基于XML配置的Spring AOP开发

# 学习目标

如果直接使用AspectJ开发的话，AOP的功能几乎都可以实现。但是在JavaEE中主要是实现**SpringAOP、Spring IoC、AspectJ**的整合，所以主要是在Spring中整合AspectJ。

# Spring AOP整合Aspect开发方法

## 利用注解方式

常用注解：@Aspect 、@Order(int)

@Before:前置通知

@After：后置通知（正常返回或抛出异常后都会执行）

@AfterReturning：返回后通知

@AfterThrowing：异常通知

@Round：环绕通知

## 利用xml配置文件方式

# 在Spring中开发AOP的主要关键点

## 注解开发：

开发AOP时，不需要对目标对象做任何改变，只需要创建**单独的Aspect类**，即用**@Aspect声明的java类**，定义AOP切入点（Pointcut）和增强的方法体即可。

最重要的关键点：**会书写切入点表达式**。

用AspectJ注解声明切面

要在Spring中声明AspectJ切面，只需要在IOC容器中将切面声明为bean实例。当在Spring IOC容器中初始化AspectJ切面之后，Spring IOC容器就会为那些与AspectJ切面相匹配的bean创建代理

在AspectJ注解中，切面只是一个带有@AspectJ注解的Java类

通知是标注有某种注解的简单的Java方法

## xml配置文件

下面详细介绍。

# XML配置方式

在Spring 配置文件中，所有的切面、切入点和增强处理都必须定义在**<aop:config.../>**元素内部。<beans.../>元素下可以包含多个**<aop:config.../>。一个<aop:config.../>中可以包含<aop:pointcut../>、<aop:advisor../>、<aop:aspect../>，这三个元素必须按照这个顺序定义，否则错误。Pointcut—》advisor—》aspect。**

**注意：<aop:config.../>方式与Spring的自动代理方式相冲突，使用<aop:aspectj-autoproxy />显式地启动了自动代理，可能会导致问题。所以建议：要么全部使用<aop:config.../>配置方式，要么全部使用自动代理方式如<aop:aspectj-autoproxy />，不要两者混合使用。**

## <aop:pointcut../>定义切入点。

### 当<aop:pointcut../>直接定义在<aop:config.../>下时，称为全局切入点；可以被多个切面bean共享。

### 当<aop:pointcut../>直接定义在<aop:aspect../>下时，称为局部切入点，只能在该aspect下使用有效。

### 配置切入点：指定两个属性即可。

id：指定切入点的标识名；

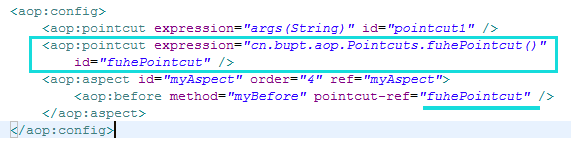
expression：**指定该切入点关联的切入点表达式**。

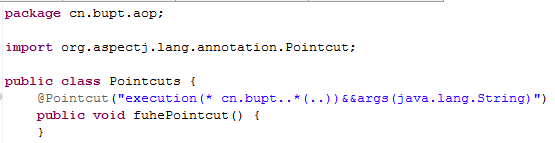
示例：**直接书写切入点表达式**。

**<aop:pointcut expression="args(String)" id="pointcut1" />**

**也可以直接使用某个切面类中的切入点作为切入点表达式。**

**如：**





### Xml配置的切入点表达式与注解配置的切入点表达式支持的切入点指示符一样：

Execution、args、bean、target、this、within。

### 组合切入点表达式

注解方式采用&&、||、!组合切入点表达式，在xml配置文件中，采用and 、or、not进行组合。

## <aop:advisor../>：暂时未使用到。

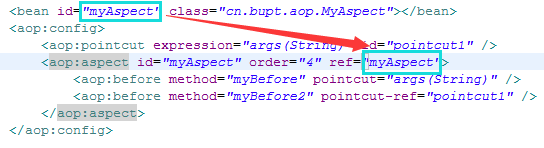
## <aop:aspect../> 配置切面

配置**<aop:aspect../>元素时，可以指定三个属性：**

### id ： 该切面的标识名；

### ref ： 将普通Bean转化为切面Bean。类似@Aspect的作用。

### order：指定切面Bean的优先级。类似@Order。值越小，优先级越高。

示例：将***myAspect***这个bean对象转换成**Aspect切面对象**。

## 配置增强处理

在**<aop:aspect../> 内部配置具体的增强处理：**

### <aop:before../>：配置Before增强处理。

属性：**pointcut或pointcut-ref**只能选择其一，pointcut用来指定一个切入点表达式，pointcut-ref用来调用一个已经存在的切入点表达式。

