基于API的Spring AOP典型开发方式

# AOP的JDK动态代理图解：

# Advice通知（接口）

体系： 接口Advice –> **BeforeAdvice—>MethodBeforeAdvice**

接口Advice –> **AfterAdvice—>AfterReturningAdvice**

**借口Advice—>Interceptor🡪MethodeInterceptor**

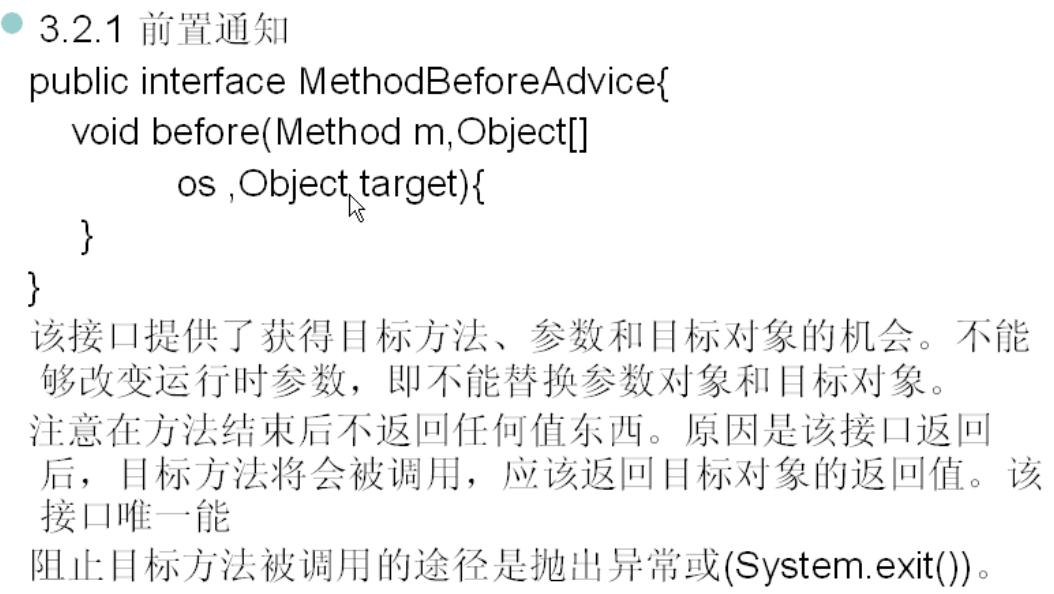
接口Advice –> **AfterAdvice—>ThrowsAdvice**

# 前置通知BeforeAdvice（MethodeBeforeAdvice方法前通知）

MethodBeforeAdvice 是一个接口，存在于org.springframework.aop包中。

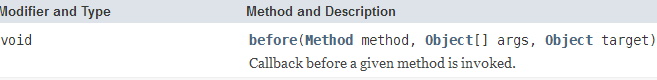


Advice invoked before a method is invoked. Such advices cannot prevent the method call proceeding, unless they throw a Throwable.、



## 方法前通知没有返回值；

## 方法前通知无法阻止被通知方法的执行，除非出现异常或System.exit()。

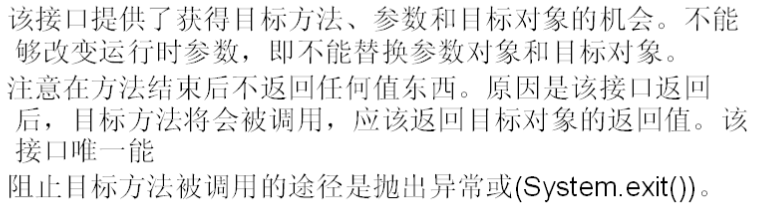


MethodBeforeAdvice 接口中只有一个方法before，参数为

### Method method：被通知的方法。

### Object[] args：

### Object target : 通知的目标。



# AfterAdvice后置通知（AfterReturningAdvice返回后通知）

AfterReturningAdvice 是一个接口，存在org.springframework.aop包中。



After returning advice is invoked only on normal method return, not if an exception is thrown. Such advice can see the return value, but cannot change it.、

