
知识点列表

编号	名称	描述	级别
1	什么是 AOP 及其好处	了解 AOP 是什么，掌握使用 Aop 编程步骤	*
2	AOP 相关概念	理解 Aop 思想	**
3	通知类型	掌握常用的通知类型，共 5 种	**
4	切入点表达式	理解切入点表达式，能看懂，会使用	*

注： "*"理解级别 "**"掌握级别 "***"应用级别

目录

1. 什么是 AOP 及其好处 *	3
2. AOP 相关概念 *	3
【案例 1】AOP 演示 **	3
3. 通知类型 **	13
【案例 2】使用通知类型 **	14
4. 切入点表达式 *	30
【课堂练习 1】记录异常日志 **	33

1. 什么是 AOP 及其好处 *

Aspect Oriented Programming 面向方面编程或面向切面编程。

AOP 关注点是共同处理，可以通过配置将其作用到某一个或多个目标对象上。好处是实现组件重复利用，改善程序结构，提高灵活性。将共通组件与目标对象解耦。

2. AOP 相关概念 *

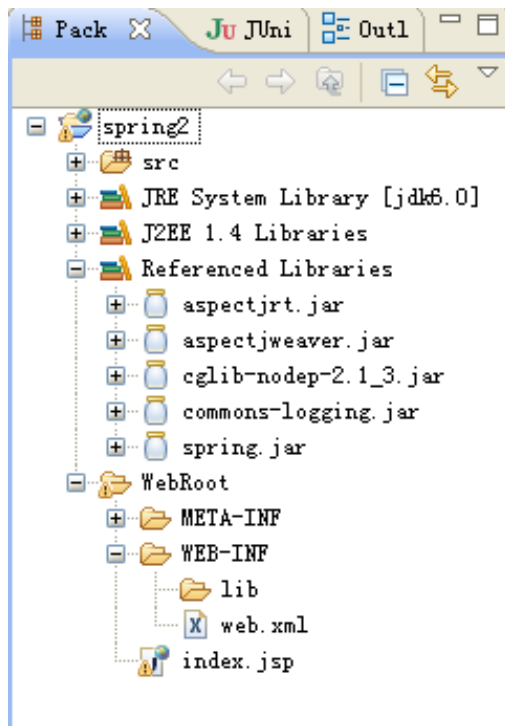
- 1) Aspect 切面（方面）
指的是共通业务处理，可以切入到多个目标对象，可多次使用
- 2) JoinPoint 连接点
指的是切面组件在目标对象上作用的位置，
例如：方法上或者发生异常。
- 3) Pointcut 切入点
切入点是连接点的集合，采用表达式指定
- 4) Target Object 目标对象
- 5) Advice 通知
指的是切面组件在连接点上执行的动作。
例如：在方法调用前、方法调用后、方法调用前后等。
- 6) AutoProxy 动态代理
采用了 AOP 之后，容器返回的对象是代理对象。用户在使用时，由代理对象调用切面组件和目标对象的功能。
 - a. 目标对象有接口采用 JDK 代理、
 - b. 目标对象没有接口采用 CGLIB 代理

【案例 1】AOP 演示 **

- 1) 新建工程 spring2
- 2) 导入 Jar 包

AoP 需要的 Jar 包：aspectjrt.jar 和 aspectjweaver.jar

动态代理需要 cglib.jar



3) 新建接口 UserService

```
package tarena.service;

public interface UserService {
    public void update();
    public void delete();
    public void save();
}
```

4) 新建实现类 UserServiceImpl

```
package tarena.service;

public class UserServiceImpl implements UserService {

    public void delete() {System.out.println("删除用户信息");}
    public void save() {System.out.println("保存用户信息");}
    public void update() {System.out.println("更新用户信息");}
}
```

5) 新建 aop.xml

```
<?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-2.5.xsd
                           http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
                           http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd" >

    <bean id="userservice" class="tarena.service.UserServiceImpl" >
    </bean>

</beans>
```

6) 新建 Test

```
package tarena.service;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {
    private static final String CONFIG = "aop.xml";
    /**
     * @param args
     */
    public static void main(String[] args) {
        ApplicationContext ac =
            new ClassPathXmlApplicationContext(CONFIG);
        UserService userService = (UserService)ac.getBean("userservice");

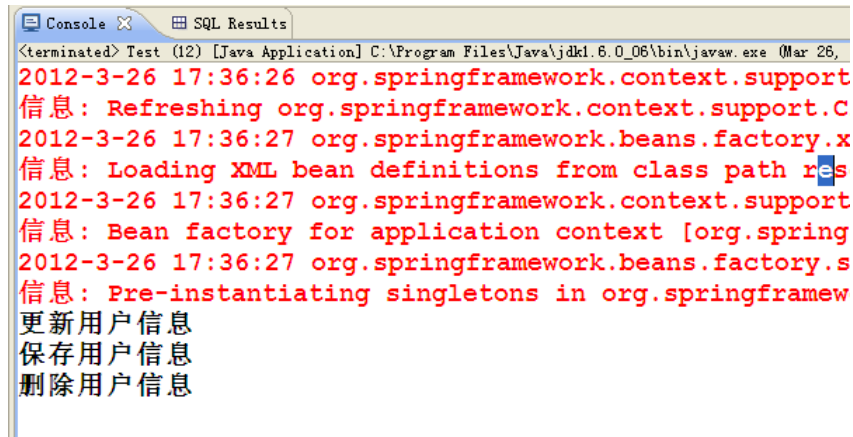
        userService.update();
        userService.save();
    }
}
```

```

        userService.delete();
    }
}

```

7) 运行 Test



```

Console [X] SQL Results
<terminated> Test (12) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 26,
2012-3-26 17:36:26 org.springframework.context.support
信息: Refreshing org.springframework.context.support.C
2012-3-26 17:36:27 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-26 17:36:27 org.springframework.context.support
信息: Bean factory for application context [org.spring
2012-3-26 17:36:27 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframe
更新用户信息
保存用户信息
删除用户信息

```

如上，已经将UserService纳入Spring容器的管理中。

我们现在有这样的需求，要为UserSerice的操作增加日志记录功能。

我们需要为更新、保存、删除操作增加记录日志功能，那么[记录日志功能就属于切面功能](#)。

增加AoP功能

8) 新建 aop.OptLogger

在 Spring 中，切面组件只要是普通的 bean 即可。

OptLogger 是[记录操作日志的切面组件](#)

```

package tarena.aop;

/**
 * 切面组件，记录操作日志
 * @author tarena
 *
 */
public class OptLogger {
    public void logger(){
        System.out.println("记录操作日志了...");
    }
}

```

9) 修改 aop.xml

将切面组件 OptLogger 加入到配置文件中；配置 aop

```
<?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-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

    <bean id="userService" class="tarena.service.UserServiceImpl"></bean>
    <bean id="optlogger" class="tarena.aop.OptLogger"></bean>

    <aop:config>
        <aop:pointcut id="servicepointcut"
                     expression="execution(* tarena.service.*(..))" />
        <aop:aspect id="loggeraspect" ref="optlogger">
            <aop:before method="logger" pointcut-ref="servicepointcut"/>
        </aop:aspect>
    </aop:config>
</beans>
```

