基于注解的Spring AOP开发

# @Aspect介绍

**定义一个类为切面类时，需要利用@Aspect修饰该类。**

**只有这样该切面类中的增强处理（通知）才会有效。**

**在Spring中启用AspectJ注解支持**

**要在Spring应用中使用AspectJ注解，必须在classpath下包含AspectJ类库：aopalliance.jar、aspectj.weaver.jar和spring-aspects.jar。**

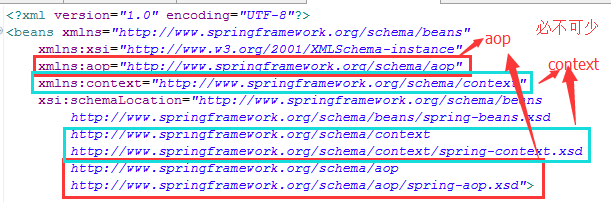
**将aop Schema添加到<beans>根元素中。**

**要在Spring IOC容器中启用AspectJ注解支持，只要早bean配置文件中定义一个空的XML元素<aop:aspectj-autoproxy>：当Spring IOC容器侦测到bean配置文件中的<aop:aspectj-autoproxy>元素时，会自动为与AspectJ切面匹配的bean创建代理。**

必须在spring配置完文件中声明：

包括三部分：

## 引入aop空间和context空间



## 自动搜索切面类

下面的功能：自动搜索切面类。

<context:component-scan base-package="*org.example,org.myaspect*">

<context:include-filter type="annotation"

expression="**org.aspectj.lang.annotation.Aspect**"/>

</context:component-scan>

如果不设置自动搜索切面类，把切面类当成是一个普通的bean加载也可以。

<bean id="applause" class="cn.bupt.aspect\_spring.Applause"></bean>

## 启动@AspectJ支持

使AspectJ注解起作用：自动为匹配的类生成代理对象

<aop:aspectj-autoproxy/>

## 简单的示例：

<?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:aop=*"http://www.springframework.org/schema/aop"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd*

*http://www.springframework.org/schema/aop*

[*http://www.springframework.org/schema/aop/spring-aop.xsd*](http://www.springframework.org/schema/aop/spring-aop.xsd)*"*>

<!-- 配置自动扫描的包 -->

<context:component-scan base-package=*"cn.bupt.aspect\_spring,com.bupt.advice"*>

<context:include-filter type=*"annotation"*

expression=*"org.aspectj.lang.annotation.Aspect"* />

</context:component-scan>

**<!-- 使AspectJ注解起作用：自动为匹配的类生成代理对象 -->**

**<aop:aspectj-autoproxy />**

<bean id=*"singer"* class=*"cn.bupt.aspect\_spring.Singer"*></bean>

</beans>

# @Order介绍

## 使用方法

指定***切面类的优先级***，只需要指定一个int类型的数值即可，如@Value(0) ，**指定的数值越小，优先级越高**。

## 如何限定同样类型增强处理的优先级

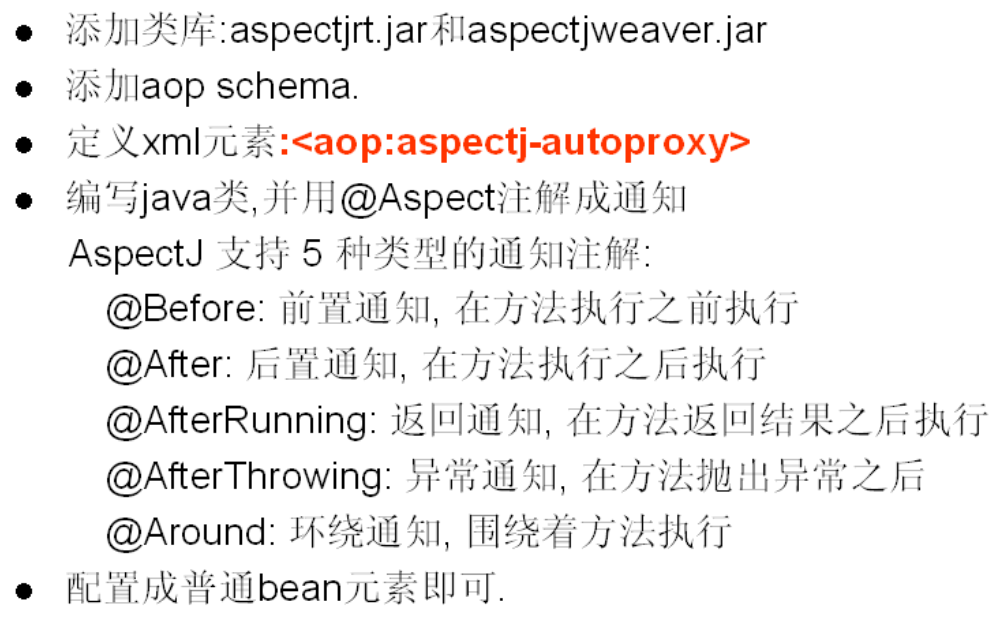
如果不同切面类里的两个相同类型的增强处理在同一个连接点被织入时，则会根据切面类的优先级来进行顺序织入；但是，如果**同一个切面类里**的两个相同类型的增强处理在同一个连接点被织入时，Spring AOP将以**随机的顺序**来织入这两个增强处理，程序没有办法控制它们的织入顺序。

