# Spring AOP的基本概念

Spring AOP提供了面向切面 编程的实现。面向切面编程在事务处理、日志记录、安全控制等操作中被广泛使用。

## AOP的基本概念

1. AOP（Aspect-Oriented Programming）即面向切面编程，与OOP（Object-Oriented P rogramming）面向对象编程相辅相成，提供了与OOP不同的抽象软件结构的视角。 在OOP中，以类作为程序的基本单元，而AOP中的基本单元是Aspect（切面）。
2. 在业务处理代码中通常有日志记录、性能统计、安全控制、事务处理、异常处理等 操作。尽管使用OOP可以通过封装或继承的方式达到代码的复用，但仍然有同样的 代码分散在各个方法中。因此，采用OOP处理日志记录等操作不仅增加了开发者的 工作量，而且提高了升级维护的困难。为了解决此类问题，AOP思想应运而生。

AOP采用横向抽取机制，将分散在各个方法中的重复代码提取出来，然后在程序编 译或运行阶段将这些抽取出来的代码应用到需要执行的地方。这种横向抽取机制， 采用OOP是无法办到的，因为OOP实现的是父子关系的纵向复用。但是AOP不是 OOP的替代品，而是OOP的补充，它们相辅相成。

## AOP常用术语

切面：切面（Aspect）是指封装横切到系统功能（例如事务处理）的类。（切面是类）

连接点：连接点（Joinpoint）是指程序运行中的一些时间点，例如方法的调用或异常的 抛出。（连接点是方法）

切入点：切入点（Pointcut）是指需要处理的连接点。在Spring AOP中，所有的方法执 行都是连接点，而切入点是一个描述信息，用于修饰连接点。通过切入点确定

哪些连接点需要被处理。（切入点是方法）

通知：通知（Advice）是由切面添加到特定的连接点（满足切入点规则）的一段代码， 。即在定义好的切入点处所要执行的程序代码。可以将其理解为切面开启后切面 的方法，因此通知是切面的具体实现。（通知是方法，而且还是切面中的）

引入：引入（Introduction）允许在现有的实现类中添加自定义的方法和属性。

目标对象：目标对象（Target Object）是指所有被通知的对象。如果AOP框架使用允许 时代理的方式（动态的AOP）来实现切面，那么通知对象总是一个代理对象。 （目标对象是对象）

代理：代理（Proxy）是将通知应用到目标对象之后被动态创建的对象。（织入产生的新 对象）

织入：织入（Weaing）是将切面代码插入到目标对象上，从而生成代理对象的过程。 根据不同的实现技术，AOP织入有3种方式：

·编译期织入，需要有特殊的Java编译器

·类装载期织入，需要有特殊的类装载器

·动态代理织入，在运行期为目标类添加通知生成子类的方式。

Spring AOP框架默认采用动态代理织入，而AspectJ（基于Java语言的AOP框

架）采用编译期织入和类装载期织入。（织入是一个组合的过程）

# 动态代理

在Java中有多种动态代理技术，例如JDK、CGLIB、Javassist、ASM，其中最常用的是动 态代理技术是JDK和CGLIB。目前，在Spring AOP中常用JDK和CGLIB两种动态代理技 术。

Spring AOP底层实现的原理是采用动态代理技术实现的。在Spring AOP中常用的动态代 理技术是JDK动态代理和CGLIB字节码增强两种动态代理技术。这两种代理技术各有用 途， JDK动态代理需要借助接口、CGLIB不需要借助接口。应根据实际情况使用，两者 无优劣之分。

## JDK动态代理

JDK动态代理是java.lang.reflect.\*包提供的方式，它必须借助一个接口才能产生代理对 象。因此，对于使用业务接口的类，Spring默认使用JDK动态代理实现AOP。通过一个 示例演示如何使用JDK动态代理实现Spring AOP：

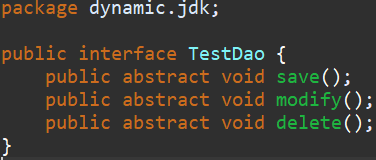
1. **创建应用**

创建一个名为ch4的Web应用，在导入所需要的JAR包。

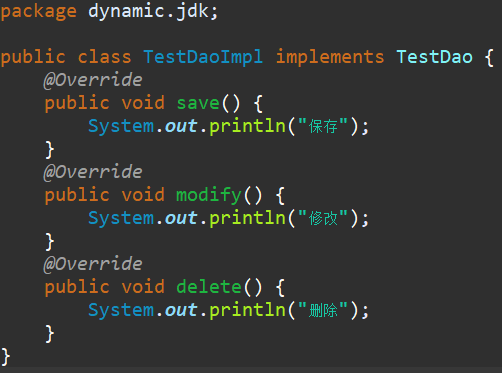
1. **创建接口及实现类**

在ch4的src目录下创建一个dynamic.jdk包，并在该包中创建接口TestDao和实现类TestDaoImpl，该实现类作为目标类，在代理类中对其方法进行增强处理。