<aop:aspect > 用于配置切面

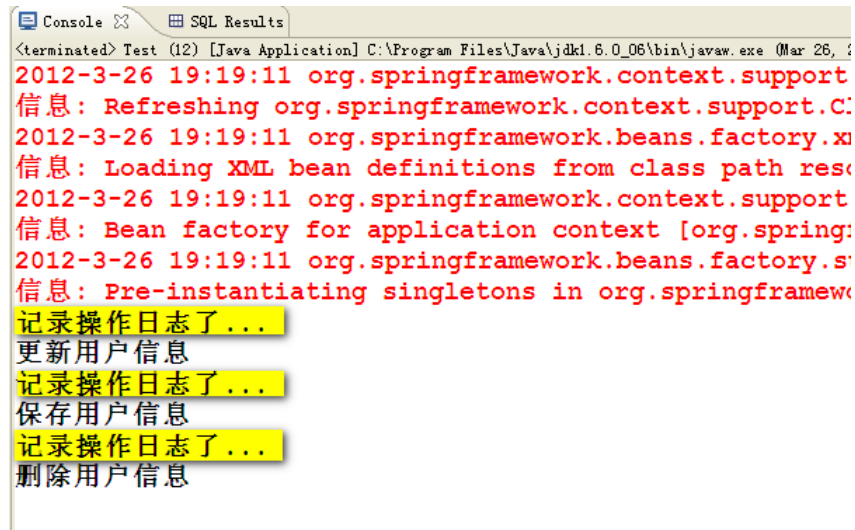
- ✓ id属性
- ✓ ref属性 用于关联切面的bean

<aop:pointcut/>用于设置切入点

- ✓ expression属性 类似于正则形式的表达式（后期讲，先使用）
execution()用于设置方法限定
- ✓ execution(* tarena.service.*(..))
表示不限定返回类型，限定指定tarena.service包下的
所有方法，不限定参数类型
- ✓ <aop:before> 表示采用before这种通知，作用于pointcut和方法上

10) 运行 Test

在操作之前都进行了记录日志操作通知



```
<terminated> Test (12) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 26, 2012 19:19:11)
2012-3-26 19:19:11 org.springframework.context.support
信息: Refreshing org.springframework.context.support.C
2012-3-26 19:19:11 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-26 19:19:11 org.springframework.context.support
信息: Bean factory for application context [org.spring:
2012-3-26 19:19:11 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
记录操作日志了...
更新用户信息
记录操作日志了...
保存用户信息
记录操作日志了...
删除用户信息
```

如果这样会在操作后记录日志

11) 修改 aop.xml

```
<bean id="userservice" class="tarena.service.UserServiceImpl"></bean>
<bean id="optlogger" class="tarena.aop.OptLogger"></bean>

<aop:config>
    <aop:pointcut id="servicepointcut"
        expression="execution(* tarena.service.*(..))" />
    <aop:aspect id="loggeraspect" ref="optlogger">
        <aop:before method="logger" pointcut-ref="servicepointcut"/>
    </aop:aspect>
</aop:config>
</beans>
```

12) 运行 Test


```
Console X SQL Results
<terminated> Test (12) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 26, 2
2012-3-26 19:42:38 org.springframework.context.support.Cl
信息: Refreshing org.springframework.context.support.Cl
2012-3-26 19:42:38 org.springframework.beans.factory.xr
信息: Loading XML bean definitions from class path resc
2012-3-26 19:42:38 org.springframework.context.support
信息: Bean factory for application context [org.springf
2012-3-26 19:42:38 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
更新用户信息
记录操作日志了...
保存用户信息
记录操作日志了...
删除用户信息
记录操作日志了...
```

我们如果想将操作的内容在日志中做一些记录，该怎么做？

13) 修改 aop.xml

```
<bean id="userservice" class="tarena.service.UserServiceImpl"></bean>
<bean id="optlogger" class="tarena.aop.OptLogger"></bean>

<aop:config>
    <aop:pointcut id="servicepointcut"
        expression="execution(* tarena.service.*(..))" />
    <aop:aspect id="loggeraspect" ref="optlogger">
        <aop:around method="logger" pointcut-ref="servicepointcut"/>
    </aop:aspect>
</aop:config>
</beans>
```

14) 修改 OptLogger

当 aop.xml 中设置为<aop:around>通知形式后，我们可以通过 ProceedingJoinPoint 对象（连接点）来获取方法名、类等。

```
package tarena.aop;

import org.aspectj.lang.ProceedingJoinPoint;

/**
 * 切面组件，记录操作日志
```

```
* @author tarena
*
*/
public class OptLogger {

    public Object logger(ProceedingJoinPoint pjp) throws Throwable{

        //proceed()方法有执行目标对象的功能
        Object obj = pjp.proceed();

        //获取方法名
        String method =
            pjp.getSignature().getName();
        //获取目标对象类名
        String clazzName =
            pjp.getTarget().getClass().getName();

        System.out.println(
            "执行了" + clazzName + "的" + method + "方法");

        return obj;
    }
}
```

15) 运行 Test

```
Console X SQL Results
<terminated> Test (12) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 26, 2012)
2012-3-26 20:00:59 org.springframework.context.support
信息: Refreshing org.springframework.context.support.C
2012-3-26 20:00:59 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-26 20:00:59 org.springframework.context.support
信息: Bean factory for application context [org.spring
2012-3-26 20:00:59 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
更新用户信息
执行了 tarena.service.UserServiceImpl 的 update 方法
保存用户信息
执行了 tarena.service.UserServiceImpl 的 save 方法
删除用户信息
执行了 tarena.service.UserServiceImpl 的 delete 方法
```

我们可以做的更用户友好些

16) 新建 opt.properties

内容是更友好的一些文字描述

```
tarena.service.UserServiceImpl.update=\u7528\u6237\u66f4\u65b0\u64cd\u4f5c
tarena.service.UserServiceImpl.save=\u7528\u6237\u4fdd\u5b58\u64cd\u4f5c
tarena.service.UserServiceImpl.delete=\u7528\u6237\u5220\u9664\u64cd\u4f5c
```

17) 新建 PropertiesUtil

```
package tarena.util;

import java.io.IOException;
import java.util.Properties;

public class PropertiesUtil {

    static Properties props = new Properties();

    private PropertiesUtil(){}

    public static Properties getInstance(String path)
    throws IOException{
        props.load(
            PropertiesUtil.class.getClassLoader()
```

```

        .getResourceAsStream(path));

    return props;
}

public static String getProperty(String key){
    String val = "";
    if(props != null){
        String prop = props.getProperty(key);
        if(prop != null){
            val = prop;
        }
    }
    return val;
}
}

```

18) 修改 OptLogger

```

package tarena.aop;

import org.aspectj.lang.ProceedingJoinPoint;

import tarena.util.PropertiesUtil;

/**
 * 切面组件，记录操作日志
 * @author tarena
 *
 */
public class OptLogger {
    public Object logger(ProceedingJoinPoint pjp)
        throws Throwable{
        Object obj = pjp.proceed();//执行目标对象的功能

        String methodName = pjp.getSignature().getName();
        String clazzName = pjp.getTarget().getClass().getName();

        PropertiesUtil.getInstance("tarena/opt.properties");
    }
}

