## Spring AOP技术

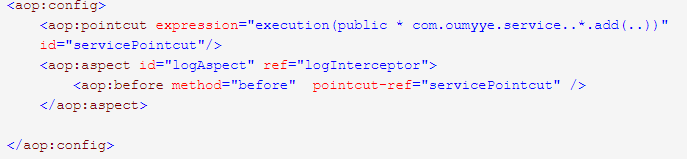
AOP—面向切面编程，是一种对OOP的一种补充，能够将多个方法之前，或者之后的关注点抽取出来，进行模块化，重复使用，避免了代码的冗余。

### AOP术语

* Join point：连接点，在Spring AOP中，连接点其实就是一个具体方法的前后左右，或者抛异常的时候。
* Pointcut：切点，可以理解为一系列连接点的集合，它使用一种正则表达式，任何匹配该表达式的连接点，都可以被功能增强。
* Advice：通知，又可以称为增强，它会在具体的某一连接点采取行动，通知包括”before”,”after”,”around”,”afterReturn”，指定了具体什么时候会执行增强，以及增强的内容是什么，
* Aspect：切面，是PointCut与Advice的结合，它定义了在哪些方法，什么时候，执行怎样的增强。

Aop是通过给被增强对象创建代理的方式来实现，一种是使用java动态代理的方式来实现，动态代理需要使用接口来创建代理，但如果需要增强的方法不在接口中，那么就需要使用gclib来创建代理，gclib是jvm运行的时候给具体的对象创建一个子类，子类拥有父类的接口，这样就可以在调用父类方法之前或之后对父类方法进行增强，但是不存在一个.class文件，因为是动态生成的。具体使用动态代理还是使用gclib这个是由spring框架决定。

### Aop基于xml配置：



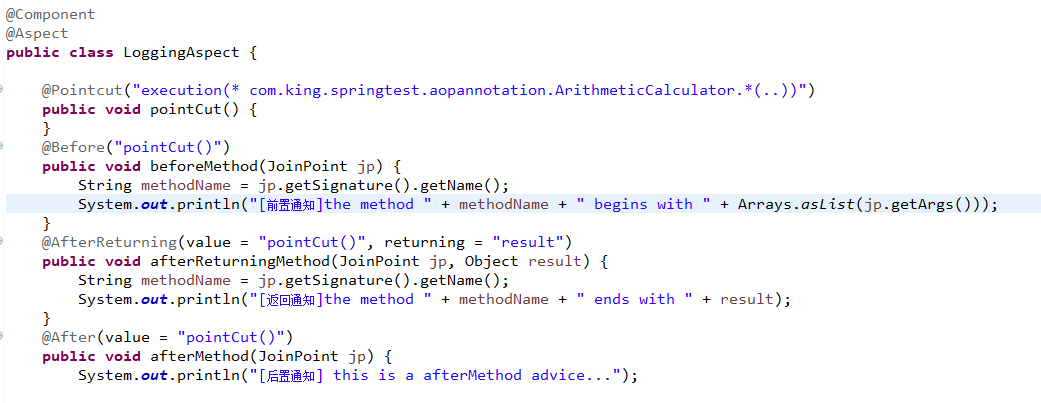
<aop:pointcut>是切点标签，expression指定匹配连接点的正则表达式，意思就是说需要在哪些方法上执行增强。一般来说是指定接口的方法，调用实现类的方法的时候就会进行增强。

<aop:aspect>是切面标签，用于整合advice以及pointcut。ref指定的是执行增强的那个类，增强类是一个pojo，并没有什么特殊。

<aop:before>定义了在切点之前进行增强，pointcut-ref关联了之前定义的pointcut。当然还有其他的advice例如after，afterRerun。

### Aop基于注释配置

基于xml文件的Aop配置方式比较繁琐，基于注解的配置能够保证开发思维的连贯性。



基于注解的所有配置都集成在一个java类中，它并不是一个普通的pojo类，他是一个切面类，@Aspect意味着spring会将它内部的注解进行Aop解析，但是@Aspect并不会被ioc容器解析，不会被ioc容器扫描然后加入ioc容器中，所以需要使用@Component来题型ioc容器把它加载进去。

@Pointcut注解是定义了切点，需要定义在一个方法上面例如pointCut()方法，到时候advice引用的时候直接引用该方法的名称，也就是value=”pointCut()”属性，这样spring就可以根据方法来获取到方法上的@Pointcut标签，从而真正的获取到切点定义。

@Before注解是定义了advice执行的时间，before意味着在代理对的方法之前执行，括号中的值是注有@Pointcut的方法的名称。

Advice方法可以传入参数，JoinPoint是对连接点的一个封装，当代理对象调用advice方法的时候，可以对连接点相关的信息做一些处理，例如可以获取到连接点的签名，连接点方法参数，被代理对象的class等等。当advice是@AfterReturnning的时候，就可以将被代理的方法的返回值注入到adivce方法中，例如注释中参数(value=”Pointcut”,returning=”result”)，returning的值就是advice方法中那个参数的名称，意思就是被代理方法有返回值的话就把这个返回值传入到advice方法中名称跟returnning的值相同的对象中。

### AOP源码解析

AOP的本质其实是动态代理，在创建动态代理的时候，会把增强放入到动态代理当中，然后在真正调用target方法之前或者之后去调用那些增强方法。

基于注解式的AOP过程如下：

* 首先，从BeanFactory中找到所有带有@Aspect注解的类，
* 将类中的标注有增强的方法都封装起来，保存@Aspect类名以及对应方法，到时候可以调用反射来执行增强。
* 对于前置增强需要封装成interceptor，而后置增强不需要
* 找到target对象的所有相关的增强，并将其放入ProxyFactory中。
* 通过ProxyFactory可以创建一个代理，在调用target方法之前或者之后调用增强