[Spring Aop详尽教程](http://blog.csdn.net/wangpeng047/article/details/8556800)

标签： [Spring](http://www.csdn.net/tag/Spring)[aop](http://www.csdn.net/tag/aop)[教程](http://www.csdn.net/tag/%e6%95%99%e7%a8%8b)

2013-01-30 19:07 16437人阅读 [评论](http://blog.csdn.net/wangpeng047/article/details/8556800#comments)(7) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/wangpeng047/article/details/8556800#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

学习总结（45） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg java（91） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg spring（16） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录

1. [一概念](http://blog.csdn.net/wangpeng047/article/details/8556800#t0)
2. [二用途](http://blog.csdn.net/wangpeng047/article/details/8556800#t1)
3. [三详解](http://blog.csdn.net/wangpeng047/article/details/8556800#t2)
   1. [切面Aspect](http://blog.csdn.net/wangpeng047/article/details/8556800#t3)
   2. [连接点Joinpoint](http://blog.csdn.net/wangpeng047/article/details/8556800#t4)
   3. [通知Advice](http://blog.csdn.net/wangpeng047/article/details/8556800#t5)
   4. [切入点Pointcut](http://blog.csdn.net/wangpeng047/article/details/8556800#t6)
   5. [目标对象Target Object](http://blog.csdn.net/wangpeng047/article/details/8556800#t7)
   6. [AOP代理AOP Proxy](http://blog.csdn.net/wangpeng047/article/details/8556800#t8)

**一、概念**

AOP（Aspect Oriented Programming）：面向切面编程。

面向切面编程（也叫面向方面编程），是目前软件开发中的一个热点，也是Spring框架中的一个重要内容。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。

**二、用途**

日志记录，性能统计，安全控制，权限管理，事务处理，异常处理，资源池管理。

**三、详解**

注意：代码请见下篇博文

**1. 切面（Aspect）**

官方的抽象定义为“一个关注点的模块化，这个关注点可能会横切多个对象”，在本例中，“切面”就是类TestAspect所关注的具体行为，例如：AServiceImpl.barA()的调用就是切面TestAspect所关注的行为之一。“切面”在ApplicationContext中<aop:aspect>来配置。

**2. 连接点（Joinpoint）**

程序执行过程中的某一行为，例如，AServiceImpl.barA()的调用或者BServiceImpl.barB(String \_msg, int \_type)抛出异常等行为。

**3. 通知（Advice）**

“切面”对于某个“连接点”所产生的动作，例如，TestAspect中对com.spring.service包下所有类的方法进行日志记录的动作就是一个Advice。其中，一个“切面”可以包含多个“Advice”，例如TestAspect。Advice共有如下5种类型：

A 前置通知（Before advice） ：在某连接点（JoinPoint）之前执行的通知，但这个通知不能阻止连接点前的执行。xml中在<aop:aspect>里面使用<aop:before>元素进行声明；例如，TestAspect中的doBefore方法。注解中使用@Before声明；例如，TestAnnotationAspect中的doBefore方法。

B 后通知（After advice） ：当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。xml中在<aop:aspect>里面使用<aop:after>元素进行声明。例如，TestAspect中的doAfter方法，所以AOPTest中调用BServiceImpl.barB抛出异常时，doAfter方法仍然执行。注解中使用@After声明。

C 返回后通知（After return advice） ：在某连接点正常完成后执行的通知，不包括抛出异常的情况。xml中在<aop:aspect>里面使用<after-returning>元素进行声明。注解中使用@AfterReturning声明；

D 环绕通知（Around advice） ：包围一个连接点的通知，类似Web中Servlet规范中的Filter的doFilter方法。可以在方法的调用前后完成自定义的行为，也可以选择不执行。xml中在<aop:aspect>里面使用<aop:around>元素进行声明。例如，TestAspect中的doAround方法。注解中使用@Around声明。

E 抛出异常后通知（After throwing advice） ： 在方法抛出异常退出时执行的通知。xml中在<aop:aspect>里面使用<aop:after-throwing>元素进行声明。例如，TestAspect中的doThrowing方法。注解中使用@AfterThrowing声明。

