浅谈我眼里的 AOP

作者：xiaoli
原文链接：https://ld246.com/article/1538123788523
来源网站：链滴

不管你是在校大学生还是已在工作岗位的苦逼程序员，只要你使用过spring，那么你对该名词一定会熟悉。如果有人让你解释下它，我们脑子里出于本能的一下会蹦出来这些东西，他是面向切面的编程切面、切点等等。但是它们是什么，是怎么运作的，以及会运用在项目哪些方面？这一系列的疑问，让我来抛砖引玉好好聊聊

1 关于AOP的官方解释

AOP称为面向切面编程，在程序开发中主要用来解决一些系统层面上的问题，比如日志，事务，权限待。还记得Struts2的拦截器么，他就是基于AOP的设计思想。

2 关于AOP的基本概念你必须知道

1. 切面（Aspect） 通常是一个类，里面可以定义切入点和通知

2. 连接点（JointPoint） 程序执行过程中明确的点，一般是方法的调用

3. 通知（Advice） AOP在特定的切入点上执行的增强处理，有before, after, afterReturning, afte Throwing, around

4. 切入点(Pointcut) 就是带有通知的连接点，在程序中主要体现为书写切入点表达式

5. AOP代理 AOP框架创建的对象，代理就是对目标对象的增强，Spring中的AOP代理可以使JDK态代理，也可以是CGLIB代理，前者基于接口，后者基于子类

3 关于Spring AOP你必须知道的

Spring中的AOP代理还是离不开Spring的IOC容器，代理的生成，管理及其依赖关系都是由IOC容器责，Spring默认使用JDK动态代理，在需要代理类而不是代理接口的时候，Spring会自动切换为使用GLIB代理，不过现在的项目都是面向接口编程，所以JDK动态代理相对来说用的还是多一些。

4 基于注解的AOP

1. 在进入正题之前你需要了解各种通知类型

(1)Before:在目标方法被调用之前做增强处理,@Before只需要指定切入点表达式即可

(2)AfterReturning:在目标方法正常完成后做增强,@AfterReturning除了指定切入点表达式后，还可指定一个返回值形参名returning,代表目标方法的返回值

(3)AfterThrowing:主要用来处理程序中未处理的异常,@AfterThrowing除了指定切入点表达式后，可以指定一个throwing的返回值形参名,可以通过该形参名来访问目标方法中所抛出的异常对象

(4)After:在目标方法完成之后做增强，无论目标方法时候成功完成。@After可以指定一个切入点表达式

(5)Around:环绕通知,在目标方法完成前后做增强处理,环绕通知是最重要的通知类型,像事务,日志等都环绕通知,注意编程中核心是一个ProceedingJoinPoint

2. 启用@AsjectJ支持

首先在spring配置文件中配置下面一句:

3. 切面创建实例

```
/**
* @description: 日志切面
* @author: xuzhiwei-009
* @create: 2018-09-28 15:49
*/
@Aspect
@Component
public class LogAspect {
/**
* 定义切入点
*/
@Pointcut("execution(* com.mrxuxu.demo.service..*.*(..))")
public void pointCut(){}
  @Before("pointCut()")
  public void deBefore(JoinPoint joinPoint){
    System.out.println("AOP Before Advice...");
  }
  @After("pointCut()")
  public void doAfter(JoinPoint joinPoint){
    System.out.println("AOP After Advice...");
  }
  @AfterReturning(pointcut="pointCut()",returning="returnVal")
  public void afterReturn(JoinPoint joinPoint,Object returnVal){
    System.out.println("AOP AfterReturning Advice:" + returnVal);
  }
  @AfterThrowing(pointcut="pointCut()",throwing="error")
  public void afterThrowing(JoinPoint joinPoint,Throwable error){
```

```
    System.out.println("AOP AfterThrowing Advice..." + error);
    System.out.println("AfterThrowing...");
  }
  @Around("pointCut()")
  public void around(ProceedingJoinPoint pjp){
    System.out.println("AOP Aronud before...");
    try {
      pjp.proceed();
    } catch (Throwable e) {
      e.printStackTrace();
    }
    System.out.println("AOP Aronud after...");
  }
}
```

5 基于XML配置的AOP

1. xml配置

```
<aop:config>
  <aop:aspect id="loggerAspect" ref="logger">
    <aop:around method="record" pointcut="(execution(* com.aijava.distributed.ssh.service..*.
dd*(..))
        or execution(* com.aijava.distributed.ssh.service..*.update*(..))
        or execution(* com.aijava.distributed.ssh.service..*.delete*(..)))
        and !bean(logService)"/>
  </aop:aspect>
</aop:config>
```

2. 切面定义

```
/**
* 切面
*/
@Component
public class Logger {
  public Object record(ProceedingJoinPoint pjp){

  }
}
```

6 话题之外，面试的时候面试官问起aop的时候千万不要着，第一它并不复杂，第二你在项目真的是使用过的

问： 你使用过aop吗

答：使用过啊，如果采用spring来开发项目的话，都会用到吧

问：你为什么会在项目中使用aop，或者说在哪些场景中使用到aop

答：其实在spring项目开发中，一般必然会用到aop的地方，最常见的就是service层中的事务处理，个就是出现错误时的错误回滚。抛开这个必然，其实在日常开发中在另外两种场景中使用到aop。第一:以前在做某一个模块开发的时候，由于该模块的业务相当复杂，数据流也比较大导致通过之前的日志件难以排查问题，这个时候我就想着引入aop来对这个模块进行特殊的日志记录。第二就是权限控制一个经典的使用案例就是spring security,它就是基于aop的思想实现的，所有的接口调用都会统一处。

7 话题之外，当问起为什么要使用aop（aop的用处、aop的点），你可以回答

1 实现代码的复用（避免重复冗余的业务代码）

2 解耦（避免强耦合带来的负面影响）