TestDao接口代码如下：



TestDaoImpl实现类代码如下：



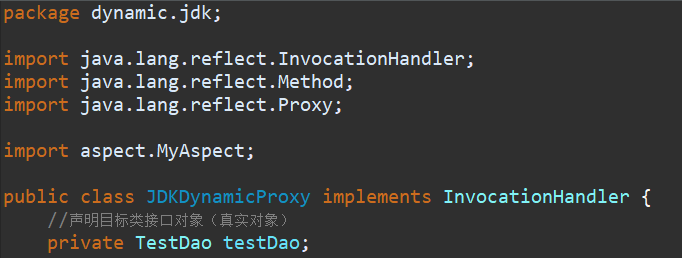
1. **创建切面类**

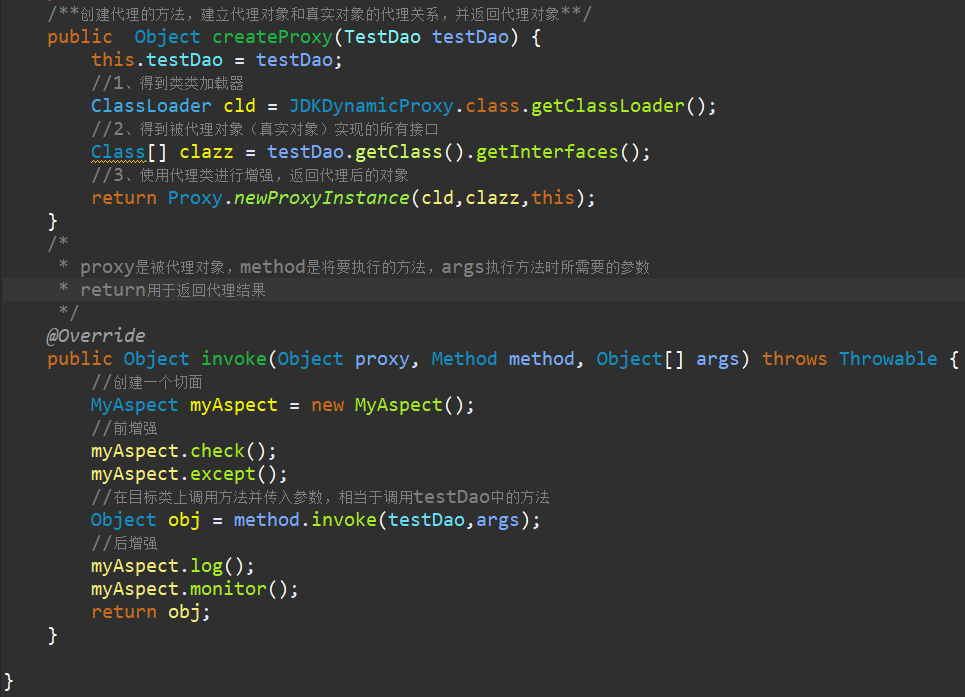
在ch4的src目录下创建一个aspect包，并在该包中创建切面类MyAspect，注意在该类中可以定义多个通知（增加处理的功能方法）。代码如下：



1. **创建代理类**

在dynamic.jdk包中创建代理类JDKKynamicProxy。在JDK动态代理中代理类必须实现java.lang.reflect.InvocationHandler接口，并编写代理方法，在代理方法中需要通过java.lang.reflet.Proxy实现动态代理。





1. **创建测试类**

在dynamic.jdk包中创建测试类JDKDynamicTest。在主方法中创建代理对象和目标对象，然后从代理对象中获取对目标对象增强后的对象，最后调用该对象的添加、修改和删除方法。JDKDynamicTest类代码如下：



运行程序，可以看出目标方法实现类增强。

## CGLIB动态代理

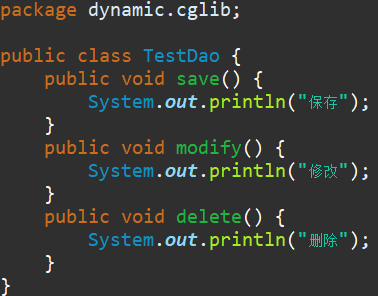
JDK动态代理必须提供接口才能使用，即要求增强的类必须实现接口，对于没有实 现接口的类，只能采用CGLIB动态代理。

CGLIB（Code Generation Library）是一个高性能开源的代码生成包，采用非常底层的字节码技术，对指定的目标类生成一个子类，并对子类进行增强。在spring-core.jar.5.0.2包中已经集成了CGLIB所需要的类。