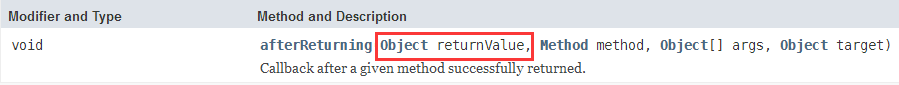
只有一个方法afterReturning。参数有4个：

### Object returnValue:返回值对象。

### Method method：方法。

### Object[] args:参数。

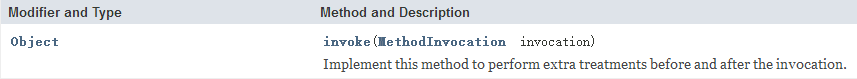
### Object target：目标。



# 环绕通知Interceptor：MethodeInterceptor方法环绕通知（方法拦截器）

**MethodeInterceptor存在包org.aopalliance.intercept中：**

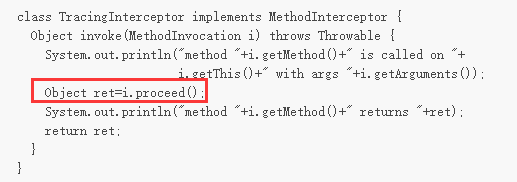




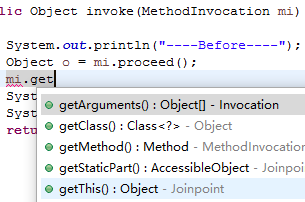
Intercepts calls on an interface on its way to the target. These are nested "on top" of the target.

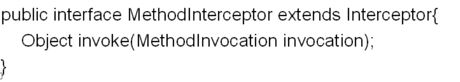
The user should implement the [invoke(MethodInvocation)](http://docs.spring.io/spring/docs/current/javadoc-api/org/aopalliance/intercept/MethodInterceptor.html#invoke-org.aopalliance.intercept.MethodInvocation-) method to modify the original behavior. E.g. the following class implements a tracing interceptor (traces all the calls on the intercepted method(s)):

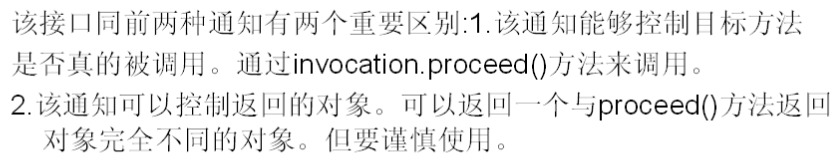
使用示例：



MethodInvocation :可以获取很多东西。



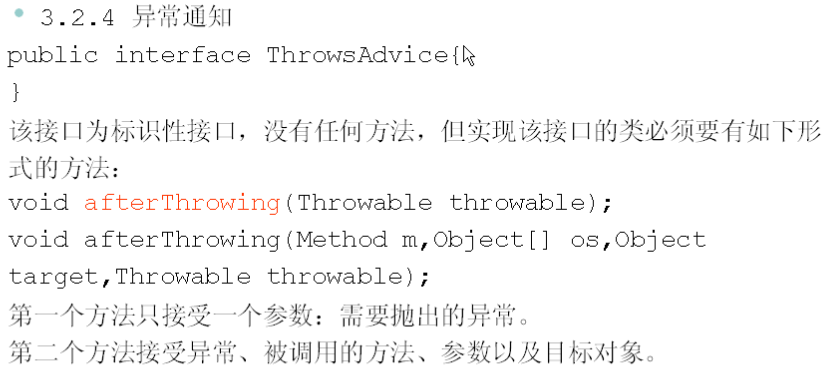




# 异常通知：ThrowsAdvice

ThrowsAdvice：标识性接口。当时实现接口时，必须要实现如下形式的方法。

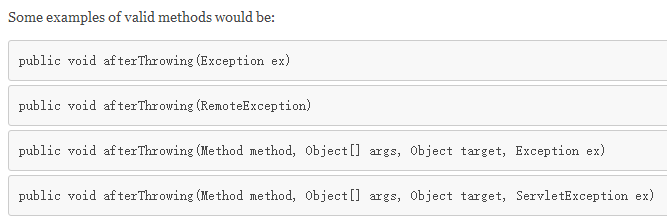
接口Advice –> AfterAdvice—>ThrowsAdvice



Tag interface for throws advice.