```

```

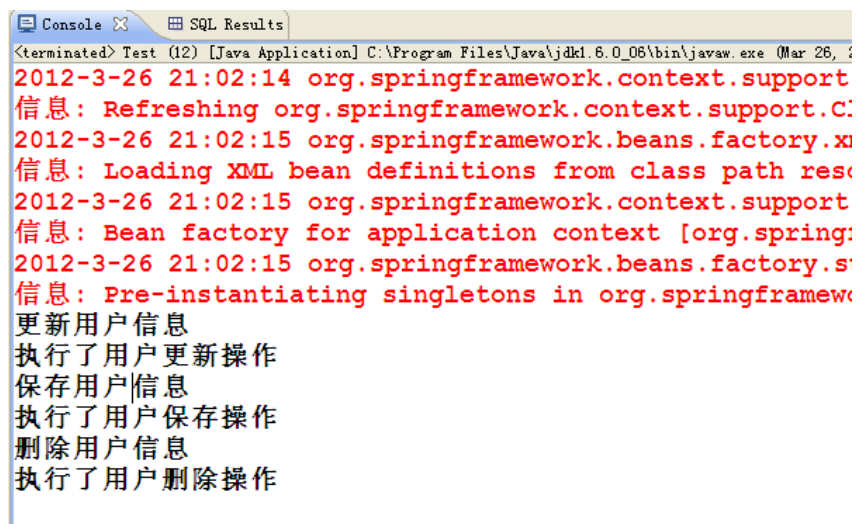
        String key = clazzName+"."+methodName;

        System.out.println(
            "执行了"+PropertiesUtil.getProperty(key));

        return obj;
    }
}

```

19) 运行 Test



The screenshot shows a console window with the following content:

```

<terminated> Test (12) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 26, 2012 21:02:14)
2012-3-26 21:02:14 org.springframework.context.support.ClassPathXmlApplicationContext: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext
2012-3-26 21:02:15 org.springframework.beans.factory.xml.XmlBeanDefinitionReader: Loading XML bean definitions from class path resource[s]
2012-3-26 21:02:15 org.springframework.context.support.ClassPathXmlApplicationContext: Bean factory for application context [org.springframework.beans.factory.xml.XmlBeanDefinitionReader]
2012-3-26 21:02:15 org.springframework.beans.factory.support.DefaultListableBeanFactory: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory
更新用户信息
执行了用户更新操作
保存用户信息
执行了用户保存操作
删除用户信息
执行了用户删除操作

```

如上案例，我们可以了解到IoC是解决两个对象之间的关系，AOP是解决一个对象和某一批对象之间的关系。

(案例结束)

3. 通知类型 **

- 1) 前置通知
 - <aop:before>
 - 在目标方法调用之前执行。不能阻止后续执行，除非抛异常
- 2) 后置通知
 - <aop:after-returning>
 - 在目标方法调用之后执行。目标方法正常结束才执行。
- 3) 最终通知
 - <aop:after>

在目标方法调用之后执行。目标方法正常或异常都执行。

4) 异常通知

<aop:after-throwing>

在目标方法调用发生异常之后执行。

5) 环绕通知

<aop:around>

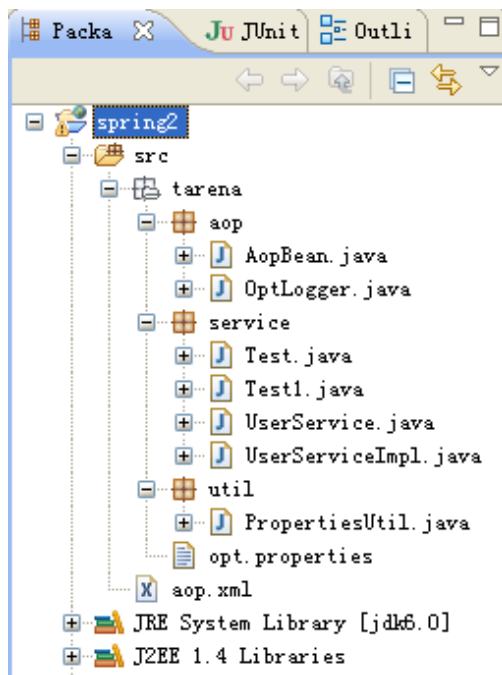
在目标方法调用之前和之后执行。

这 5 种类型的通知，在内部调用时这样组织

```
try{
    调用前置通知
    环绕前置处理
    调用目标对象方法
    环绕后置处理
    调用后置通知
}catch(Exception e){
    调用异常通知
}finally{
    调用最终通知
}
```

【案例 2】使用通知类型 **

1) 使用工程 spring2



2) 前置通知

a. 新建 AopBean

```
package tarena.aop;

public class AopBean {
    //前置通知方法
    public void mybefore(){
        System.out.println("--前置通知--");
    }
}
```

b. 修改 aop.xml

```
<?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-2.5.xsd
http://www.springframework.org/schema/context
```

```

http://www.springframework.org/schema/context/spring-context-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

<bean id="userservice" class="tarena.service.UserServiceImpl"></bean>
<bean id="aopbean" class="tarena.aop.AopBean"></bean>

<aop:config>
    <aop:pointcut id="servicepointcut"
        expression="within (tarena.service.*)" />

    <aop:aspect id="aspectbean" ref="aopbean">
        <aop:before method="mybefore" pointcut-ref="servicepointcut"/>
    </aop:aspect>
</aop:config>
</beans>

```

c. 新建 Test1

```

package tarena.service;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test1 {
    private static final String CONFIG = "aop.xml";
    /**
     * @param args
     */
    public static void main(String[] args) {
        try{
            ApplicationContext ac =
                new ClassPathXmlApplicationContext(CONFIG);
            UserService userService =
                (UserService)ac.getBean("userservice");
            userService.update();
            userService.save();
            userService.delete();
        }catch(Exception e){

```

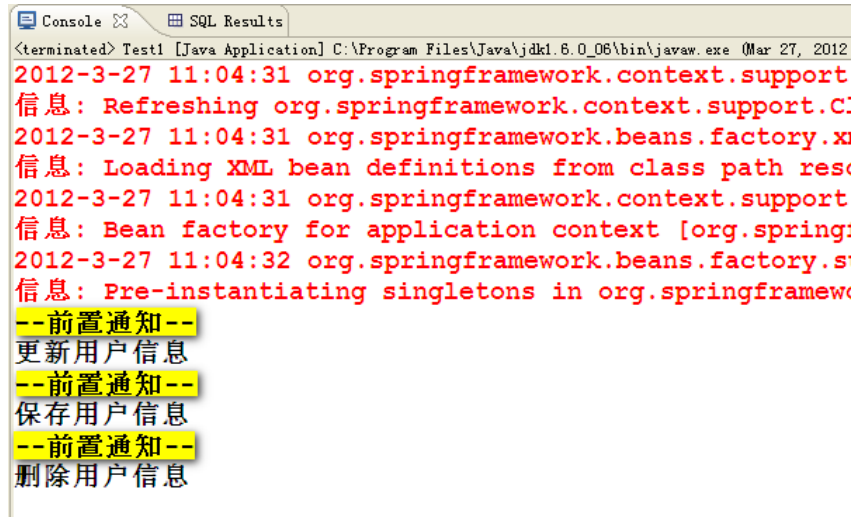


```

    }
}
}

```

d. 运行 Test



```

Console [SQL Results]
<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.8.0_06\bin\javaw.exe (Mar 27, 2012)
2012-3-27 11:04:31 org.springframework.context.support
信息: Refreshing org.springframework.context.support.C
2012-3-27 11:04:31 org.springframework.beans.factory.xml
信息: Loading XML bean definitions from class path res
2012-3-27 11:04:31 org.springframework.context.support
信息: Bean factory for application context [org.spring
2012-3-27 11:04:32 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframe
--前置通知--
更新用户信息
--前置通知--
保存用户信息
--前置通知--
删除用户信息