通知执行顺序：前置通知→环绕通知连接点之前→连接点执行→环绕通知连接点之后→返回通知→后通知

                                                                                                 →(如果发生异常)异常通知→后通知

**4. 切入点（Pointcut）**

匹配连接点的断言，在AOP中通知和一个切入点表达式关联。例如，TestAspect中的所有通知所关注的连接点，都由切入点表达式execution(\* com.spring.service.\*.\*(..))来决定。

● 切入点表达式

**execution**：用于匹配方法执行的连接点；

**within**：用于匹配指定类型内的方法执行；

**this**：用于匹配当前AOP代理对象类型的执行方法；注意是AOP代理对象的类型匹配，这样就可能包括引入接口也类型匹配；注意this中使用的表达式必须是完整类名，不支持通配符；

**target**：用于匹配当前目标对象类型的执行方法；注意是目标对象的类型匹配，这样就不包括引入接口也类型匹配；注意target中使用的表达式必须是完整类名，不支持通配符；

**args**：用于匹配当前执行的方法传入的参数为指定类型的执行方法；参数类型列表中的参数必须是完整类名，通配符不支持；args属于动态切入点，这种切入点开销非常大，非特殊情况最好不要使用；

**@within**：用于匹配所以持有指定注解类型内的方法；注解类型也必须是完整类名；

**@target**：用于匹配当前目标对象类型的执行方法，其中目标对象持有指定的注解；注解类型也必须是完整类名；

**@args**：用于匹配当前执行的方法传入的参数持有指定注解的执行；注解类型也必须是完整类名；

**@annotation**：用于匹配当前执行方法持有指定注解的方法；注解类型也必须是完整类名；

**bean**：Spring AOP扩展的，AspectJ没有对于指示符，用于匹配特定名称的Bean对象的执行方法；

**reference pointcut**：表示引用其他命名切入点，只有注解风格支持，XML风格不支持。

● 匹配语法

**[plain]** [view plain](http://blog.csdn.net/wangpeng047/article/details/8556800) [copy](http://blog.csdn.net/wangpeng047/article/details/8556800)

1. \* 匹配任何数量字符；
3. .. 匹配任何数量字符的重复，如在类型模式中匹配任何数量子包；而在方法参数模式中匹配任何数量参数。
5. + 匹配指定类型的子类型；仅能作为后缀放在类型模式后边。

例如：

java.lang.String   匹配String类型；

java.\*.String   匹配java包下的任何“一级子包”下的String类型；如匹配java.lang.String，但不匹配java.lang.ss.String。

java..\*   匹配java包及任何子包下的任何类型；如匹配java.lang.String、java.lang.annotation.Annotation。

java.lang.\*ing   匹配任何java.lang包下的以ing结尾的类型；

java.lang.Number+   匹配java.lang包下的任何Number的自类型；如匹配java.lang.Integer，也匹配java.math.BigInteger。

● 匹配种类

**A 类**

**[plain]** [view plain](http://blog.csdn.net/wangpeng047/article/details/8556800) [copy](http://blog.csdn.net/wangpeng047/article/details/8556800)

1. 注解 类名

注解：可选，类上持有的注解，如@Deprecated；

类名：必填，任何类的完整名称。

**B 方法**

**[plain]** [view plain](http://blog.csdn.net/wangpeng047/article/details/8556800) [copy](http://blog.csdn.net/wangpeng047/article/details/8556800)

1. 注解 修饰符 返回值类型 类名 方法名(参数列表) 异常列表

注解：可选，方法上持有的注解，如@Deprecated；

修饰符：可选，如public、protected；

返回值类型：必填，可以是任何类型模式；“\*”表示所有类型；

类名：可选，任何类的完整名称；

方法名：必填，可以使用“\*”进行模式匹配；

参数列表：“()”表示方法没有任何参数；“(..)”表示匹配接受任意个参数的方法，“(..,java.lang.String)”表示匹配接受java.lang.String类型的参数结束，且其前边可以接受有任意个参数的方法；“(java.lang.String,..)” 表示匹配接受java.lang.String类型的参数开始，且其后边可以接受任意个参数的方法；“(\*,java.lang.String)” 表示匹配接受java.lang.String类型的参数结束，且其前边接受有一个任意类型参数的方法；

