Spring AOP 面向切面编程:理解篇

一、到底什么是 AOP（面向切面编程）？

无论在学习或者面试的时候，大家都会张口说 spring 的特性 AOP 和 IOC(控制反转咱们下一篇讲)，有些大神理解的很到位，但是对于大多数初中级工程师来讲还是模糊阶段，但是为什么会有 AOP 这种技术呢？傻瓜都知道：为了开发者的方便！！！！就是为了我们少写代码，省劲！要记住上面我说的话！

下面我举个例子给大家说明一下：

有 A，B,C 三个方法，但是在调用每一个方法之前，要求打印一个日志：某一个方法被开始调用了！

在调用每个方法之后，也要求打印日志：某个方法被调用完了！

一般人会在每一个方法的开始和结尾部分都会添加一句日志打印吧，这样做如果方法多了，就会有很多重复的代码，显得很麻烦，这时候有人会想到，为什么不把打印日志这个功能封装一下，然后让它能在**指定**的地方（比如执行方法前，或者执行方法后）**自动**的去调用呢？如果可以的话，业务功能代码中就不会掺杂这一下其他的代码，所以 AOP 就是做了这一类的工作，比如，日志输出，事务控制，异常的处理等。。

如果把 AOP 当做成给我们写的“业务功能”增添一些特效，就会有这么几个问题：

1.我们要制作哪些特效

2.这些特效使用在什么地方

3.这些特效什么时候来使用

二、有了这三个疑问，加上上面的讲解，下面我们来说一下 AOP 的一些术语（一下看不懂不要紧，慢慢理解）

1.通知（Advice）

就是你想要的功能，也就是上面说的 安全，事物，日志等。你给先定义好把，然后在想用的地方用一下。

2.连接点（JoinPoint）

这个更好解释了，就是 spring 允许你使用通知的地方，那可真就多了，基本每个方法的前，后（两者都有也行），或抛出异常时都可以是连接点，spring 只支持方法连接点.其他如 aspectJ 还可以让你在构造器或属性注入时都行，不过那不是咱关注的，只要记住，和方法有关的前前后后（抛出异常），都是连接点。

3.切入点（Pointcut）

上面说的连接点的基础上，来定义切入点，你的一个类里，有 15 个方法，那就有几十个连接点了对把，但是你并不想在所有方法附近都使用通知（使用叫织入，以后再说），你只想让其中的几个，在调用这几个方法之前，之后或者抛出异常时干点什么，那么就用切点来定义这几个方法，让切点来筛选连接点，选中那几个你想要的方法。

4.切面（Aspect）

切面是通知和切入点的结合。现在发现了吧，没连接点什么事情，连接点就是为了让你好**理解**切点，搞出来的，明白这个**概念**就行了。通知说明了干什么和什么时候干（什么时候通过方法名中的 before,after，around 等就能知道），而切入点说明了在哪干（指定到底是哪个方法），这就是一个完整的切面定义。

5.引入（introduction）

允许我们向现有的类添加新方法属性。这不就是把切面（也就是新方法属性：通知定义的）用到目标类中吗

6.目标（target）

引入中所提到的目标类，也就是要被通知的对象，也就是真正的业务逻辑，他可以在毫不知情的情况下，被咱们织入切面。而自己专注于业务本身的逻辑。

7.代理(proxy)

怎么实现整套 aop 机制的，都是通过代理，这个一会给细说。

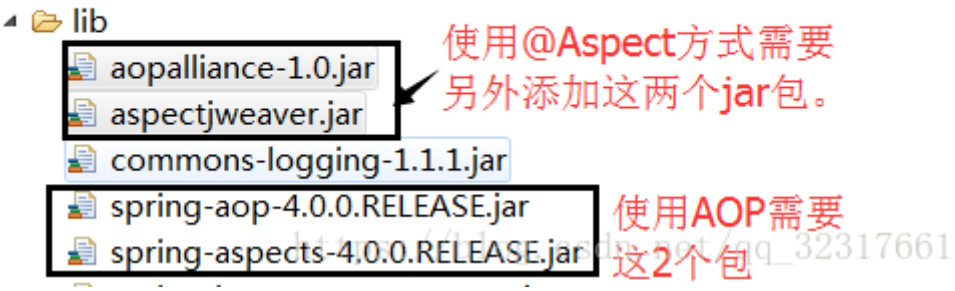
8.织入(weaving)

把切面应用到目标对象来创建新的代理对象的过程。有 3 种方式，spring 采用的是运行时，为什么是运行时，后面解释。

关键就是：切点定义了哪些连接点会得到通知

三、为了便于大家理解，我写了个简单的代码示例：

引入相关jar包



2、Spring的配置文件 applicationContext.xml 中引入context、aop对应的命名空间；配置自动扫描的包，同时使切面类中相关方法中的注解生效，需自动地为匹配到的方法所在的类生成代理对象。

```xml
<?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.o
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spri
        http://www.springframework.org/schema/context http://www.springframework.org/schema/cont

    <!-- 配置自动扫描的包 -->
    <context:component-scan base-package="com.qcc.beans.aop"></context:component-scan>
    <!-- 自动为切面方法中匹配的方法所在的类生成代理对象。 -->
    <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
</beans>
```


