的返回值     int String 通用的写法，可以编写 \* 不能省略不写的  
                   包名+类名       不能省略不写的，编写 \* UserServiceImpl AccountServiceImpl  
                   方法名称         save() 可以写 \*  
                   参数列表         (..) 表示任意类型和个数的参数  
                   比较通用的表达式：execution(public \* cn.tx.\*.\*ServiceImpl.\*(..))  
            -->  
            <aop:before method="log" pointcut="execution(\* cn.tx.\*.\*ServiceImpl.save\*(..))" />  
​  
        </aop:aspect>  
    </aop:config>  
​  
</beans>

### 4. AOP的通知类型

1. 前置通知 目标方法执行前，进行增强。
2. 最终通知 目标方法执行成功或者失败，进行增强。
3. 后置通知 目标方法执行成功后，进行增强。
4. 异常通知 目标方法执行失败后，进行增强。
5. 环绕通知 目标方法执行前后，都可以进行增强。目标对象的方法需要手动执行。

<?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:context="http://www.springframework.org/schema/context"  
       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/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">  
​  
    <!--管理bean对象-->  
    <bean id="userService" class="cn.tx.demo2.UserServiceImpl"/>  
​  
    <!--配置切面类，把该类交给IOC容器管理-->  
    <bean id="myXmlAspect" class="cn.tx.demo2.MyXmlAspect"/>  
​  
    <!--配置AOP的增强-->  
    <aop:config>  
        <!--配置切面 = 切入点 + 通知组成-->  
        <aop:aspect ref="myXmlAspect">  
             
            <!--  
                AOP的通知类型  
                    前置通知：目标方法执行前，进行增强。  
                        <aop:before method="log" pointcut="execution(\* cn.tx.\*.\*ServiceImpl.save\*(..))" />  
                    最终通知：目标方法执行成功或者失败，进行增强。  
                        <aop:after method="log" pointcut="execution(\* cn.tx.\*.\*ServiceImpl.save\*(..))" />  
                    后置通知：目标方法执行成功后，进行增强。  
                        <aop:after-returning method="log" pointcut="execution(\* cn.tx.\*.\*ServiceImpl.save\*(..))" />  
                    异常通知：目标方法执行失败后，进行增强。  
                        <aop:after-throwing method="log" pointcut="execution(\* cn.tx.\*.\*ServiceImpl.save\*(..))" />  
                    环绕通知：目标方法执行前后，都可以进行增强。目标对象的方法需要手动执行。  
            -->  
            <aop:around method="aroundLog" pointcut="execution(\* cn.tx.\*.\*ServiceImpl.save\*(..))" />  
​  
        </aop:aspect>  
    </aop:config>  
​  
</beans>

## 第四章：Spring的AOP技术-注解方式

### 1. AOP注解方式入门程序

创建maven工程，导入坐标。编写接口，完成IOC的操作。步骤略。

编写切面类

给切面类添加注解 @Aspect，编写增强的方法，使用通知类型注解声明

package cn.tx.demo3;  
​  
import org.aspectj.lang.annotation.Aspect;  
import org.aspectj.lang.annotation.Before;  
import org.springframework.stereotype.Component;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \* 切面类  
 \*/  
@Component  // 把该类交给IOC去管理  
@Aspect     // 声明是切面类 == <aop:aspect ref="myXmlAspect">  
public class MyAnnoAspect {  
​  
    /\*\*  
     \* 通知的方法  
     \*/  
    // @Before(value = "切入点的表达式")  
    @Before(value = "execution(public \* cn.tx.demo3.OrderServiceImpl.save(..))")  
    public void log(){  
        System.out.println("增强了...");  
   }  
​  
}  
​

配置文件中开启自动代理

<aop:aspectj-autoproxy/>

编写测试

package cn.tx.test;  
​  
import cn.tx.demo2.UserService;  
import cn.tx.demo3.OrderService;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.test.context.ContextConfiguration;  
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
@RunWith(SpringJUnit4ClassRunner.class)  
@ContextConfiguration("classpath:applicationContext\_demo3.xml")  
public class Demo3 {  
​  
    @Autowired  
    private OrderService orderService;  
​  
    /\*\*  
     \* 测试  
     \*/  
    @Test  
    public void run1(){  
        orderService.save();  
   }  
​  
}  
​

### 2. 通知类型的注解

通知类型注解

​ @Before -- 前置通知

​ @AfterReturing -- 后置通知

​ @Around -- 环绕通知（目标对象方法默认不执行的，需要手动执行）

​ @After -- 最终通知

​ @AfterThrowing -- 异常抛出通知

### 3. 纯注解的方式

纯注解的方式

package cn.tx.demo3;  
​  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.EnableAspectJAutoProxy;  
​  
/\*\*  
 \* 拓薪教育 -- 腾讯课程认证机构  
 \* 樱木老师  
 \*/  
@Configuration      // 配置类  
@ComponentScan(value = "cn.tx.demo3")   // 扫描包  
@EnableAspectJAutoProxy     // 开启自动代理 == <aop:aspectj-autoproxy />  
public class SpringConfig {  
      
}  
​