异常列表：可选，以“throws 异常全限定名列表”声明，异常全限定名列表如有多个以“,”分割，如throws java.lang.IllegalArgumentException, java.lang.ArrayIndexOutOfBoundsException。

**C Bean**

可以使用Bean的id或name进行匹配，并且可使用通配符“\*”。

● 匹配逻辑运算

可以使用且（&&）、或（||）、非（！）来组合切入点表达式。由于在XML中使用“&&”需要使用转义字符“&amp;&amp;”来代替之，所以很不方便，因此Spring ASP 提供了and、or、not来代替&&、||、！。

● 切入点表达式示例

**A execution**

|  |  |
| --- | --- |
| 模式 | 描述 |
| public \* \*(..) | 任何公共方法的执行 |
| \* cn.javass..IPointcutService.\*() | cn.javass包及所有子包下IPointcutService接口中的任何无参方法 |
| \* cn.javass..\*.\*(..) | cn.javass包及所有子包下任何类的任何方法 |
| \* cn.javass..IPointcutService.\*(\*) | cn.javass包及所有子包下IPointcutService接口的任何只有一个参数方法 |
| \* (!cn.javass..IPointcutService+).\*(..) | 非“cn.javass包及所有子包下IPointcutService接口及子类型”的任何方法 |
| \* cn.javass..IPointcutService+.\*() | cn.javass包及所有子包下IPointcutService接口及子类型的的任何无参方法 |
| \*  cn.javass..IPointcut\*.test\*(java.util.Date) | cn.javass包及所有子包下IPointcut前缀类型的的以test开头的只有一个参数类型为java.util.Date的方法，注意该匹配是根据方法签名的参数类型进行匹配的，而不是根据执行时传入的参数类型决定的如定义方法：public void test(Object obj);即使执行时传入java.util.Date，也不会匹配的； |
| \* cn.javass..IPointcut\*.test\*(..) throws IllegalArgumentException, ArrayIndexOutOfBoundsException | cn.javass包及所有子包下IPointcut前缀类型的的任何方法，且抛出IllegalArgumentException和ArrayIndexOutOfBoundsException异常 |
| \* (cn.javass..IPointcutService+&& java.io.Serializable+).\*(..) | 任何实现了cn.javass包及所有子包下IPointcutService接口和java.io.Serializable接口的类型的任何方法 |
| @java.lang.Deprecated \* \*(..) | 任何持有@java.lang.Deprecated注解的方法 |
| @java.lang.Deprecated @cn.javass..Secure  \* \*(..) | 任何持有@java.lang.Deprecated和@cn.javass..Secure注解的方法 |
| @(java.lang.Deprecated ||  cn.javass..Secure) \* \*(..) | 任何持有@java.lang.Deprecated或@  cn.javass..Secure注解的方法 |
| (@cn.javass..Secure \*) \*(..) | 任何返回值类型持有@cn.javass..Secure的方法 |
| \* (@cn.javass..Secure \*).\*(..) | 任何定义方法的类型持有@cn.javass..Secure的方法 |
| \* \*(@cn.javass..Secure (\*) ,  @cn.javass..Secure (\*)) | 任何签名带有两个参数的方法，且这个两个参数都被@ Secure标记了，如public void test(@Secure String str1,@Secure String str1); |
| \* \*((@ cn.javass..Secure \*))或\* \*(@ cn.javass..Secure \*) | 任何带有一个参数的方法，且该参数类型持有@ cn.javass..Secure；如public void test(Model model);且Model类上持有@Secure注解 |
| \* \*(@cn.javass..Secure (@cn.javass..Secure \*)  ,@ cn.javass..Secure (@cn.javass..Secure  \*)) | 任何带有两个参数的方法，且这两个参数都被@ cn.javass..Secure标记了；且这两个参数的类型上都持有@  cn.javass..Secure； |
| \* \*(java.util.Map<cn.javass..Model, cn.javass..Model>, ..) | 任何带有一个java.util.Map参数的方法，且该参数类型是以<  cn.javass..Model, cn.javass..Model >为泛型参数；注意只匹配第一个参数为java.util.Map,不包括子类型；如public void test(HashMap<Model, Model> map, String str);将不匹配，必须使用“\* \*(java.util.HashMap<cn.javass..Model,cn.javass..Model>, ..)”进行匹配；而public void test(Map map, int i);也将不匹配，因为泛型参数不匹配 |
| \*  \*(java.util.Collection<@cn.javass..Secure \*>) | 任何带有一个参数（类型为java.util.Collection）的方法，且该参数类型是有一个泛型参数，该泛型参数类型上持有@cn.javass..Secure注解；如public void test(Collection<Model> collection);Model类型上持有@cn.javass..Secure |

