切面使用@AfterThrowing：异常通知实现的AOP，有Exception类参数

（了解）

1）、在d-spring-aop-aspectj-04项目中新建bao04包，将bao01包的类复制过来。具体内容如下：

接口中

```
package com.studymyself.bao04;

public interface SomeService {

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

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

    //public Object doFirst(String name,Integer age);

    public void doSecond(Integer a,Integer b);

}
```

目标实现类中

```
package com.studymyself.bao04;

//目标方法
public class SomeServiceImpl implements SomeService {

    @Override
    public void doSecond(Integer a, Integer b) {
        //切面功能：目标方法出异常后，通过邮件、短信等方式发送异常信息到开发人员
        System.out.println("==doSecond方法执行=="+a/b);
    }
}
```

切面类中

```
package com.studymyself.bao04;


import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
```

```
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;

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 {



    /**
     * 异常通知定义方法规则：
     *  1、公共方法，public修饰
     *  2、无返回值
     *  3、方法名自定义，但要见名知意
     *  4、方法有参数。参数类型Exception，还有就是JoinPoint（但要放在参数第一位置）
     *
     */

    /**
     * @AfterThrowing:异常通知
     *    属性：
     *       value：切人入点表达式
     *       throwing：名称自定义，存储的是目标方法抛出的异常对象，
     *                   参数中形参名要和该名称一致
     *    位置：通知方法上面
     *    特点：
     *    1、在目标方法抛出异常时执行
     *    2、可以做异常的监控，监控目标执行时是不是有异常
     *       如果有异常就通过邮件或短信等方式发送异常
     *
     *       执行的就是：
     *       try{
     *           someServiceImpl.doSecond(..)
     *       }catch{
     *           myAfterThrowing(..)
     *       }
     */
    @AfterThrowing(value = "execution(*
*..SomeServiceImpl.doSecond(..))",throwing = "ex")
    public void myAfterThrowing(Exception ex){

        System.out.println("异常通知：目标方法发生异常执行，异常信
息："+ex.getMessage());
        //下面语句模拟发送异常
        try {
            Thread.sleep(1000*3);
            System.out.println("登录默认邮箱中...");
            Thread.sleep(1000*3);
            System.out.println("邮箱登录成功！");
            Thread.sleep(1000*3);
            System.out.println("选择默认收件人...");
```

```
            Thread.sleep(1000*3);
            System.out.println("正在发送异常信息邮件中...");
            Thread.sleep(1000*3);
            System.out.println("发送成功！");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }


}
```

2）在配置文件中声明对象

3）、新建一个MyTest04测试类，如下

```
package com.studymyself;

import com.studymyself.bao04.SomeService;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyTest04 {

    /**
     * 测试异常通知的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("someService4");

        //直接执行接口中doSecond方法，验证是否进行了功能增强
        proxy.doSecond(5,6);
    }
}
```

由于先前把bao01包装的切面类中的通知方法上的切入点表达式写成了SomeServiceImpl的任意方法，导致测试的时候所有目标方法执行前都执行了bao01中切面类的功能

还有最后一个通知注解@After，其通知方法是没有返回值的，参数有的话就是JoinPoint，特点是在目标方法执行后执行，而且总是会执行的，因为在目标方法执行后执行，可能目标方法出异常后面的代码是执行不了的，但是该注解的通知方法无需顾及这个，所以一般做资源关闭的功能。就是捕捉的trycatch语句块中的finally语句块，无论怎样该语句块的内容都会执行

下面演示一个辅助注解@Pointcut的功能

```
package com.studymyself;


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

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 {

    /**
     * @Pointcut注解：定义和切入管理点。
     *       当项目中的多个切入点表达式的内容是重复的，或者说的可以复用的。
     *       就可以使用@Pointcut
     *      属性：
     *      value：可以复用的切入点表达式
     *      位置：写在自定义的一个私有无参数的方法上，该方法无需内容
     *    特点：
     *       该注解定义在一个方法上，这时该方法的“方法名()”就代表了该注解中的切入点表达式
     *       所以在通知注解中的value属性的值就可以直接填“方法名()”，非常便捷
     */

    //写一个自定义方法，建议私有的无参数的，然后添加@Pointcut
    @Pointcut(value = "execution(* *..SomeServiceImpl.doOther(..))")
    private void myExecution(){
        //无需代码
    }

    //@Before(value = "execution(* *..SomeServiceImpl.doOther(..))")
    @Before(value = "myExecution()")
    public void myBefore(JoinPoint jp){
```

```
        //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());
    }


    //@AfterReturning(value = "execution(*
*..SomeServiceImpl.doOther(..))",returning = "res")
    @AfterReturning(value = "myExecution()",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");
        }

    }


}
```

一般情况下，当目标类有接口，AspectJ默认使用JDK动态代理，当没有接口能够继承就会使用CGLIB动态代理。如果当目标类有接口，还能继承的情况下，希望框架使用CGLIB动态代理时，在pom文件中的声明自动代理生成器标签后面添加生成代理目标对象类型属性，然后值使用true就行了。

```
<aop:aspectj-autoproxy proxy-target-class="true"/>
```