There are not any methods on this interface, as methods are invoked by reflection. Implementing classes must implement methods of the form:

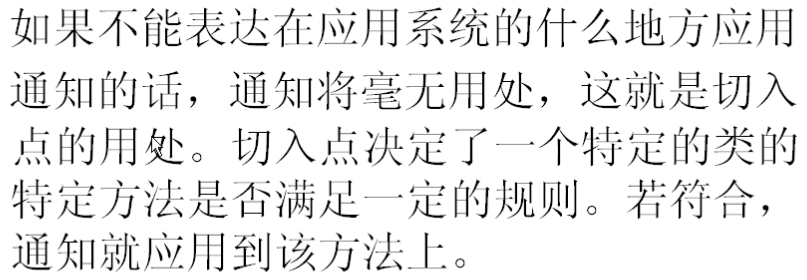


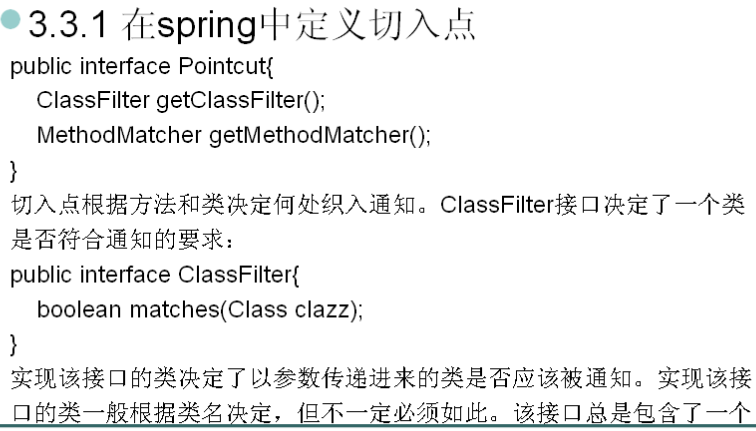


The first three arguments are optional, and only useful if we want further information about the joinpoint, as in AspectJ **after-throwing** advice.