通过一个示例演示CGLIB动态代理的实现过程：

1. **创建目标类**

在先前创建的ch4应用中的src目录下创建一个dynamic.cglib包，在该包中创建目标类TestDao，注意该目标类不需要实现任何接口。代码如下：



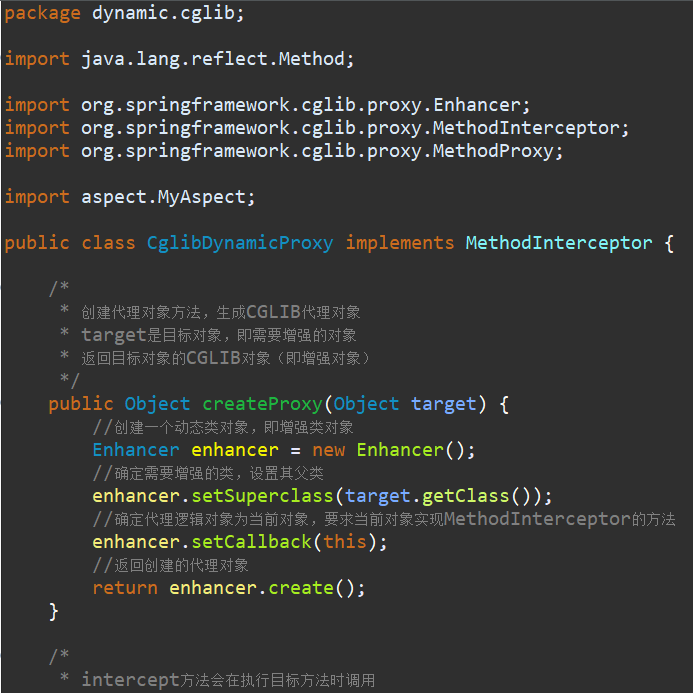
1. **创建切面类**

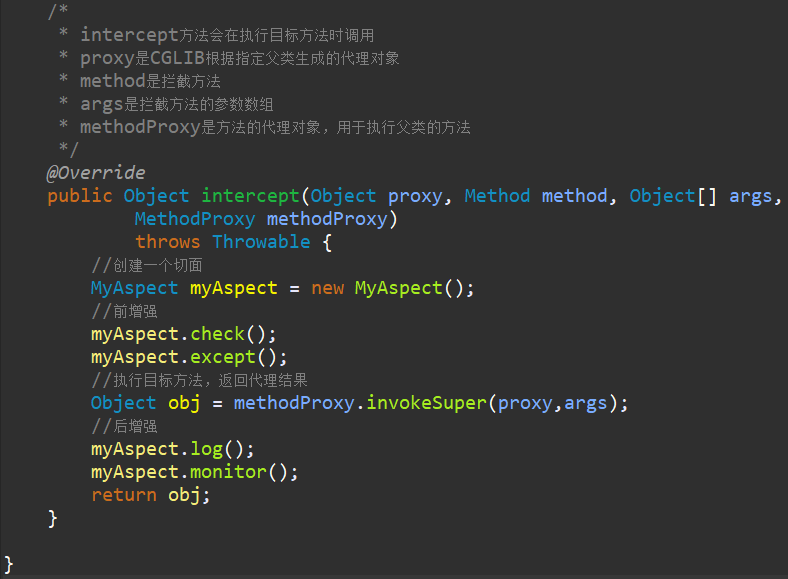
在ch4的src目录下创建一个aspect包，并在该包中创建切面类MyAspect，注意在该类中可以定义多个通知（增加处理的功能方法）。代码如下：



1. **创建代理类**

在dynamic.cglib包中创建代理类CglibDynamicProxy，该类实现MethodInterceptor 接口。代码如下：





1. **创建测试类**

在dynamic.cglib包中创建测试类CglibDynamicTest。在主方法中创建代理对象和目标对象，然后从代理对象中获取对目标对象增强后的对象，最后调用该对象的方法，查看方法是否进行增强处理。代码如下：



查看运行结果，可以看出对目标方法进行了增强。

# Spring AOP中的通知

## 通知类型

在Spring中根据通知在目标类中的连接点（切入点）位置，通知可以分为6种类型。

1. 环绕通知（org.dopalliance.intercept.MethodInterceptor）

环绕通知是在目标方法执行前和执行后实施增强，可应用于日志记录、事务处理等 功能。

1. 前置通知（org.springframework.aop.MethodBeforeAdvice）

前置通知是在目标方法执行前实施增强，可应用于权限管理等功能。

1. 后置返回通知（org.springframework.aop.MethodReturningAdvice）

后置返回通知是在目标方法成功执行后实施增强，可应用于关闭流、删除临时文件 等功能。

1. 后置通知（org.srpingframework.aop.AfterAdvice）

后置（最终）通知是在目标方法执行后实施增强，与后置返回通知不同的是，不管 是否发生异常都要执行该类通知，可应用于释放资源。

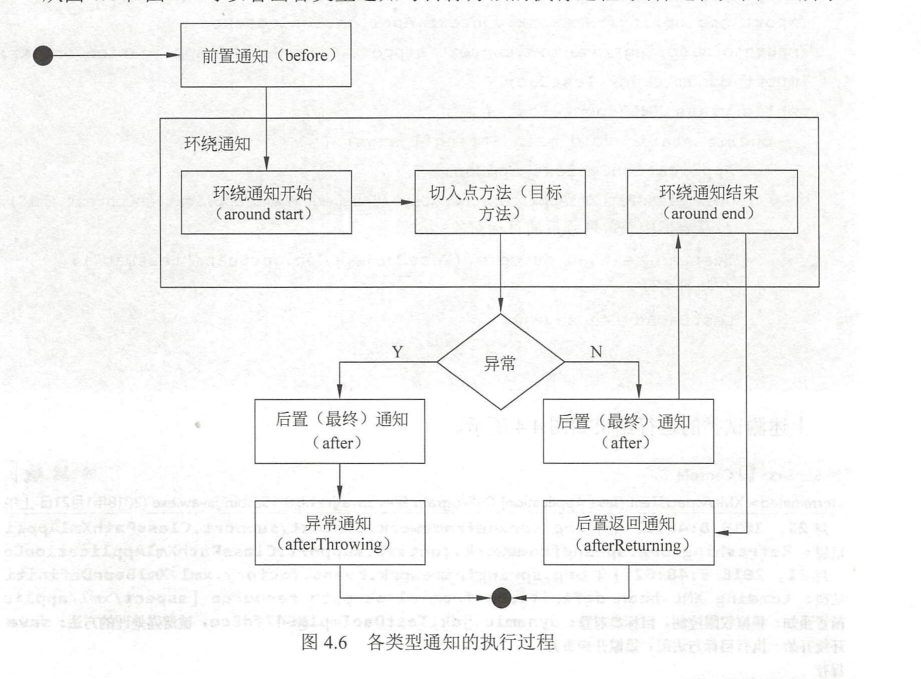
1. 异常通知（org.springframework.aop.ThrowsAdivce）

异常通知是在目标方法（切入点）抛出异常后实施增强，可应用于异常处理、记录 日志等。

1. 引入通知（org.springframework.aop.IntroductionInterceptor）

引入通知是在目标类中添加一些新的方法和属性，可应用于修改目标类（增强类）。

## 通知的执行流程



# 实现AOP

## 使用代理类的实现AOP

使用org.springframework.beans.factory.ProxyFactoryBean创建代理是实现Spring AOP 最基本的实现方式，ProxyFactoryBean是org.springframework.beans.factory.FactoryBea n接口的实现类，FactoryBean负责实例化一个Bean实例，ProxyFactoryBean负责为其 他Bean实例创建代理实例。ProxyFactoryBean类的常用属性如下所示：

|  |  |
| --- | --- |
| **属性** | **描述** |
| target | 代理的目标对象 |
| proxyInterfaces | 代理需要实现的接口列表，如果是多个接口，使用以下格式赋值：  <list>  <value></value>  ......  <list> |
| interceptorNames | 需要织入的目标的advice（指定切面） |
| proxyTargetClass | 是否对类代理而不是接口，默认为false，使用JDK动态代理；当为true时，使用CGLIB动态代理 |
| singleton | 返回的代理实例是否为单例，默认为true |
| optimize | 当设置为true时强制使用CGLIB动态代理 |

