切面使用@AfterReturning：后置通知实现的AOP，有Object类参数

1）、在d-spring-aop-aspectj-04项目中新建bao02包，将bao01包装的复制过来。在接口中新增一个doOther方法，实现类实现它，然后在切面类中新建一个使用后置通知@AfterReturning的通知方法，切面功能是通过目标方法的返回值判断，输出doOther方法的返回值。

接口如下：

```
package com.studymyself.bao02;

public interface SomeService {

    public void doSome(String name,Integer age);

    public Object doOther(String name,int age);

}
```

实现类如下：

```
package com.studymyself.bao02;

//目标方法
public class SomeServiceImpl implements SomeService {
    @Override
    public void doSome(String name,Integer age) {
        //添加新功能：（目标方法执行之前）输出doSome方法执行时间
        System.out.println("==目标方法doSome执行==");
    }

    @Override
    public Object doOther(String name, int age) {
        //添加新功能：（目标方法执行之后）根据该方法的返回值做出相应处理
        return name+age;
    }
}
```

切面类如下

```
package com.studymyself.bao02;


import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
```

```
import java.sql.ResultSet;
import java.util.Date;

/**
 * @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 {

    /**
     * @Before前置通知
     * @param jp
     */
    @Before(value = "execution( public void *..SomeServiceImpl.doSome(..))")
    public void myBefore(JoinPoint jp){

        //获取方法否完整定义
        System.out.println("方法的签名（定义）="+jp.getSignature());
        System.out.println("方法名="+jp.getSignature().getName());

        //获取方法的实参，有两个，一个返回的是Object数组，一个返回的是第一个实参
        Object[] args = jp.getArgs();
        for (Object arg:args){
            System.out.println("参数="+arg);
        }

        //增加的功能，即切面代码
        System.out.println("使用@Before前置通知的切面功能：输出目标方法执行时间\n"+new
Date());
    }

    /**
     * 后置通知定义方法规则：
     *  1、公共方法，public修饰
     *  2、无返回值，void
     *  3、方法名自定义，但要见名知意
     *  4、方法有参数。建议Object，参数名自定义。
     *  从@AfterReturning就可以看出，其作用就是得等到目标方法执行后返回一个值
     *  然后后置通知方法才能把这个值作为其参数，然后功能代码中就有对返回值的条件
     *  语句判断，进行实现相应功能
     */

    /**
     * @AfterReturning:后置通知
     *       属性：
     *       value：切入入点表达式
     *       returning：自定义的变量名，存储目标方法返回值的。
     *       所以下面通知方法中的形参名就得和这个变量名的名称一致
     *    位置：通知方法上面
     *    特点：
     *    1、目标方法执行后执行
     *    2、能够获取目标方法的返回值，通知方法中可通过条件语句用该返回值
     *       进行不同的功能实现
     *    3、可以修改这个返回值。但是能否影响到目标方法的返回值可以根据自己之前学过的
     *       参数传递进行思考。返回值是一个值，存储在栈内存，一个方法一个栈，肯定无法影响
```

```
     *       返回值是一个引用，保存的是一个内存地址，内存地址唯一指向堆中的对象，可以说内存地址就
     *       是这个对象，这个引用就是这个对象，我们把另一个对象的内存地址赋给这个引用后，之前的内存
     *       地址就没了，也就是说返回值的内存地址变了，对象就变了。而这里的Object
     * @param res
     */
    @AfterReturning(value = "execution(*
*..SomeServiceImpl.doOther(..))",returning = "res")
    public void myAfterReturning(Object res){
        //Object res;是目标方法执行后的返回值，根据返回值做切面功能的不同处理
        if (res!=null){

            System.out.println("后置通知：添加对返回值的判断做出的功能，输出返回
值："+res);

        }else {
            System.out.println("参数为空");
        }

    }


}
```

2）、在配置文件中声明对象

3）、新建测试了MyTest02，如下

```
package com.studymyself;

import com.studymyself.bao02.SomeService;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyTest02 {

    /**
     * 测试后置通知的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("someService2");

        //直接执行接口中doSome方法，验证是否进行了功能增强
```

```
        proxy.doOther("aaa",0);
    }
}
```