```

3) 后置通知

a. 修改 AopBean

```

package tarena.aop;

public class AopBean {
    //前置通知方法
    public void mybefore(){
        System.out.println("--前置通知--");
    }

    //后置通知方法
    public void myafterReturning(){
        System.out.println("--后置通知--");
    }
}

```

b. 修改 aop.xml

```

<bean id="userservice"
    class="tarena.service.UserServiceImpl"> </bean>

```

```

<bean id="aopbean" class="tarena.aop.AopBean"></bean>

<aop:config>
    <aop:pointcut id="servicepointcut"
        expression="within (tarena.service.*)"/>

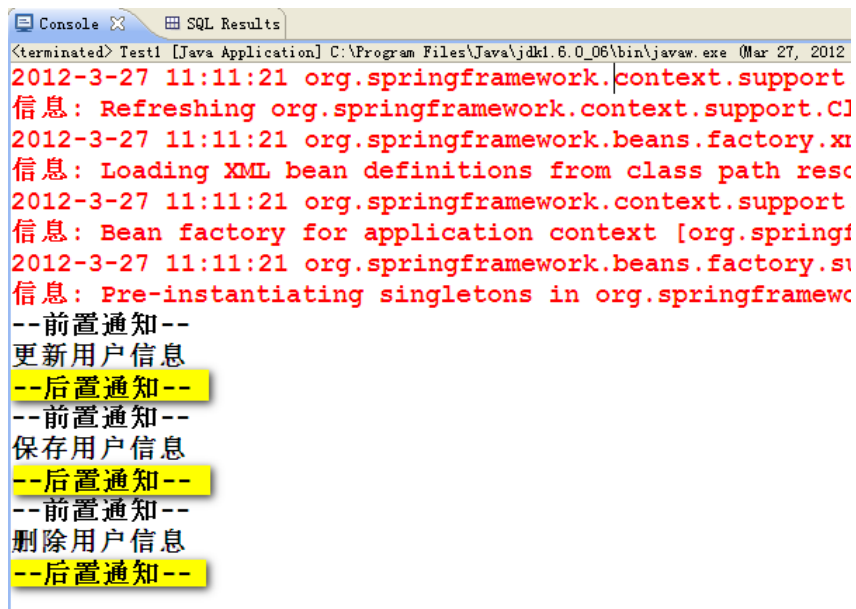
    <aop:aspect id="aspectbean" ref="aopbean">
        <aop:before method="mybefore"
            pointcut-ref="servicepointcut"/>

        <aop:after-returning method="myafterReturning"
            pointcut-ref="servicepointcut"/>

    </aop:aspect>
</aop:config>

```

c. 运行 Test1



```

<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2012
2012-3-27 11:11:21 org.springframework.context.support.Cl
信息: Refreshing org.springframework.context.support.Cl
2012-3-27 11:11:21 org.springframework.beans.factory.xi
信息: Loading XML bean definitions from class path resc
2012-3-27 11:11:21 org.springframework.context.support
信息: Bean factory for application context [org.springfi
2012-3-27 11:11:21 org.springframework.beans.factory.si
信息: Pre-instantiating singletons in org.springframework
--前置通知--
更新用户信息
--后置通知--
--前置通知--
保存用户信息
--后置通知--
--前置通知--
删除用户信息
--后置通知--

```

需要注意的是：

如是正常执行程序情况下，会提示通知；如果目标方法出现异常，将不会执行后置通知

接下来我们添加一些异常，模拟出现异常状况

d. 修改 UserServiceImpl

```
package tarena.service;

public class UserServiceImpl implements UserService {

    public void delete() {
        System.out.println("删除用户信息");

        //模拟NullPointerException
        String s = null;
        s.length();
    }

    public void save() {
        System.out.println("保存用户信息");
    }

    public void update() {
        System.out.println("更新用户信息");
    }
}
```

e. 运行 Test

后置通知是在运行目标方法执行成功后调用的，如果目标方法执行失败（出现异常），后置通知将不被调用

```
Console  SQL Results
<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2012
2012-3-27 11:22:11 org.springframework.context.support
信息: Refreshing org.springframework.context.support.C
2012-3-27 11:22:11 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-27 11:22:12 org.springframework.context.support
信息: Bean factory for application context [org.spring:
2012-3-27 11:22:12 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
--前置通知--
更新用户信息
--后置通知--
--前置通知--
保存用户信息
--后置通知--
--前置通知--
删除用户信息
```

除此特点外，后置通知可以得到目标方法的返回值

f. 修改 UserServiceImpl

```
package tarena.service;

public class UserServiceImpl implements UserService {

    public void delete() {
        System.out.println("删除用户信息");

        //模拟NullPointerException
        // String s = null;
        // s.length();
    }

    public boolean save() {
        System.out.println("保存用户信息");
        return true;
    }

    public void update() {
        System.out.println("更新用户信息");
    }
}
```

g. 修改 AopBean

```
package tarena.aop;

public class AopBean {
    //前置通知方法
    public void mybefore(){
        System.out.println("--前置通知--");
    }

    //后置通知方法
    public void myafterReturning(Object retVal){
        System.out.println("--后置通知--" + retVal);
    }
}
```

h. 修改 aop.xml

```
<bean id="userservice"
      class="tarena.service.UserServiceImpl"></bean>
<bean id="aopbean" class="tarena.aop.AopBean"></bean>

<aop:config>
    <aop:pointcut id="servicepointcut"
        expression="within (tarena.service.*)" />

    <aop:aspect id="aspectbean" ref="aopbean">
        <aop:before method="mybefore"
            pointcut-ref="servicepointcut"/>

        <aop:after-returning method="myafterReturning"
            returning="retVal"
            pointcut-ref="servicepointcut"/>

    </aop:aspect>
</aop:config>
```

i. 运行 Test1

```

Console X SQL Results
<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2012
2012-3-27 11:34:28 org.springframework.context.support
信息: Refreshing org.springframework.context.support.C
2012-3-27 11:34:28 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-27 11:34:28 org.springframework.context.support
信息: Bean factory for application context [org.spring
2012-3-27 11:34:28 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframe
--前置通知--
更新用户信息
--后置通知--null
--前置通知--
保存用户信息
--后置通知--true
--前置通知--
删除用户信息
--后置通知--null

```

4) 异常通知

a. 修改 AopBean

```

package tarena.aop;

public class AopBean {
    //前置通知方法
    public void mybefore(){
        System.out.println("--前置通知--");
    }

    //后置通知方法
    public void myafterReturning(Object retVal){
        System.out.println("--后置通知--" + retVal);
    }

    //异常通知方法
    public void myafterException(){
        System.out.println("--异常通知--");
    }
}