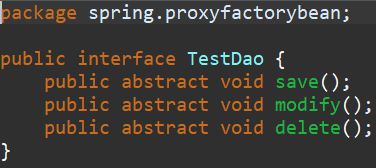
示例如下：

1. **导入相关的JAR包**

在核心JAR包的基础上需要向ch4应用的WEB-INF/lib目录下导入JAR包，spring-aop-5.0.2.RELELALSE.jar和aopallicance-1.0.jar。aopalliance-1.0.jar是AOP联盟提供的规范包，可以在<https://mvnrepository.com/artifact/aopalliance/aopalliance>地址下载

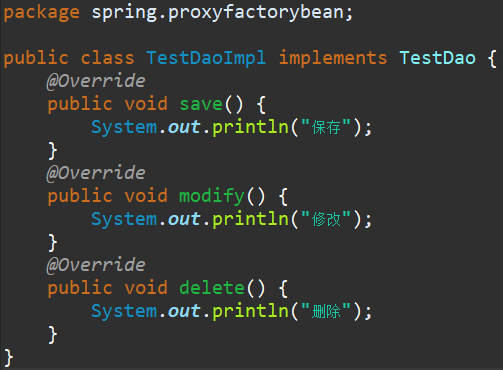
**2）创建目标类要实现的接口**

在src目录下创建spring.proxyfactorybean包，在包中创建TestDao接口，代码如下：



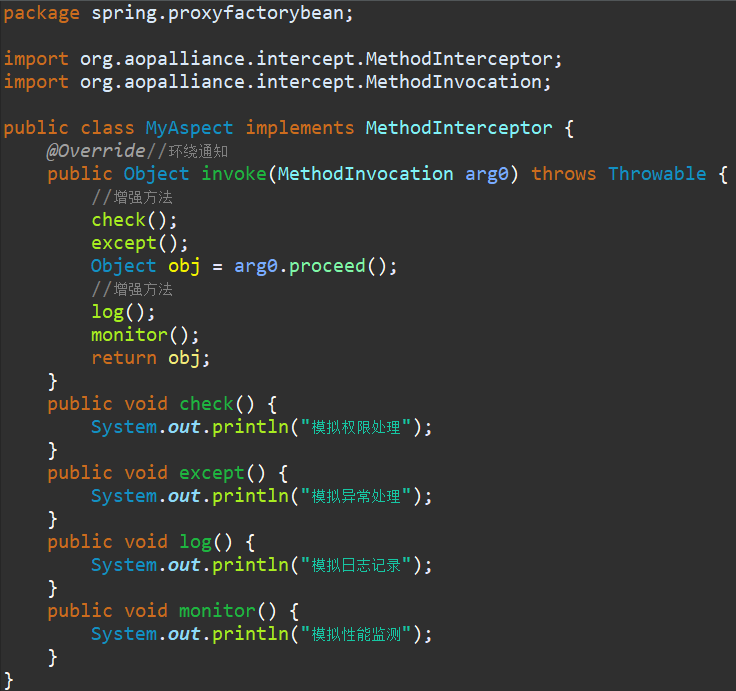
**3）创建目标类**

在spring.proxyfactorybean包中创建目标类，该目标类实现了TestDao接口，代码如下：



1. **创建切面类**

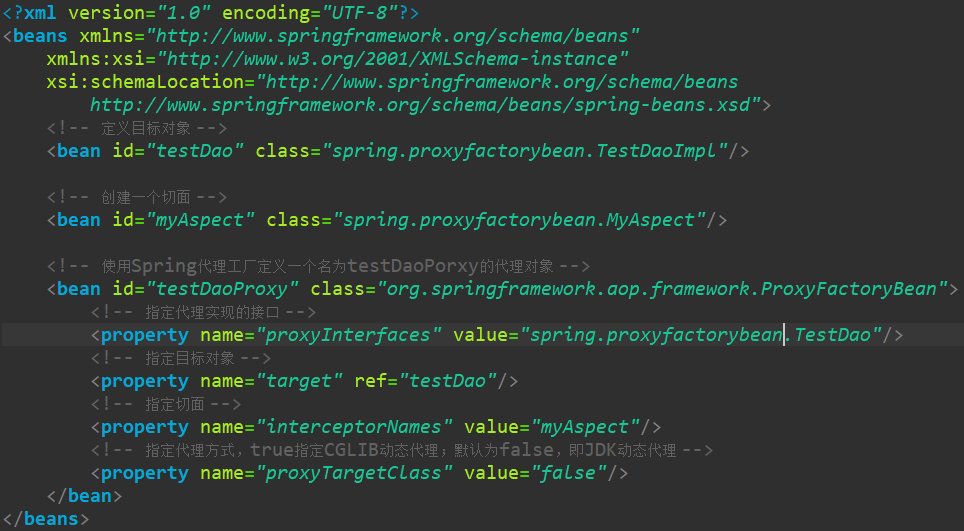
由于该实例实现环绕通知，所以切面类需要实现org.aopalliance.intercept.MethodInterceptor接口。在src目录下创建一个spring.proxyfactorybean包，在该包中创建切面类MyAspect，代码如下：



切面类实现了org.aopalliance.intercept.MethodInterceptor接口，表示实现环绕通知， 并覆写了一个相应的方法，在方法中完成环绕通知想要完成的功能。如果需要实现 其他通知，如前置、后置等通知，则实现通知对应的接口即可，然后覆写相应的方 法。