**B within**

|  |  |
| --- | --- |
| 模式 | 描述 |
| within(cn.javass..\*) | cn.javass包及子包下的任何方法执行 |
| within(cn.javass..IPointcutService+) | cn.javass包或所有子包下IPointcutService类型及子类型的任何方法 |
| within(@cn.javass..Secure \*) | 持有cn.javass..Secure注解的任何类型的任何方法必须是在目标对象上声明这个注解，在接口上声明的对它不起作用 |

**C this**

|  |  |
| --- | --- |
| 模式 | 描述 |
| this(cn.javass.spring.chapter6.service.IPointcutService) | 当前AOP对象实现了 IPointcutService接口的任何方法 |
| this(cn.javass.spring.chapter6.service.IIntroductionService) | 当前AOP对象实现了 IIntroductionService接口的任何方法也可能是引入接口 |

**D target**

|  |  |
| --- | --- |
| 模式 | 描述 |
| target(cn.javass.spring.chapter6.service.IPointcutService) | 当前目标对象（非AOP对象）实现了 IPointcutService接口的任何方法 |
| target(cn.javass.spring.chapter6.service.IIntroductionService) | 当前目标对象（非AOP对象） 实现了IIntroductionService 接口的任何方法不可能是引入接口 |

|  |  |
| --- | --- |
| 模式 | 描述 |
| args (java.io.Serializable,..) | 任何一个以接受“传入参数类型为 java.io.Serializable” 开头，且其后可跟任意个任意类型的参数的方法执行，args指定的参数类型是在运行时动态匹配的 |

**F @within**

|  |  |
| --- | --- |
| 模式 | 描述 |
| @within  cn.javass.spring.chapter6.Secure) | 任何目标对象对应的类型持有Secure注解的类方法；必须是在目标对象上声明这个注解，在接口上声明的对它不起作用 |

**G @target**

|  |  |
| --- | --- |
| 模式 | 描述 |
| @target (cn.javass.spring.chapter6.Secure) | 任何目标对象持有Secure注解的类方法；必须是在目标对象上声明这个注解，在接口上声明的对它不起作用 |

**H @args**

|  |  |
| --- | --- |
| 模式 | 描述 |
| @args (cn.javass.spring.chapter6.Secure) | 任何一个只接受一个参数的方法，且方法运行时传入的参数持有注解 cn.javass.spring.chapter6.Secure；动态切入点，类似于arg指示符； |

**I @annotation**

|  |  |
| --- | --- |
| 模式 | 描述 |
| @annotation(cn.javass.spring.chapter6.Secure  ) | 当前执行方法上持有注解  cn.javass.spring.chapter6.Secure将被匹配 |

**J bean**

|  |  |
| --- | --- |
| 模式 | 描述 |
| bean(\*Service) | 匹配所有以Service命名（id或name）结尾的Bean |

**K reference pointcut**

**[java]** [view plain](http://blog.csdn.net/wangpeng047/article/details/8556800) [copy](http://blog.csdn.net/wangpeng047/article/details/8556800)