```

b. 修改 aop.xml

```
<bean id="userservice"
      class="tarena.service.UserServiceImpl"></bean>
<bean id="aopbean" class="tarena.aop.AopBean"></bean>

<aop:config>
  <aop:pointcut id="servicepointcut"
    expression="within (tarena.service.*)" />

  <aop:aspect id="aspectbean" ref="aopbean">
    <aop:before method="mybefore"
      pointcut-ref="servicepointcut"/>

    <aop:after-returning method="myafterReturning"
      returning="retVal"
      pointcut-ref="servicepointcut"/>

    <aop:after-throwing method="myafterException"
      pointcut-ref="servicepointcut"/>

  </aop:aspect>
</aop:config>
```

c. 修改 UserServiceImpl

```
package tarena.service;

public class UserServiceImpl implements UserService {

  public void delete() {
    System.out.println("删除用户信息");

    //模拟NullPointerException
    String s = null;
    s.length();
  }

  public boolean save() {
```

```

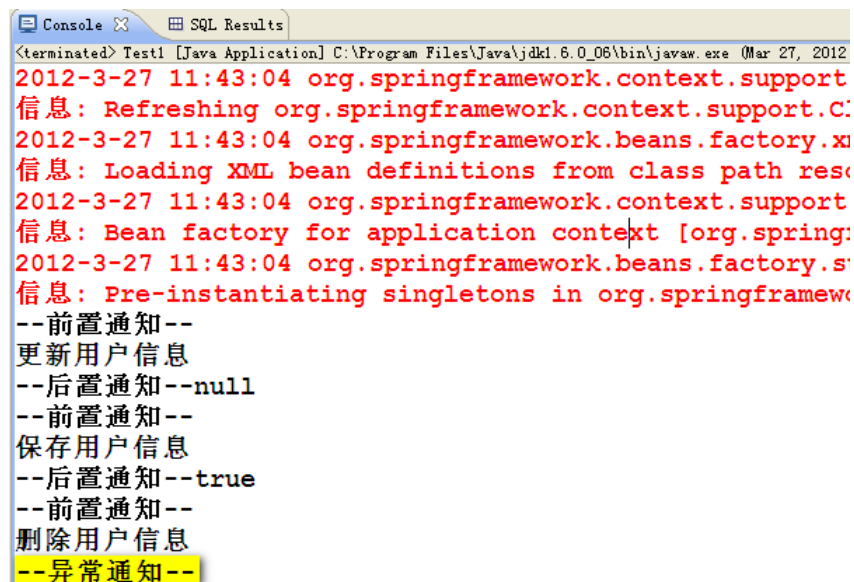
        System.out.println("保存用户信息");
        return true;
    }

    public void update() {
        System.out.println("更新用户信息");
    }
}

```

d. 运行 Test

因为在执行目标方法 delete()时发生了异常，所以触发了异常通知



```

<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2012
2012-3-27 11:43:04 org.springframework.context.support
信息: Refreshing org.springframework.context.support.C
2012-3-27 11:43:04 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-27 11:43:04 org.springframework.context.support
信息: Bean factory for application context [org.spring
2012-3-27 11:43:04 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
--前置通知--
更新用户信息
--后置通知--null
--前置通知--
保存用户信息
--后置通知--true
--前置通知--
删除用户信息
--异常通知--

```

e. 修改 UserServiceImpl

如果我们在 delete()方法中捕获了异常。

```

package tarena.service;

public class UserServiceImpl implements UserService {

    public void delete() {
        System.out.println("删除用户信息");

        try {

```



```

        //模拟NullPointerException
        String s = null;
        s.length();
    } catch (Exception e) {
    }
}

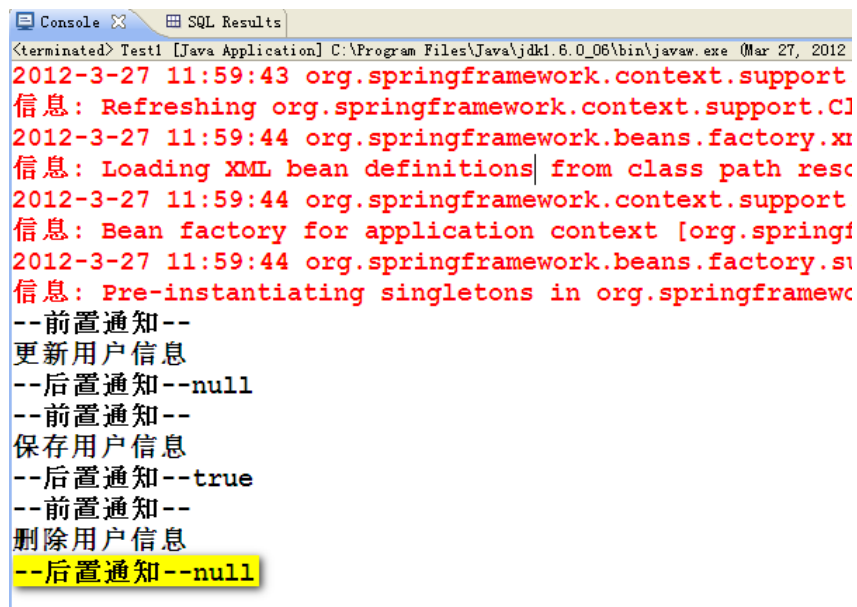
public boolean save() {
    System.out.println("保存用户信息");
    return true;
}

public void update() {
    System.out.println("更新用户信息");
}
}

```

f. 运行 Test1

如果我们在执行 delete()方法时捕获了异常，异常通知就不会执行



```

Console  SQL Results
<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2012
2012-3-27 11:59:43 org.springframework.context.support
信息: Refreshing org.springframework.context.support.Cl
2012-3-27 11:59:44 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-27 11:59:44 org.springframework.context.support
信息: Bean factory for application context [org.springf
2012-3-27 11:59:44 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
--前置通知--
更新用户信息
--后置通知--null
--前置通知--
保存用户信息
--后置通知--true
--前置通知--
删除用户信息
--后置通知--null

```

和后置通知相同，我们可以获取异常对象

g. 修改 AopBean

```

package tarena.aop;

public class AopBean {
    //前置通知方法
    public void mybefore(){
        System.out.println("--前置通知--");
    }

    //后置通知方法
    public void myafterReturning(Object retVal){
        System.out.println("--后置通知--" + retVal);
    }

    //异常通知方法
    public void myafterException(Exception ex){
        System.out.println("--异常通知begin--");
        ex.printStackTrace();
        System.out.println("--异常通知end--");
    }
}

```

h. 修改 aop.xml

```

<bean id="userservice"
    class="tarena.service.UserServiceImpl"></bean>
<bean id="aopbean" class="tarena.aop.AopBean"></bean>

<aop:config>
    <aop:pointcut id="servicepointcut"
        expression="within (tarena.service.*)" />

    <aop:aspect id="aspectbean" ref="aopbean">
        <aop:before method="mybefore"
            pointcut-ref="servicepointcut"/>

        <aop:after-returning method="myafterReturning"
            returning="retVal"
            pointcut-ref="servicepointcut"/>
    </aop:aspect>
</aop:config>