1. **配置切面并指定代理**

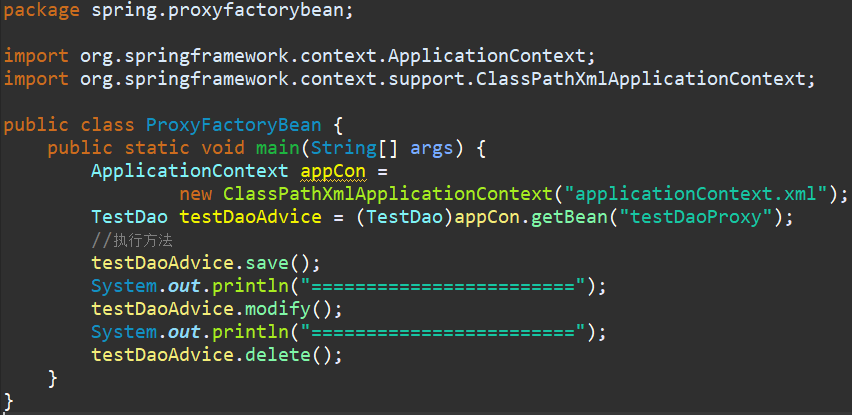
切面类需要配置为Bean实例，这样Spring容器才能识别为切面对象，在src目录下创建Spring配置文件applicationContext.xml，在Spring配置文件中配置切面和指定代理对象。代码如下：



在上述配置文件中首先通过<bean>元素定义了目标对象和切面，然后使用ProxyFactoryBean类定义了代理对象；由于要增强的目标类实现了接口，所以只能使用JDK动态代理，将proxyTargetClass属性的值设置为false。

1. **创建测试类**

在spring.proxyfactorybean包中创建测试类ProxyFactoryBeanTest，在主方法中使用Spring容器获取代理对象，并执行目标方法。代码如下：



运行结果，观察调用的目标类中的方法是否增强。

## 使用AspectJ实现AOP

AspectJ是一个基于Java语言的AOP框架，对于目前的Spring框架，建议开发者使用 AspectJ开发Spring AOP。

使用AspectJ实现AOP需要导入spring-aspects-5.0.2.RELEASE.jar和aspectjweaver-1.8.13. jar包。

使用AspectJ实现Spring AOP有两种方式：一是基于XML配置开发AspectJ，二是基于 注解开发AspectJ。

### 基于XML配置开发AOP

基于XML配置开发AspectJ是指通过XML配置文件定义切面、切入点及通知，所有这 些定义都必须在<aop:config>元素内。<aop:config>元素及其子元素如下所示：

|  |  |
| --- | --- |
| **元素名称** | **用途** |
| <aop:config> | 顶层配置元素，在配置文件的<beans>下可以包含多个该元素 |
| <aop:aspect> | 定义一个切面，<aop:config>元素的子元素，属性ref指定切面的定义 |
| <aop:poincut> | 配置切入点，<aop:aspect>元素的子元素，属性expression指定通知增强哪些方法，即要增强哪些方法，可以是切入点表达式 |
| <aop:before> | 配置前置通知，<aop:aspect>元素的子元素，属性method指定前置通知方法，属性pointcut-ref指定管理的切入点 |
| <aop:after-returning> | 配置后置返回通知，<aop:aspect>元素的子元素，属性method指定后置返回通知方法，属性poincut-ref指定关联的切入点 |
| <aop:around> | 配置环绕通知，<aop:aspect>元素的子元素，属性method指定环绕通知方法，属性pointcut-ref指定关联的切入点 |
| <aop:after-throwing> | 配置异常通知，<aop:aspect>元素的子元素，属性method指定异常通知方法，属性pointcut-ref指定关联的切入点，没有异常发生时将不会执行 |
| <aop:after> | 配置后置(最终)通知，<aop:aspect>元素的子元素，属性method指定后置（最终）通知方法，属性pointcut-ref指定关联的切入点 |
| <aop:declare-parents> | 给通知引入新的额外接口，增强功能（不要求掌握该类型的通知） |

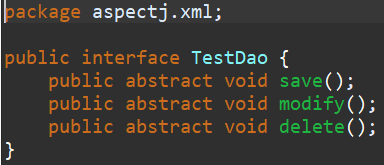
通过一个示例演示使用AspectJ基于XML配置的方法开发AOP：

1. **导入AspectJ框架相关的JAR包**

向ch4应用的WEB-INF/lib目录中导入JAR包spring-aspects-5.0.2.RELEASE.jar和aspectjweaver-1.8.13.jar。spring-asepcts-5.0.2.RELEASE.jar是Spring为AspectJ提供的实现，在Spring安装包目录已提供，aspectjweaver-1.8.13.jar是AspectJ框架所提供的规范包，可以在<https://mvnrepository.com/artifact/org.aspectj/aspectjweaver> 下载。