1. @Pointcut(value="bean(\*Service)")
2. **private** **void** pointcut1(){}
4. @Pointcut(value="@args(cn.javass.spring.chapter6.Secure)")
5. **private** **void** pointcut2(){}
7. @Pointcut(value="pointcut1()&&pointcut2()")
8. **private** **void** pointcut3(){}

● 通知方法参数注入

在Spring AOP中，除了execution和bean指示符不能传递参数给通知方法，其他指示符都可以将匹配的相应参数或对象自动传递给通知方法。例如：

**[java]** [view plain](http://blog.csdn.net/wangpeng047/article/details/8556800) [copy](http://blog.csdn.net/wangpeng047/article/details/8556800)

1. @Before(value="execution(\* test(\*)) && args(param)", argNames="param")
2. **public** **void** before1(String param) {
3. System.out.println("===param:" + param);
4. }

首先execution(\* test(\*))匹配任何方法名为test，且有一个任何类型的参数；

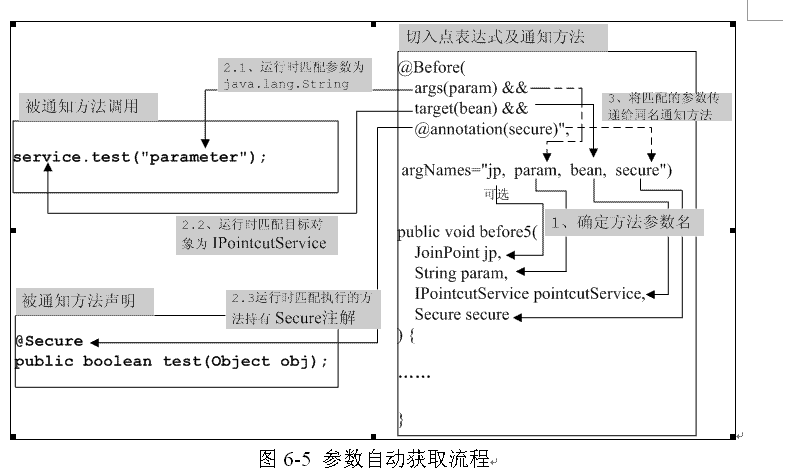
args(param)将首先查找通知方法上同名的参数，并在方法执行时（运行时）匹配传入的参数是使用该同名参数类型，即java.lang.String；如果匹配将把该被通知参数传递给通知方法上同名参数。

其他指示符（除了execution和bean指示符）都可以使用这种方式进行参数绑定。

● 综合示例

**[java]** [view plain](http://blog.csdn.net/wangpeng047/article/details/8556800) [copy](http://blog.csdn.net/wangpeng047/article/details/8556800)

1. @Before(value="args(param) && target(bean) && @annotation(secure)", argNames="jp,param,bean,secure")
2. **public** **void** before5(JoinPoint jp, String param,
3. IPointcutService pointcutService, Secure secure) {
4. ……
5. }



除了上边介绍的普通方式，也可以对使用命名切入点自动获取参数：

**[java]** [view plain](http://blog.csdn.net/wangpeng047/article/details/8556800) [copy](http://blog.csdn.net/wangpeng047/article/details/8556800)

1. @Pointcut(value="args(param)", argNames="param")
2. **private** **void** pointcut1(String param){}
4. @Pointcut(value="@annotation(secure)", argNames="secure")
5. **private** **void** pointcut2(Secure secure){}
7. @Before(value = "pointcut1(param) && pointcut2(secure)", argNames="param, secure")
8. **public** **void** before6(JoinPoint jp, String param, Secure secure) {
9. ……
10. }

**5. 目标对象（Target Object）**

被一个或者多个切面所通知的对象。例如，AServcieImpl和BServiceImpl，当然在实际运行时，Spring AOP采用代理实现，实际AOP操作的是TargetObject的代理对象。

**6. AOP代理（AOP Proxy）**

在Spring AOP中有两种代理方式，JDK动态代理和CGLIB代理。默认情况下，TargetObject实现了接口时，则采用JDK动态代理，例如：AServiceImpl；反之，采用CGLIB代理，例如：BServiceImpl。强制使用CGLIB代理需要将 <aop:config>的 proxy-target-class属性设为true。