```

```
<aop:after-throwing method="myafterException"
    throwing="ex"
    pointcut-ref="servicepointcut"/>

</aop:aspect>
</aop:config>
```

i. 修改 UserServiceImpl

```
package tarena.service;

public class UserServiceImpl implements UserService {

    public void delete() {
        System.out.println("删除用户信息");

        //模拟NullPointerException
        String s = null;
        s.length();
    }

    public boolean save() {
        System.out.println("保存用户信息");
        return true;
    }

    public void update() {
        System.out.println("更新用户信息");
    }
}
```

j. 运行 Test1

```
Console  SQL Results
<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2012)
--前置通知--
删除用户信息
--异常通知begin--
java.lang.NullPointerException
    at tarena.service.UserServiceImpl.delete(Users
    at sun.reflect.NativeMethodAccessorImpl.invoke
    at sun.reflect.NativeMethodAccessorImpl.invoke
    at sun.reflect.DelegatingMethodAccessorImpl.in
    at java.lang.reflect.Method.invoke(Method.java
    at org.springframework.aop.support.AopUtils.in
    at org.springframework.aop.framework.Reflectiv
    at org.springframework.aop.framework.Reflectiv
    at org.springframework.aop.framework.adapter.A
    at org.springframework.aop.framework.Reflectiv
    at org.springframework.aop.aspectj.AspectJAfte
    at org.springframework.aop.framework.Reflectiv
    at org.springframework.aop.framework.adapter.M
    at org.springframework.aop.framework.Reflectiv
    at org.springframework.aop.interceptor.ExposeI
    at org.springframework.aop.framework.Reflectiv
    at org.springframework.aop.framework.JdkDynami
    at $Proxy0.delete(Unknown Source)
    at tarena.service.Test1.main(Test1.java:19)
--异常通知end--
```

5) 最终通知

a. 修改 AopBean

```
package tarena.aop;

public class AopBean {
    //前置通知方法
    public void mybefore(){
        System.out.println("--前置通知--");
    }

    //后置通知方法
    public void myafterReturning(Object retVal){
        System.out.println("--后置通知--" + retVal);
    }

    //异常通知方法
    public void myafterException(Exception ex){
```

```

        System.out.println("--异常通知begin--");
        ex.printStackTrace();
        System.out.println("--异常通知end--");
    }

    //最终通知
    public void myafter(){
        System.out.println("--最终通知--");
    }
}

```

6) 修改 aop.xml

```

<bean id="userservice"
      class="tarena.service.UserServiceImpl"></bean>
<bean id="aopbean" class="tarena.aop.AopBean"></bean>

<aop:config>
    <aop:pointcut id="servicepointcut"
        expression="within (tarena.service.*)" />

    <aop:aspect id="aspectbean" ref="aopbean">
        <aop:before method="mybefore"
            pointcut-ref="servicepointcut"/>

        <aop:after-returning method="myafterReturning"
            returning="retVal"
            pointcut-ref="servicepointcut"/>

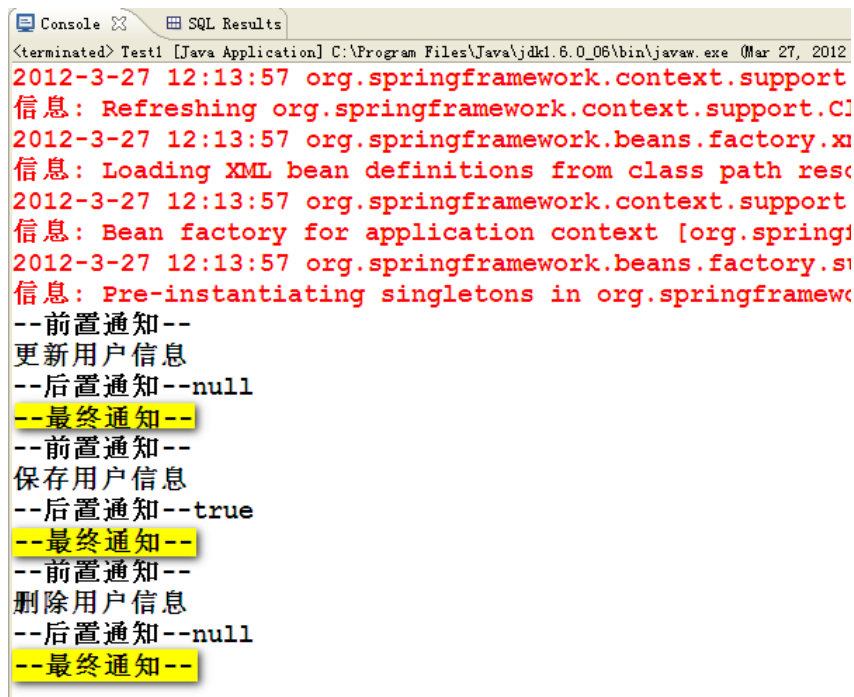
        <aop:after-throwing method="myafterException"
            throwing="ex"
            pointcut-ref="servicepointcut"/>

        <aop:after method="myafter"
            pointcut-ref="servicepointcut"/>
    </aop:aspect>
</aop:config>

```

7) 运行 Test1

注意：最终通知不论是否发生异常都会执行



```
<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2012)
2012-3-27 12:13:57 org.springframework.context.support
信息: Refreshing org.springframework.context.support.Cl
2012-3-27 12:13:57 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-27 12:13:57 org.springframework.context.support
信息: Bean factory for application context [org.spring
2012-3-27 12:13:57 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframew
--前置通知--
更新用户信息
--后置通知--null
--最终通知--
--前置通知--
保存用户信息
--后置通知--true
--最终通知--
--前置通知--
删除用户信息
--后置通知--null
--最终通知--
```

通知的基本使用就是如此，

环绕通知[<aop:around />](#)在 OptLogger 中已经讲过了。

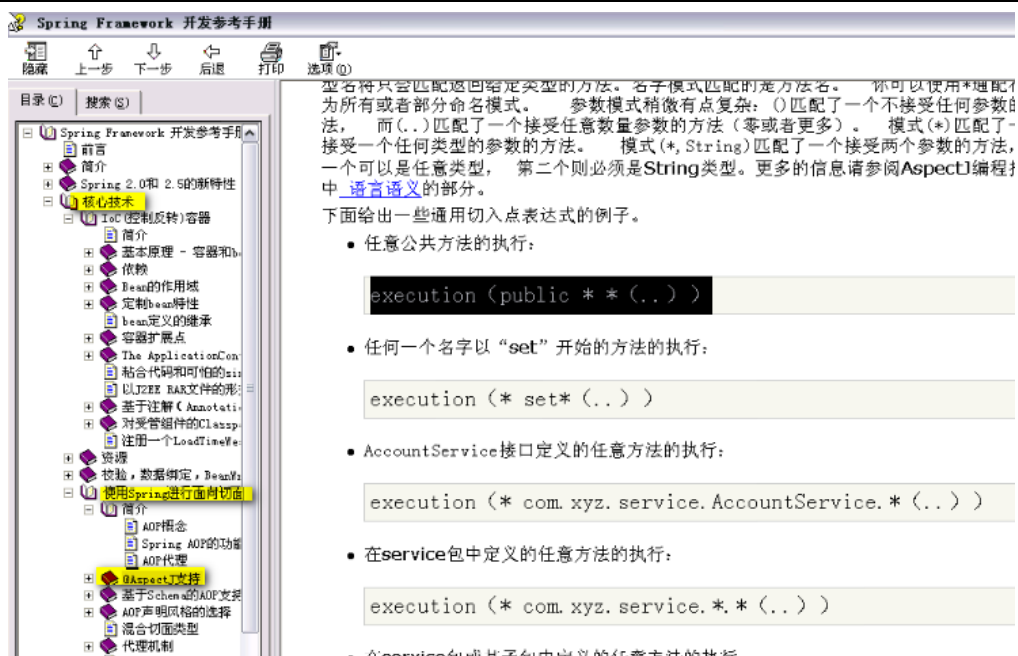
(案例结束)