# Advice通知的顺序：

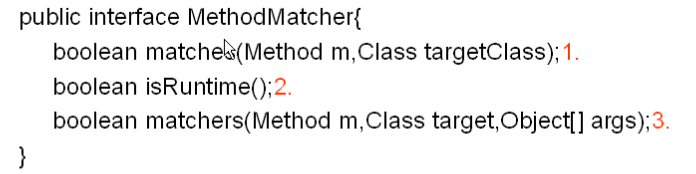
# 在Spring中定义切入点

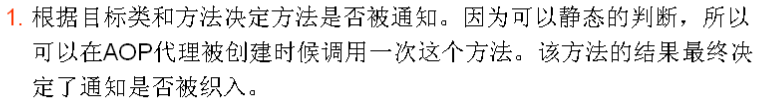


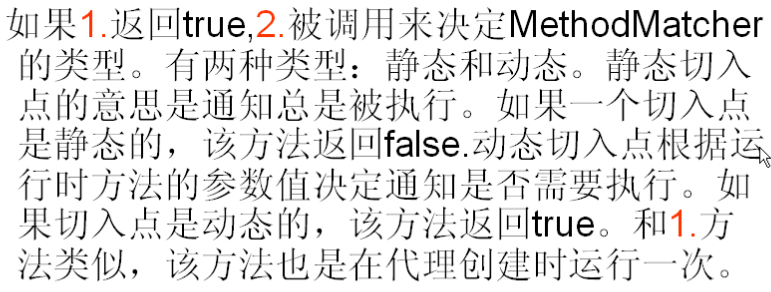


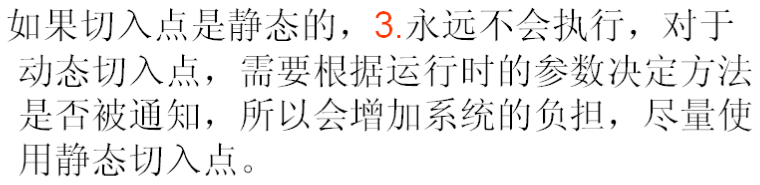


方法匹配：

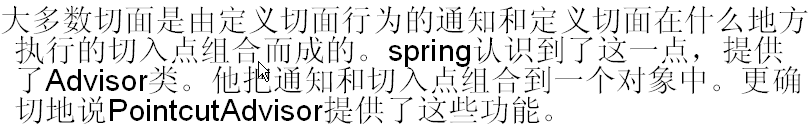


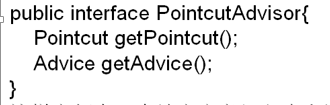






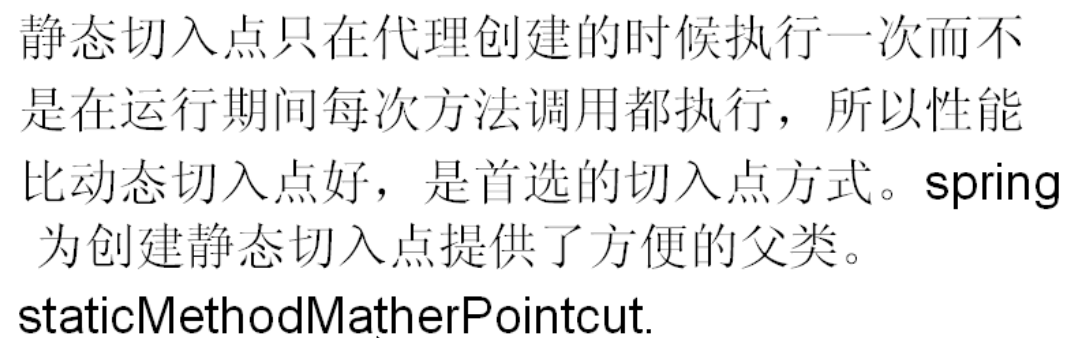
# Advisor：理解



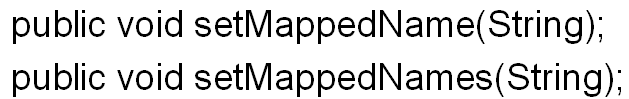


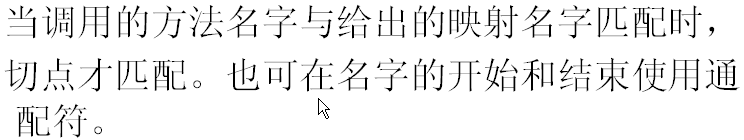


使用Spring的静态切入点：





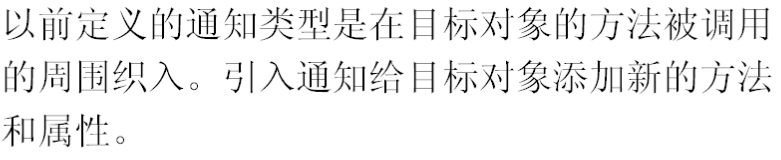




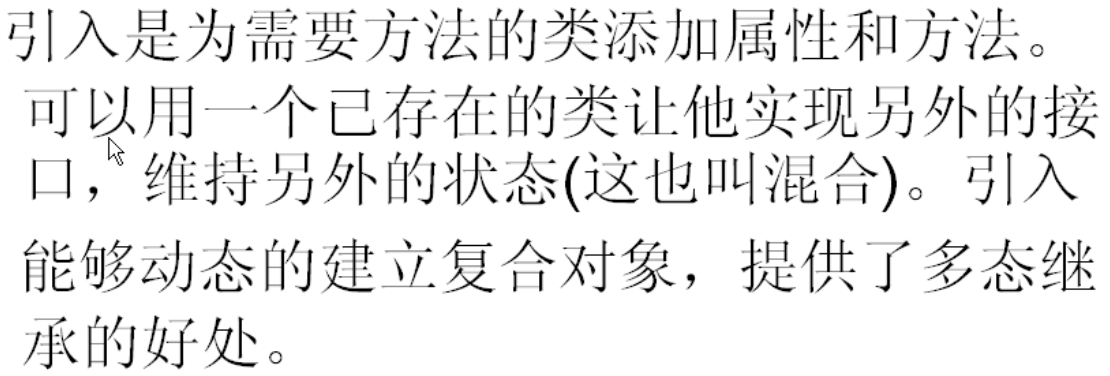
一次只能给一个**Advice**进行封装。通过列出方法的名字，进行限制对哪些方法进行添加**通知**。相当于加入了**切入点**。**Advice+切入点🡺Advisor**