上面 jar 包引入成功，并且配置完毕以后，咱们随便创建一个接口和实现类（这个就跟平时创建 service 接口和实现类一样）

这里我就不展示代码了，比如 QueryService 接口和 QueryServiceImpl 实现类，里面有一个 queryList()查询方法！

接下来我们要创建具体的切面类：

1.先创建一个类，比如：MyAspect.java

2.在类上使用 @Aspect 注解 使之成为切面类

3.在类上使用 @Component 注解 把切面类加入到 IOC 容器中，或者在 spring 配置文件中创建 bean 也可以，也可以在它上面加@Service 注解，目的就是让它实例化

请看创建的切面类，类中的方法名随意取，关键是方法上面的注解，用@Aspect 注解方式来实现前置通知、返回通知、后置通知、异常通知、环绕通知！！！！！！！

```
import java.util.Arrays;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;
@Aspect
@Component
public class MyAspect {

    /**      * 前置通知：目标方法执行之前执行以下方法体的内容      *
```

```
@param jp       */
    @Before("execution(* com.qcc.beans.aop.*.*(..))")
    public void beforeMethod(JoinPoint jp){
        String methodName = jp.getSignature().getName();
        System.out.println("【前置通知】the method 【" + methodName +
"】 begins with " + Arrays.asList(jp.getArgs()));
        }


/**      * 返回通知：目标方法正常执行完毕时执行以下代码       * @param
jp       * @param result       */
    @AfterReturning(value="execution(*
com.qcc.beans.aop.*.*(..))",returning="result")
    public void afterReturningMethod(JoinPoint jp, Object result){
        String methodName = jp.getSignature().getName();
        System.out.println("【返回通知】the method 【" + methodName +
"】 ends with 【" + result + "】");
        }

    /**      * 后置通知：目标方法执行之后执行以下方法体的内容，不管是否发
生异常。       * @param jp       */
```

```
    @After("execution(* com.qcc.beans.aop.*.*(..))")
    public void afterMethod(JoinPoint jp){
        System.out.println("【后置通知】this is a afterMethod advice...");
        }


    /**      * 异常通知：目标方法发生异常的时候执行以下代码      */
    @AfterThrowing(value="execution(*
com.qcc.beans.aop.*.*(..))",throwing="e")
    public void afterThorwingMethod(JoinPoint jp, NullPointerException
e){
        String methodName = jp.getSignature().getName();
        System.out.println("【异常通知】the method 【" + methodName +
"】 occurs exception: " + e);


    }

}
```

上面加粗的几个注解大家结合注释都应该明白什么意思，这些注解决定了方法在什么时间点被执行；

注解中的 execution 用来表示这个切面类中的该方法在哪里执行，也就是作用的目标，看到这里的同学，应该能明白，其实 AOP 简单的来说就是这么回事！

注意：这里的 execution 中我作用的路径是某个实现类下面的全部方法，还可以具体作用到某一个方法，详情请看:https://www.cnblogs.com/rainy-shurun/p/5195439.html

看一下执行的结果：

```
【环绕通知中的--->前置通知】: the method 【getResultPage】 begins with [com.qding.insurance.param.CompensateRecordParam@7d5744b4
【前置通知】the method 【getResultPage】 begins with [com.qding.insurance.param.CompensateRecordParam@7d5744b4]
18:03:40.429 [http-bio-8080-exec-3] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Returning cached instance of singletc
【环绕通知中的--->前置通知】: the method 【getResultPage】 begins with [com.qding.insurance.param.CompensateRecordParam@7d5744b4
【前置通知】the method 【getResultPage】 begins with [com.qding.insurance.param.CompensateRecordParam@7d5744b4]
【环绕通知中的--->返回通知】: the method 【getResultPage】 ends with com.qding.framework.common.basemodel.ResultPage@227b9676
【环绕通知中的--->后置通知】: ------------------end.----------------------
【后置通知】this is a afterMethod advice...
【返回通知】the method 【getResultPage】 ends with 【com.qding.framework.common.basemodel.ResultPage@227b9676】
【环绕通知中的--->返回通知】: the method 【getResultPage】 ends with com.qding.framework.common.basemodel.ResultPage@227b9676
【环绕通知中的--->后置通知】: ------------------end.----------------------
【后置通知】this is a afterMethod advice...
【返回通知】the method 【getResultPage】 ends with 【com.qding.framework.common.basemodel.ResultPage@227b9676】
```

因为没发生异常，所以异常通知没有打印！AOP 原理的简单实现就这么简单，讲到这里再说一下 spring 事务的开启，提交或回滚，是不是也是 AOP 的前置-后置或者环绕的体现呢，哈哈！趁热打铁，咱们下一章讲一下关于 AOP 和 spring 事务的内容，会重点讲到 aop 代理机制、spring 事务如何开启和同一个 service 两个方法调用事务失效的问题！！！