4. 切入点表达式 *

请下载 [Spring2.5 Reference 中文版.zip](#) (注：.chm 文件，请在 windows 系统下查看)

参考 [Spring FrameWork 开发参考手册](#)

核心技术 -- 使用 Spring 进行面向切面编程 -- @AspectJ 支持



打开 [6.2.3.4. 实例](#)，可以查看到切入点表达式实例，常用的都已经在文档中提供，请参考学习。

6.2.3.4. 示例

Spring AOP 用户可能会经常使用 `execution` 切入点指示符。执行表达式的格式如下：

```
execution (modifiers-pattern? ret-type-pattern declaring-type-pattern? throws-pattern?)
```

除了返回类型模式（上面代码片段中的 `ret-type-pattern`），名字模式和参数模式以外，型必须依次匹配一个连接点。你会使用的最频繁返回类型模式是*，它代表了匹配任意的方法。名字模式匹配的是方法名。你可以使用*通配符作为所有或者部分命名模式。法，而(..)匹配了一个接受任意数量参数的方法（零或者更多）。模式(*)匹配了一个接受两个参数的方法，第一个可以是任意类型，第二个则必须是String类型。更多的信息，下面给出一些通用切入点表达式的例子。

- 任意公共方法的执行：

```
execution (public * * (..))
```

- 任何一个名字以“set”开始的方法的执行：

```
execution (* set* (..))
```

- AccountService接口定义的任意方法的执行：

切入点表达式用于指定目标对象及其作用位置。

1) execution 方法限定

execution(modifiers-pattern?
ret-type-pattern
declaring-type-pattern?
name-pattern(param-pattern)
throws-pattern?)

execution(public * *(..))

表示无要求，只要是 public 修饰的方法就行

execution(* set*(..))

只要以 set 打头的方法，就可以。

execution(* com.xyz.service.AccountService.*(..))

匹配是介个类 com.xyz.service.AccountService.*(..)下的所有方法

execution(* com.xyz.service.*.*(..))

匹配 com.xyz.service 介个包下的所有方法

execution(* com.xyz.service..*.*(..))

注意这个和上面的很像，就多几个点，表示 service 及其子包下所有方法
(上面的不包括子包)

2) within 类型限定

within(com.xyz.service.AccountService)

限定 com.xyz.service.AccountService 类中所有方法

和 execution(* com.xyz.service.AccountService.*(..))效果相同

within(com.xyz.service.*)

限定 com.xyz.service 包下的所有方法 (不包含子包)

within(com.xyz.service..*)

限定 com.xyz.service 包下的所有方法 (包含子包)

3) this/target 特定类型限定

实现 AccountService 接口的代理对象的任意连接点

this(com.xyz.service.AccountService)

实现 AccountService 接口的目标对象的任意连接点

target(com.xyz.service.AccountService)

注意 this 和 target 的区别

4) args 方法参数类型限定

任何一个只接受一个参数，并且运行时所传入的参数是 Serializable 接口

args(java.io.Serializable)

5) bean 对 Bean 对象名称限定

匹配 bean 对象名称以 service 结尾的对象 bean(*service)

【课堂练习 1】记录异常日志 **

1) 使用【案例 1】项目 spring2

2) 导入 log4j 的 jar 包

请下载 log4j-1.2.11.zip

3) 新建 log4j.properteis

```
log4j.rootLogger=WARN,stdout,file
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.SimpleLayout
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=D:\\error.log
log4j.appender.file.layout=org.apache.log4j.SimpleLayout
```

4) 新建 ExceptionLogger

```
package tarena.aop;

import org.apache.log4j.Logger;

public class ExceptionLogger {
    Logger logger = Logger.getLogger(ExceptionLogger.class);
    public void loggerExcetpion(Exception ex){
        //将ex异常信息写入文件中
        logger.error(ex);
    }
}
```

5) 修改 aop.xml

```
<bean id="userservice" class="tarena.service.UserServiceImpl"></bean>
    <bean id="exceptionlogger" class="tarena.aop.ExceptionLogger"></bean>

    <aop:config>
        <aop:pointcut id="servicepointcut"
            expression="execution(* tarena.service.*(..))" />
```

```
<aop:aspect id="loggeraspect" ref="exceptionlogger">
    <aop:after-throwing throwing="ex" method="loggerExcetpion"
        pointcut-ref="servicepointcut"/>
</aop:aspect>
</aop:config>
```

6) UserServiceImpl

```
package tarena.service;

public class UserServiceImpl implements UserService {

    public void delete() {
        System.out.println("删除用户信息");

        //模拟NullPointerException
        String s = null;
        s.length();
    }

    public boolean save() {
        System.out.println("保存用户信息");
        return true;
    }

    public void update() {
        System.out.println("更新用户信息");
    }
}
```

7) Test

```
package tarena.service;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test1 {
    private static final String CONFIG = "aop.xml";
}
```

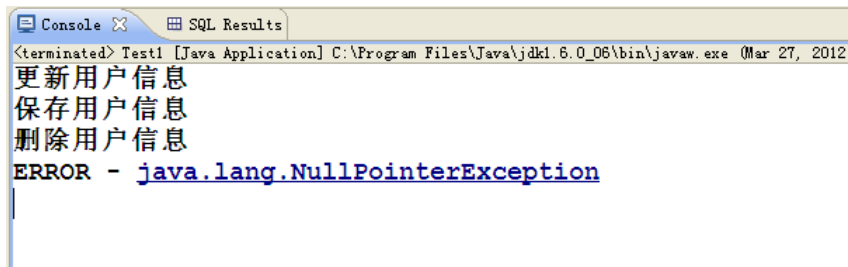
```

/**
 * @param args
 */
public static void main(String[] args) {
    try {
        ApplicationContext ac =
            new ClassPathXmlApplicationContext(CONFIG);
        UserService userService =
            (UserService)ac.getBean("userservice");
        userService.update();
        userService.save();
        userService.delete();
    } catch (Exception e) {

    }
}
}

```

8) 运行 Test



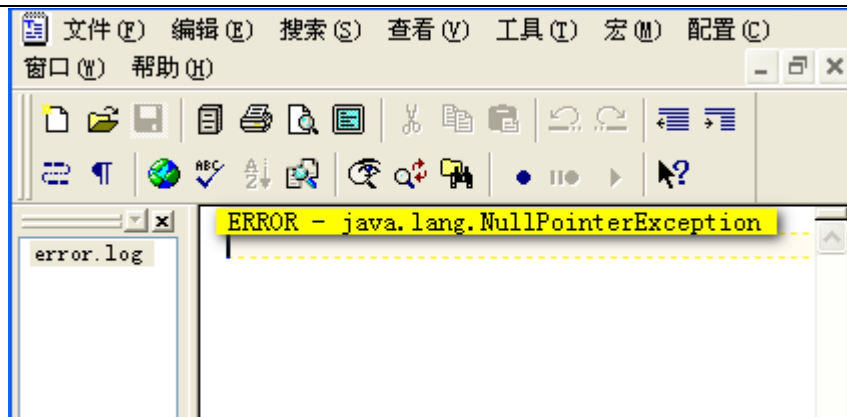
```

<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2012)
更新用户信息
保存用户信息
删除用户信息
ERROR - java.lang.NullPointerException

