切面使用@Around：环绕通知实现的AOP，有ProceedingJoinPoint类参数

1）、在d-spring-aop-aspectj-04项目中新建bao03包，将bao01包的类复制过来。具体内容如下：

接口中

```
package com.studymyself.bao03;

public interface SomeService {

    //public void doSome(String name,Integer age);

    //public Object doOther(String name,int age);

    public Object doFirst(String name,Integer age);

}
```

目标实现类中

```
package com.studymyself.bao03;

//目标方法
public class SomeServiceImpl implements SomeService {

    @Override
    public Object doFirst(String name, Integer age) {
        //要增加的功能是：本方法执行前输出执行时间，执行后提交事务
        System.out.println("==目标方法doFirst执行==");
        return name+age;
    }

}
```

切面类中

```
package com.studymyself.bao03;


import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

import java.util.Date;
```

```java
/**
 * @Aspect:这是AspectJ框架的注解
 *    作用：表示当前类是切面类
 *    切面类：用来给业务方法增加功能的类，里面有切面功能的代码
 *    位置：写在定义类的上面
 */
@Aspect
public class MyAspectJ {



    /**
     * 环绕通知定义方法规则：
     *  1、公共方法，public修饰
     *  2、必须有一个返回值，建议Object
     *  3、方法名自定义，但要见名知意
     *  4、方法有参数。固定参数类型ProceedingJoinPoint。
     *
     */

    /**
     * @Around:环绕通知
     *    属性：
     *       value：切人入点表达式
     *    位置：通知方法上面
     *    特点：
     *    1、它是功能最强的通知
     *    2、该注解的通知方法能够在目标方法的执行前和执行后增强功能
     *    3、控制目标方法是否被调用执行
     *    4、修改原来目标方法的执行结果
     *    环绕通知注解的方法就相当于是之前JDK动态代理中实现了InvocationHandler接口的invoke
方法
     *        该方法中的参数类ProceedingJoinPoint是JoinPoint类的子类，所以这个参数由
AspectJ提供值
     *    参数值中包含了目标方法的所有信息，所以这个类的内部对目标方法的信息使用反射机制获取
     *    了目标方法的Method对象，然后有该类中一个方法proceed，该方法又调用目标方法的Method
对象
     *    的invoke方法来完成目标方法的调用
     *        当我们在环绕通知方法中使用ProceedingJoinPoint参数调用proceed方法时就实现了
控制目标方法的
     *    调用，所以ProceedingJoinPoint参数相当于是Method。
     *        因为参数继承JoinPoint，所以我们可以用参数获取目标方法的信息，例如实参，编写判
断实参的条件
     *    语句来完成目标方法的执行控制
     *        就是说我们获取的动态代理对象中doFirst方法中执行的是myAround方法。
     *
     *
     * @param pjp
     */
    @Around(value = "execution(* *..SomeServiceImpl.doFirst(..))")
    public Object myAround(ProceedingJoinPoint pjp){

        //实现环绕通知
        //目标方法执行结果
        Object res= null;

        //目标方法执行前执行输出执行时间日志
```

```
        System.out.println("环绕通知：目标方法前执行，打印执行时间："+new Date());

        //调用目标方法，通过判断目标方法参数决定是否调用目标方法
        if (pjp.getArgs()!=null||pjp.getArgs().length>0||pjp.getArgs().length<5)
{
            try {
                res = pjp.proceed();
            } catch (Throwable throwable) {
                throwable.printStackTrace();
            }
        }
        //目标方法执行后执行提交事务方法
        System.out.println("环绕通知：目标方法后执行，提交事务");

        //在这里可以修改目标方法执行结果

        return res;

    }


}
```

2）在配置文件中声明对象

3）、新建一个MyTest03测试类，如下

```
package com.studymyself;

import com.studymyself.bao03.SomeService;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyTest03 {

    /**
     * 测试环绕通知的AOP实现
     */
    @Test
    public void testMyBefore(){

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

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

        //获取目标类对象或代理对象（前提这个类是实现了AOP情况下）
        SomeService proxy = (SomeService) ac.getBean("someService3");

        //直接执行接口中doFirst方法，验证是否进行了功能增强
        proxy.doFirst("zzz",0);//相当于直接执行切面类中的myAround方法
    }
}
```

测试结果如下


Tests passed: 1 of 1 test – 607 ms
E:\JAVASE1.8\bin\java.exe ...
环绕通知：目标方法前执行，打印执行时间：Mon Feb 08 02:20:56 CST 2021
==目标方法doFirst执行==
环绕通知：目标方法后执行，提交事务
Process finished with exit code 0