如果确实需要保证它们以固定的顺序被织入，则可考虑将多个增强处理压缩成一个增强处理，或者将不同的增强处理重构到不同的切面类中，通过在切面类级别上进行排序。

## 示例



# 五种通知



## @Before：前置通知。

## @AfterReturning：返回后通知。（只有正常返回了，才会加入通知）

## @After：后置通知。（无论方法正常执行与否，都会加入通知）

## @Around：环绕通知。

## @AfterThrowing：异常通知。

注意：这里AfterReturning与After都是后置通知，应该归为一类，缺少引入通知。

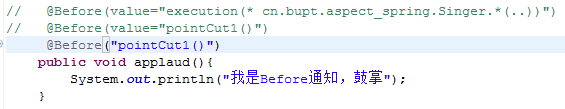
下面对这几个注解通知单独介绍。

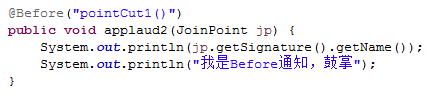
# @Before 前置通知

首先简单介绍定义切入点：**定义一个名叫pointCut1()的切入点**。



定义@Before，只有一个属性，value，指定切入点表达式或调用现有的切入点表达式。value可以省略。



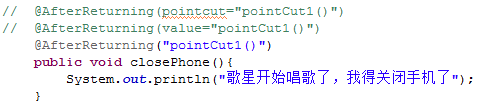


# @AfterReturning 返回后通知

**返回后通知可以获取目标方法的返回值**，当然也可以选择不获取目标方法的返回值。

定义AfterReturning，有两个属性，一个是value/pointcut，一个是returning（可以不指定）。

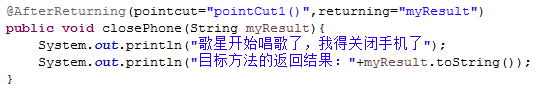
## 不获取目标方法的返回值



## 获取目标方法的返回值

注意：@AfterReturning只可以获取目标方法的返回值，但是不能改变目标方法的返回值。因为当目标方法执行完之后，执行增强处理部分，只是获取了该返回值，改变不了。

这里returning指定的名字与此处方法的参数指定的参数名必须一致。

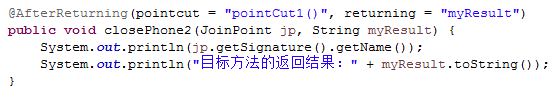


**重点**：

returning有两个作用：

### 用来接收目标方法的返回值；

### 限定切入点：只匹配那些返回值类型与此处方法形参类型一致的方法。 对待增加通知的方法的返回值类型进行了限定，如果这些方法的返回值类型与待接收的类型不一致，则这些方法就不会被增强，即就能成为目标方法。当接收类型定义为Object类型，则就相当于对方法返回值类型没有任何限制。例如：上面的例子中，目标方法的返回类型必须是String才可以，否则不会被增强。



# @After 后置通知

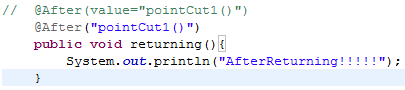
## 介绍

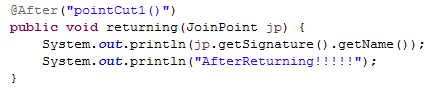


## 注意：

无论目标方法如何结束，或者正常执行完毕，或者抛出异常，@After都会被织入。这种增强处理经常用于释放资源（资源回收）。@After的作用非常类似于异常处理中finally块的作用。

定义@After，**只有一个属性，value**。





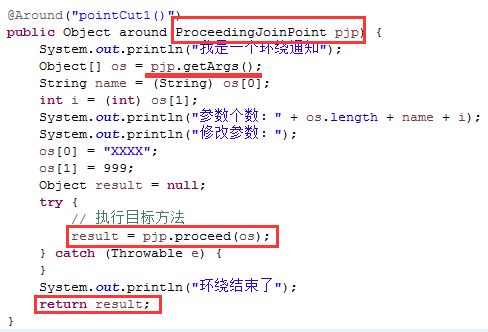
# @Around 环绕通知

## 功能：

**@Around** 环绕通知，可以控制目标方法在什么时候执行，如何执行，甚至可以完全阻止目标方法的执行。

定义**@Around**时，只需要定义一个属性value即可，但是方法的第一个参数必须是**ProceedingJoinPoint**，通过调用其proceed()或proceed(Object[] orgs)来控制目标方法执行的时间、如何执行（改变目标方法的参数）、不调用**proceed**则不会执行**目标方法**。





## 修改目标方法的参数

首先调用ProceedingJoinPoint的getArgs方法获取目标方法的参数数组，然后对其进行查看并修改，再调用proceed(Object[] args)即可修改。当然如果已经很明确目标方法的参数，也可以直接创建一个Object数组传入进去即可。

注意：当传入的Object[] 数组的个数以及元素的具体类型不一致时，都会报出异常。

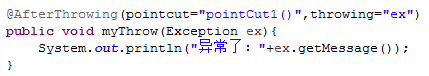
# @AfterThrowing

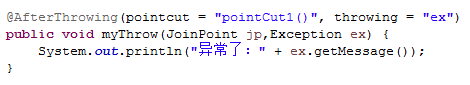
## 主要是用于程序中未处理的异常。

## 两个属性：

**一个是value/pointcut，另一个是throwing。**

与AfterReturning类似，throwing的异常名字，在方法的参数中指定类型，只有当目标方法报出与指定类型一致的异常时才会织入。





# After与AfterReturning的区别：



# 访问目标方法的参数：

将这些增强处理方法的第一个参数定义为**JoinPoint**即可。JoinPoint具有很多getXxx方法，用来获取目标方法的一些信息。如getArgs、getModifiers、getSignature、getThis、getTarget等。对于Around必须使用ProceedingJoinPoint，其实JoinPoint的子类，扩充了proceed方法。只有Around能够修改目标方法的参数，其他的通知都是只可以获取看到，但是无法修改。