```

9) 查看 d:/error.log

记录了一条错误日志



我们可以对异常信息进行拼接，使用 StringBuffer，这样显示更清晰

10) 修改 ExceptionLogger

```
package tarena.aop;

import org.apache.log4j.Logger;

public class ExceptionLogger {
    Logger logger = Logger.getLogger(ExceptionLogger.class);

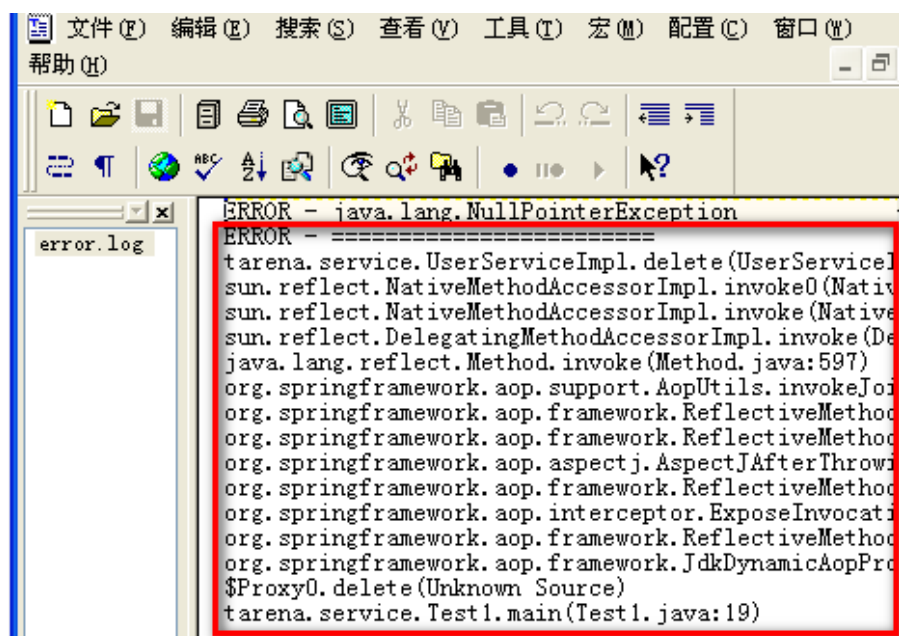
    public void loggerExcetpion(Exception ex){
        // 将ex异常信息写入文件中
        StringBuffer sb = new StringBuffer();
        sb.append("=====\n");
        StackTraceElement[] element = ex.getStackTrace();

        for(StackTraceElement e:element){
            sb.append(e+ "\n");
        }
        logger.error(sb.toString());
    }
}
```

11) 运行 Test

```
Console SQL Results
<terminated> Test1 [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 27, 2012 4:11:11 PM)
更新用户信息
保存用户信息
删除用户信息
ERROR - =====
tarena.service.UserServiceImpl.delete (UserServiceImpl.java:19)
sun.reflect.NativeMethodAccessorImpl.invoke0 (NativeMethodAccessorImpl.java:57)
sun.reflect.NativeMethodAccessorImpl.invoke (NativeMethodAccessorImpl.java:62)
sun.reflect.DelegatingMethodAccessorImpl.invoke (DelegatingMethodAccessorImpl.java:29)
java.lang.reflect.Method.invoke (Method.java:597)
org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection (AopUtils.java:309)
org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint (ReflectiveMethodInvocation.java:182)
org.springframework.aop.framework.ReflectiveMethodInvocation.invoke (ReflectiveMethodInvocation.java:151)
org.springframework.aop.aspectj.AspectJAfterThrowingAdvice.invoke (AspectJAfterThrowingAdvice.java:45)
org.springframework.aop.framework.ReflectiveMethodInvocation.invoke (ReflectiveMethodInvocation.java:151)
org.springframework.aop.interceptor.ExposeInvocationInterceptor.invoke (ExposeInvocationInterceptor.java:91)
org.springframework.aop.framework.ReflectiveMethodInvocation.invoke (ReflectiveMethodInvocation.java:151)
org.springframework.aop.framework.JdkDynamicAopProxy.invoke (JdkDynamicAopProxy.java:200)
$Proxy0.delete (Unknown Source)
tarena.service.Test1.main (Test1.java:19)
```

12) 查看 d:/error.log



```
文件(F) 编辑(E) 搜索(S) 查看(V) 工具(T) 宏(M) 配置(C) 窗口(W) 帮助(H)
ERROR - java.lang.NullPointerException
ERROR - =====
tarena.service.UserServiceImpl.delete (UserServiceImpl.java:19)
sun.reflect.NativeMethodAccessorImpl.invoke0 (NativeMethodAccessorImpl.java:57)
sun.reflect.NativeMethodAccessorImpl.invoke (NativeMethodAccessorImpl.java:62)
sun.reflect.DelegatingMethodAccessorImpl.invoke (DelegatingMethodAccessorImpl.java:29)
java.lang.reflect.Method.invoke (Method.java:597)
org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection (AopUtils.java:309)
org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint (ReflectiveMethodInvocation.java:182)
org.springframework.aop.framework.ReflectiveMethodInvocation.invoke (ReflectiveMethodInvocation.java:151)
org.springframework.aop.aspectj.AspectJAfterThrowingAdvice.invoke (AspectJAfterThrowingAdvice.java:45)
org.springframework.aop.framework.ReflectiveMethodInvocation.invoke (ReflectiveMethodInvocation.java:151)
org.springframework.aop.interceptor.ExposeInvocationInterceptor.invoke (ExposeInvocationInterceptor.java:91)
org.springframework.aop.framework.ReflectiveMethodInvocation.invoke (ReflectiveMethodInvocation.java:151)
org.springframework.aop.framework.JdkDynamicAopProxy.invoke (JdkDynamicAopProxy.java:200)
$Proxy0.delete (Unknown Source)
tarena.service.Test1.main (Test1.java:19)
```

在我们写程序的时候,如果使用框架技术(比如 struts2 和 hibernate),这些框架技术也使用 log4j 记录日志。

如果我们这样写,框架底层的一些错误也会被记录下来。

13) 修改 log4j.properties

```
log4j.rootLogger=WARN,stdout,file
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.SimpleLayout
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=D:\\error.log
log4j.appender.file.layout=org.apache.log4j.SimpleLayout
```

但是，我们关心的更多的是我们自定义的方法发生异常，所以我们一般这样写

14) 修改 log4j.properties

```
#default use Logger
log4j.rootLogger=INFO,stdout

#tarena package use Logger
log4j.logger.tarena=ERROR,file
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.SimpleLayout
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=D:\\error.log
log4j.appender.file.layout=org.apache.log4j.SimpleLayout
```

如果写 OFF，表示关闭其他的日志信息，只有 tarena 下的错误日志才记录。

```
1 log4j.rootLogger=OFF,stdout,file
2 log4j.logger.tarena=ERROR
3 log4j.appender.stdout=org.apache.log4j.Cons
4 log4j.appender.stdout.layout=org.apache.log
5 log4j.appender.file=org.apache.log4j.FileApp
6 log4j.appender.file.File=D:\\error.log
7 log4j.appender.file.layout=org.apache.log4j.:
```

(案例结束)