**method**：指定该切面Bean一个方法名，将此方法作为增强处理。

### <aop:after../>：配置After增强处理。

属性和before一样。

### <aop:after-returning../>：配置After-Returning增强处理。

属性除了具有after具有的之外，还有一个returning。

**returning属性**只有<aop:after-returning../>具有。用来指定一个形参名，AfterReturning增强处理方法通过该形参访问目标方法的返回值。指定的方法中指定了参数类型，所以仍能对连接点的返回值的类型进行限制。

### <aop:around../>：配置Around增强处理。

和before一样。

### <aop:after-throwing../>：配置After-Throwing增强处理。

属性除了具有before具有的之外，还有一个throwing。

Throwing属性：只对<aop:after-throwing../>有效，用来指定一个形参名，AfterThrowing增强处理可以通过该形参访问目标方法所抛出的异常。

这五个增强处理具有的属性。

# 注解配置方式与XML配置方式的优缺点

## XML配置方式

### 优点：

如果用户没有使用JDK1.5以上版本，则只能使用XML配置方式来管理切面、切入点和增强处理等。

使用**xml配置方式**对早期的Spring用户更加习惯，这种方式允许使用纯粹的POJO来支持AOP。当使用AOP作为工具来配置企业服务时，XML会是一个很好的选择。

### 缺点：

不能将切面、切入点、增强处理等封装到一个地方。需要查看切面、切入点、增强处理时，必须同时结合Java文件和XML配置文件来查看；但是使用@Aspect时，则只需要一个单独的类文件即可看到切面、切入点、增强处理等的全部信息。

XML配置方式比@Aspect注解方式有更多的限制：仅仅支持“singleton”切面Bean，不能在XML中组合多个命名连接点的声明。

## 注解配置方式

### 优点：

能够被Spring AOP和AspectJ同时支持。如果某一天将应用改为AspectJ来实现AOP，使用@Aspect 很容易迁移到基于AspectJ的AOP实现中。

### 缺点：

# 示例：

配置文件：

<?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"*>

<aop:config>

<aop:pointcut expression=*"cn.bupt.aop.Pointcuts.fuhePointcut()"*

id=*"fuhePointcut"* />

<aop:aspect id=*"myAspectC"* order=*"4"* ref=*"myAspect"*>

<aop:before method=*"myBefore"* pointcut-ref=*"fuhePointcut"* />

<aop:after method=*"closePhone"* pointcut-ref=*"fuhePointcut"* />

<aop:after-returning method=*"returning"*

pointcut-ref=*"fuhePointcut"* returning=*"result"* />

<aop:after-throwing method=*"myThrow"* pointcut-ref=*"fuhePointcut"*

throwing=*"ex"* />

<aop:around method=*"around"* pointcut-ref=*"fuhePointcut"* />

</aop:aspect>

</aop:config>

<bean id=*"singer"* class=*"cn.bupt.aop.Singer"* />

<bean id=*"myAspect"* class=*"cn.bupt.aop.MyAspect"*></bean>

</beans>

MyAspect.java：

**package** cn.bupt.aop;

**import** org.aspectj.lang.JoinPoint;

**import** org.aspectj.lang.ProceedingJoinPoint;

**public** **class** MyAspect {

**public** **void** myBefore() {

System.***out***.println("Before通知");

}

**public** **void** myThrow(JoinPoint jp, Exception ex) {

System.***out***.println("Throwing" + ex.getMessage());

}

**public** Object around(ProceedingJoinPoint pjp) {

System.***out***.println("环绕通知开始了֪");

Object[] os = pjp.getArgs();

String name = (String) os[0];

System.***out***.println("原来的参数" + os.length + name);

os[0] = "XXXX";

Object result = **null**;

**try** {

result = pjp.proceed();

} **catch** (Throwable e1) {

e1.printStackTrace();

}

**try** {

// 执行调用方法

result = pjp.proceed(os);

} **catch** (Throwable e) {

}

System.***out***.println("返回");

**return** result;

}

**public** **void** closePhone(JoinPoint jp) {

System.***out***.println(jp.getSignature().getName());

System.***out***.println("After 增强处理");

}

**public** **void** returning(JoinPoint jp,Object result) {

System.***out***.println(jp.getSignature().getName());

System.***out***.println("AfterReturning!!!!!"+" ... "+result);

}

}