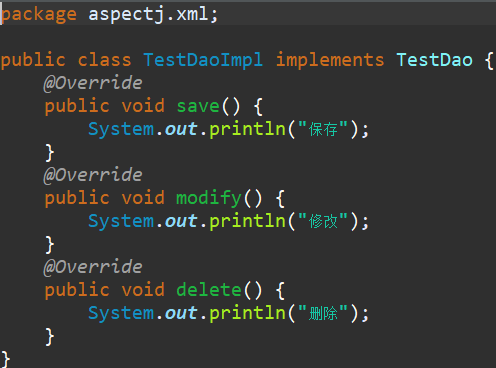
1. **创建目标类接口**

在src目录下创建aspectj.xml包，在包中创建接口TestDao，代码如下：



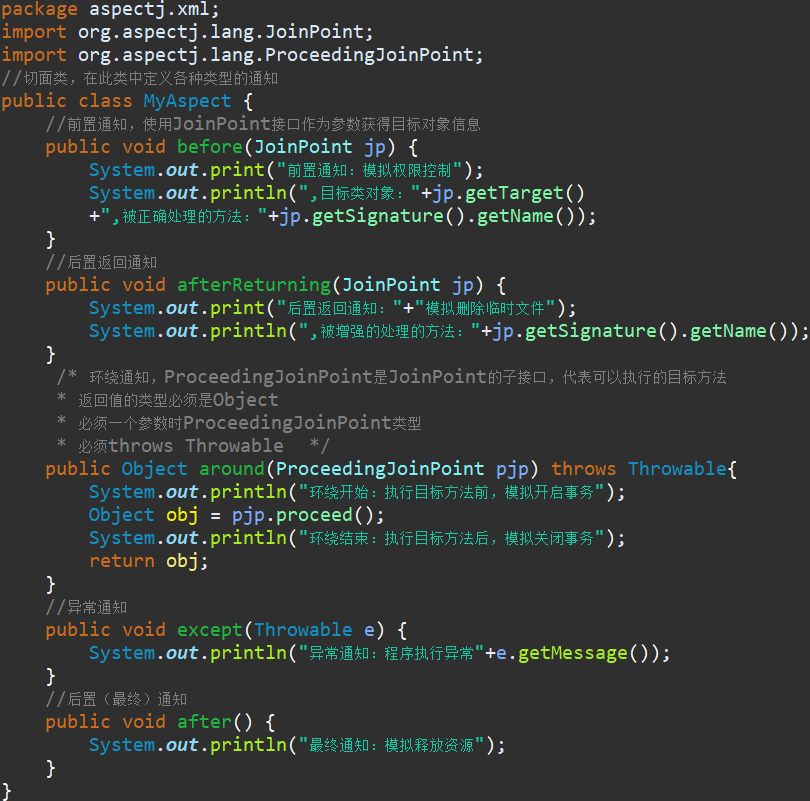
1. **创建目标类**

在aspectj.xml包中创建目标类TestDaoImpl，充当目标类。代码如下：



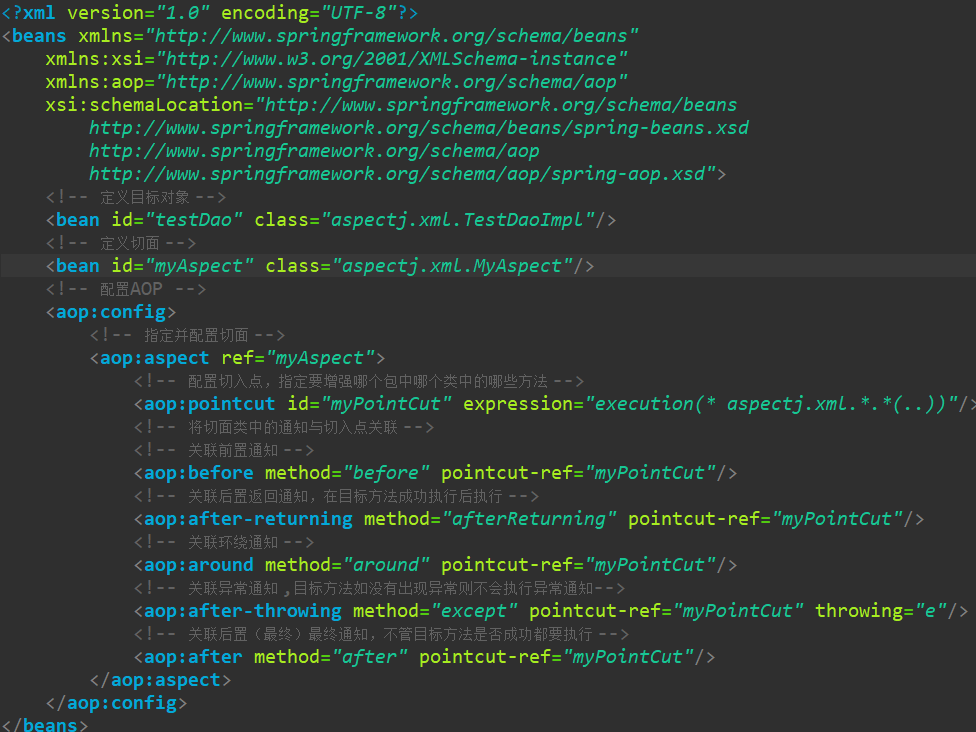
1. **创建切面类**

在aspectj.xml包中创建切面类MyAspect，并在该类中编写各种类型的通知。代码如下：



1. **创建Spring配置文件，并配置AOP**

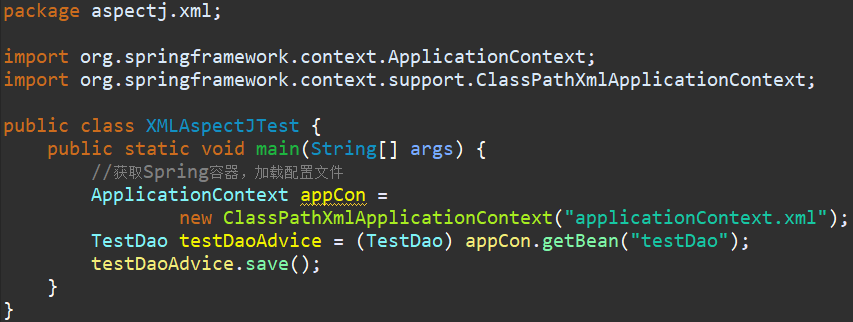
在src目录下创建Spring配置文件applicationContext.xml，并使用<aop:config>元素及子元素配置AOP。代码如下：



注意，使用AspectJ基于XML配置实现AOP需要引入aop声明文件。在使用<aop:point>元素指定要增强哪些方法时，是通过该元素的expression属性定义切入点表达式，在切入点表达式中的意思是aspectj.xml包中任意方法的执行。其中execution(\* aspectj.xml.\*.\*(..))是表达式的主体，第一个\*表示的是返回类型，使用\*代表所有类型；aspectj.xml表示需要匹配的包名；后面第二个\*表示的是类名，使用\*表示匹配包中所有的类；第三个\*表示的方法名，使用\*表示匹配所有方法；后面的（..）表示方法的参数，其中..表示任意参数。另外，第一个\*和包名之间有一个空格。