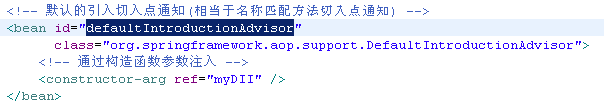
# 引入通知：增加新的属性和方法



## 创建引入：引入通知（代理引入拦截器）



需要对一个接口添加**引入通知**时，只需要创建一个类继承IntroductionInterceptor，并实现那个接口即可。

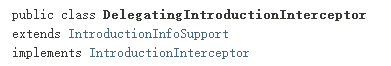


然后再那个

中，引入这个即可。就可以对被通知的类增添新的属性和方法了。

## 引入通知IntroductionInterceptor

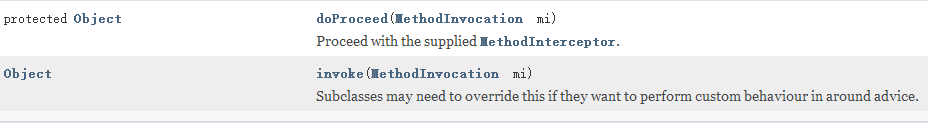
**Interceptor（接口）🡪** **IntroductionInterceptor（接口）—>** **DelegatingIntroductionInterceptor（实现类）**



**定义引入通知只需要继承DelegatingIntroductionInterceptor即可。**

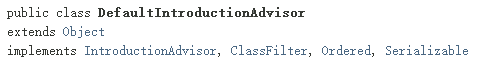
Convenient implementation of the [IntroductionInterceptor](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/aop/IntroductionInterceptor.html) interface.

**其有两个特有方法：（还有很多继承的方法）**

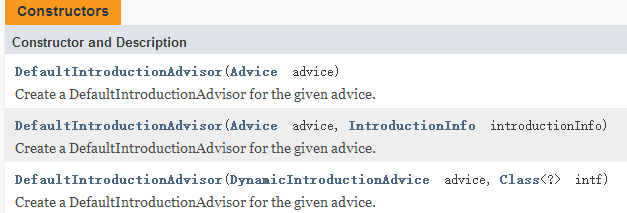


## DefaultIntroductionAdvisor ：对引入通知进行包装。

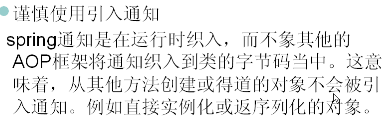
包：org.springframework.aop.support



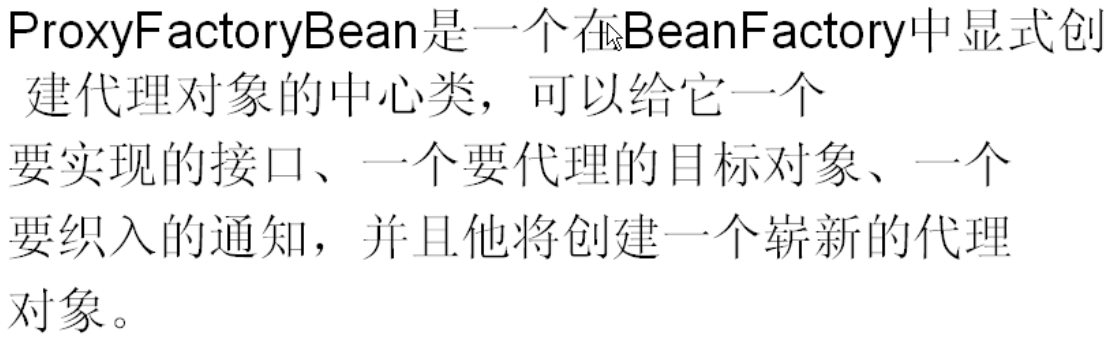
构造方法：



## 谨慎使用引用通知



# ProxyFactoryBean代理工厂Bean

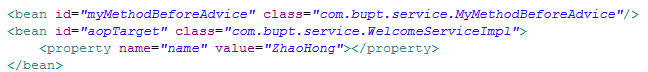


## 可以对接口进行代理

就是创建接口的实现类，从而代理接口进行操作。

## 可以对类进行代理

即创建被通知类（被代理类）的子类。





**对类进行代理**：（与目标类继承关系）



# Interceptor：拦截器。其实就是Advice。

**NameMatchMethodPointcutAdvisor**

**可以对Advice或Interceptor进行包装，增加了切入点。**



原理图：