**6）创建测试类**

在aspectj.xml包中创建测试类XMLAspectJTest，在主方法中调用目标类中的方法，查看是否增强处理。代码如下：



运行程序，查看目标类中的方法是否进行了增强。如果要观察异常通知是否执行， 可以在目标类方法中加入会产生异常的代码，如int i=10/0。

### 基于注解开发开发AOP

基于注解开发要比基于XML配置开发Aspect便捷许多，在实际开发中也推荐使用注解 的方式开发AOP。AspectJ基于注解开发的方式中，常用注解如下：

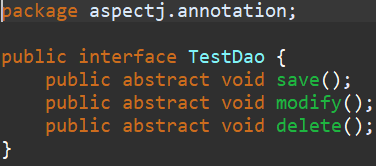
|  |  |
| --- | --- |
| **注解名称** | **描述** |
| @Aspect | 用于定义一个切面，标注在切面类上 |
| @Pointcut | 用于定义切入点表达式。在使用时需要定义一个切入点方法，该方法是一个返回值为void且方法体为空的普通方法，将该注解标注在此方法上 |
| @Before | 用于定义前置通知。在使用时通常为其指定value属性值，该值可以是已有的切入点，也可以直接定义切入点表达式 |
| @AfterReturning | 用于定义后置返回通知。在使用时通常为其指定value属性值，该值可以是已有的切入点，也可以直接定义切入点表达式 |
| @Around | 用于定义环绕通知。在使用时通常为其指定value属性值，该值可以是已有的切入点，也可以直接定义切入点表达式 |
| @AfterThrowing | 用于定义异常通知。在使用时通常为其指定value属性值，该值可以是已有的切入点，也可以直接定义切入点表达式。另外，还有一个throwing属性用于访问目标方法抛出的异常，该属性值与异常通知方法中同名的形参一致 |
| @After | 用于定义后置（最终）通知。在使用时通常为其指定value属性值，该值可以是已有的切入点，也可以直接定义切入点表达式。 |

通过一个示例讲解基于注解开发AspectJ的过程：

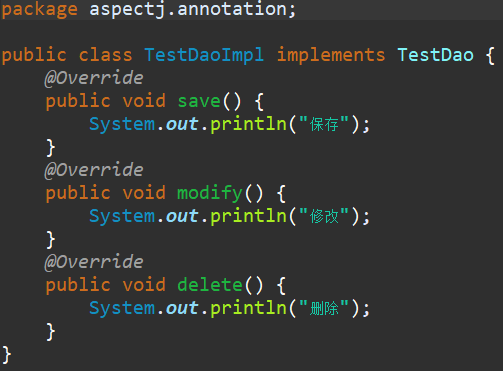
1. **创建目标类接口和目标类**

在先前创建ch4应用中的src目录下创建aspectj.annotation包，并在该包中创建TestDao接口和其实现类TestDaoImpl，TestDaoImpl充当目标类。

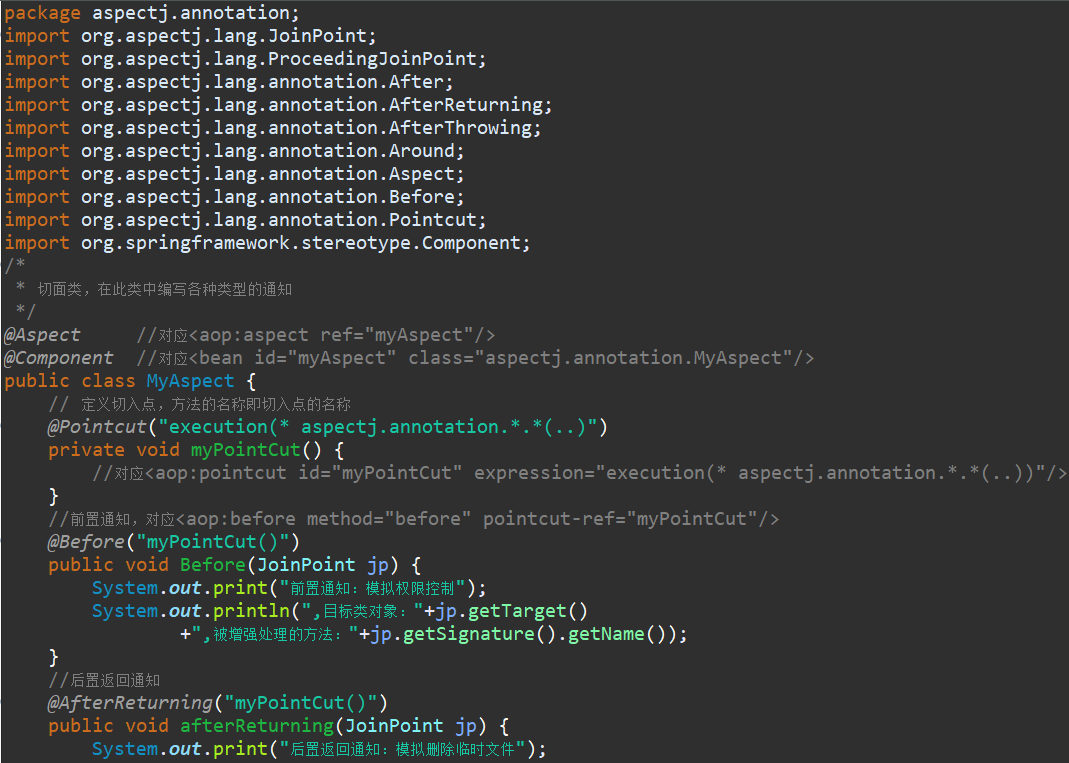
TestDao接口代码如下：

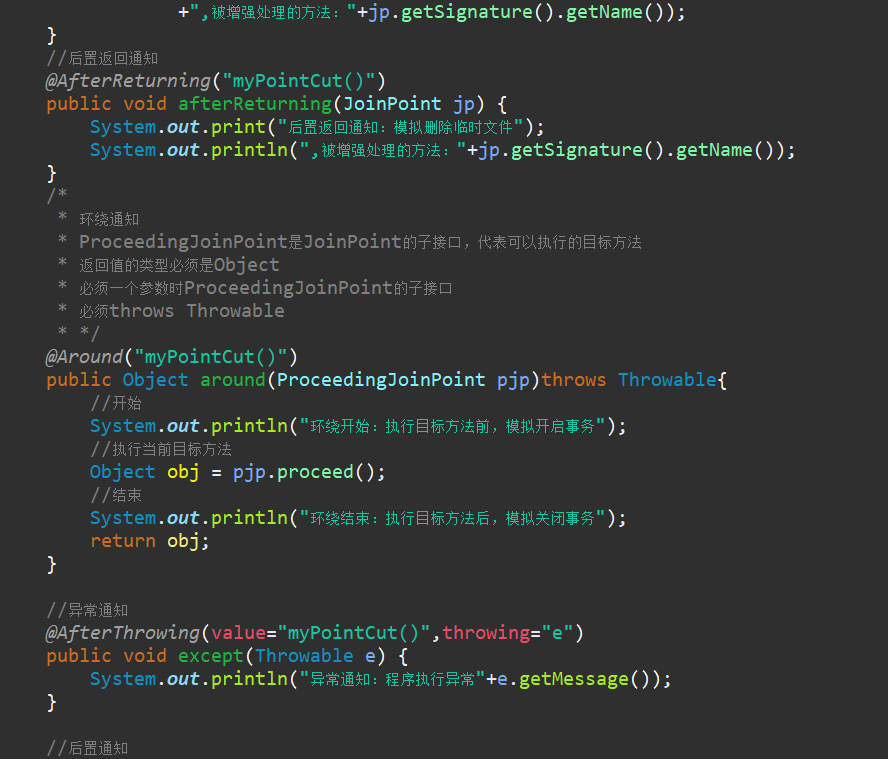


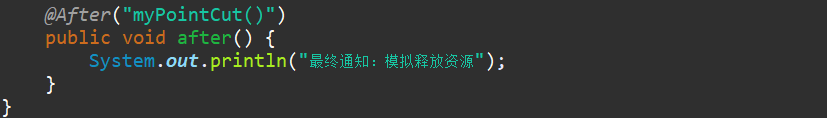
TestDaoImpl类代码如下：



1. **创建切面类**

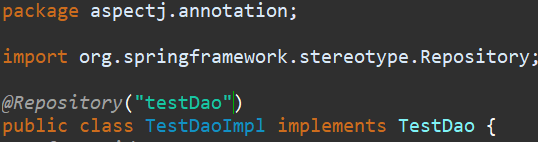
在aspectj.annotation包中创建切面类MyAspect。在该类中需要使用@Aspect注解将该类定义为切面类，由于该类在Spring中是作为组件使用的，所以还需要使用@Component注解；然后使用@Pointcut注解切入点表达式，并通过定义方法来表示切入点名称；最后在每个通知方法上添加相应的注解，并将切入点名称作为参数传递给定义的通知方法。





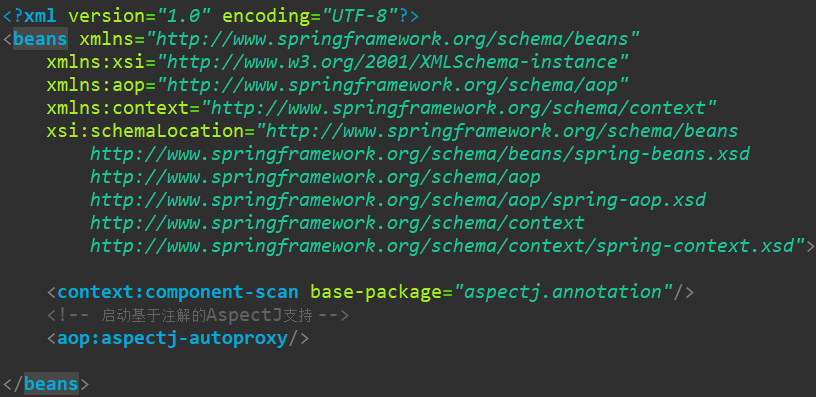
1. **注解目标类**

将aspectj.annotation包中的TestDaoImpl类使用@Repository注解托管给Spring容器。



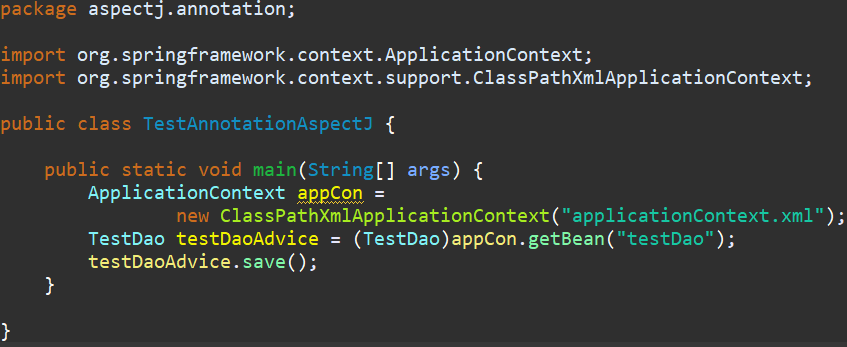
1. **配置文件**

在Spring配置文件中指定需要扫描的包，使标注的注解生效。同时需要启动基于注解的AspectJ支持。



1. **测试**

编写测试类，调用目标类中的方法，查看是否增强。



运行程序，查看调用的目标类方法是否得到了增强。