Advice?Pointcut?Aspect?JoinPoint?Advisor?

阿全啊 发布于 2019-03-16

AOP概念？傻傻分不清？

AOP是一种面向切面的编程思想，最小单位是切面；Java是基于OOP的编程思想，最小单位是类。
虽然思想不同，但是AOP却能够给Java提供强大的加持，Spring AOP的应用就是最好的例子。
AOP思想是如此优秀，所以在一开始，就有人尝试建立组织来统一规范，这个组织就是AOP联盟。
~~AOP联盟为AOP的实现提出了多种方案，主要有：基于拦截的（Interceptor）、直接修改字节码的。(可忽略)~~
先来说说Spring中，AOP的几个主要概念：

```
切面、切入点、连接点、建议（增强）
```

开始晕了... 深吸一口气，慢慢往下看。
~~一般我们做事情，时间、地点、人物、做什么缺一不可，那么我们看看这些是如何对应的。~~
Spring AOP的设计是遵循AOP联盟的。AOP联盟中有两个主要概念：**Advice、JoinPoint**

```
Advice:增强，这个比较好理解，就是想要增加的功能，比如：上下包裹一个事务、日志打印等，我们把它对
JoinPoint:行话叫：程序'运行点'。什么意思呢？就是当代码运行到需要执行Advice时的那个地方的信息，
```

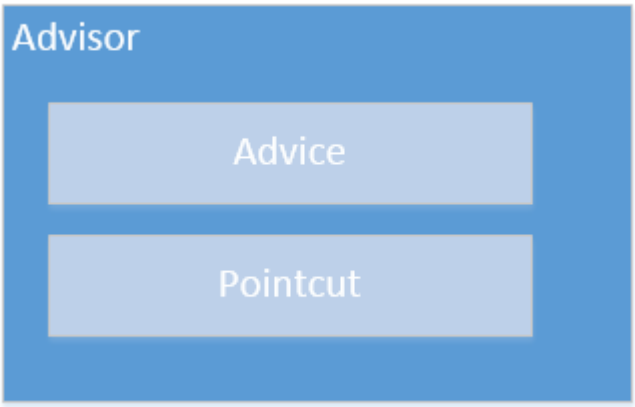
Spring引入了这两个概念，并对Advice作了增强实现：

```
@Before、@After、@Around等等：行话分别是前置增强、后置增强、环绕增强，
主要负责控制'Advice'是在方法前还是方法后等哪个范围执行，我们把它对应为 'When+Advice=> When +
```

但是单单这两个还是不够的，Spring另外提供了概念：

```
Pointcut:行话叫切入点，实际上是`Spring`抽象出来的，用来`查找哪些`需要进行增强的接口，我们把它
```

看起来好像理解了，但是有感觉还是有点混乱，好多概念，好乱，还是好乱...
这时候，Spring出手了，他也觉得有点乱，怎么办呢?
Advisor出现了，看看它的构造:



没错！Spring将Advice和Pointcut两个封装起来了，就叫Advisor。一个Advisor就能够确定什么时间地做什么，构造它需要：Advice + Pointcut。
看看在XML中一个Advisor的定义:

```xml
<aop:config>
    <aop:advisor
      pointcut="com.xyz.someapp.SystemArchitecture.businessService()"
      advice-ref="tx-advice"/>
</aop:config>

<tx:advice id="tx-advice">
    <tx:attributes>
        <tx:method name="*" propagation="REQUIRED"/>
    </tx:attributes>
</tx:advice>
```

了解了这些还不够，剩下一个概念Aspect。实际上，Aspect是一个虚的概念，它代表了一系列的Advisor，也就是说，一个Aspect其实等于:

他他他要在哪里（Pointcut）什么时候（@Before）对谁（JoinPoint）做什么（Advice）
她她她要在哪里（Pointcut）什么时候（@Before）对谁（JoinPoint）做什么（Advice）
它它它要在哪里（Pointcut）什么时候（@Before）对谁（JoinPoint）做什么（Advice）

看看一个Aspect的定义

```java
@Aspect
public class ProfilingAspect {

    @Around("methodsToBeProfiled()")
    public Object profile(ProceedingJoinPoint pjp) throws Throwable {
        StopWatch sw = new StopWatch(getClass().getSimpleName());
        try {
            sw.start(pjp.getSignature().getName());
            return pjp.proceed();
        } finally {
            sw.stop();
            System.out.println(sw.prettyPrint());
        }
    }
                                这就是一个Advisor=Advice+Pointcut

    @Pointcut("execution(public * foo..*.*(..))")
    public void methodsToBeProfiled(){}
}
```

完毕!

隐隐约约还有什么东西漏下了？没错！就是`JoinPoint`，前面说过，这个是运行时的对象，是对`Who`的抽象，代码：

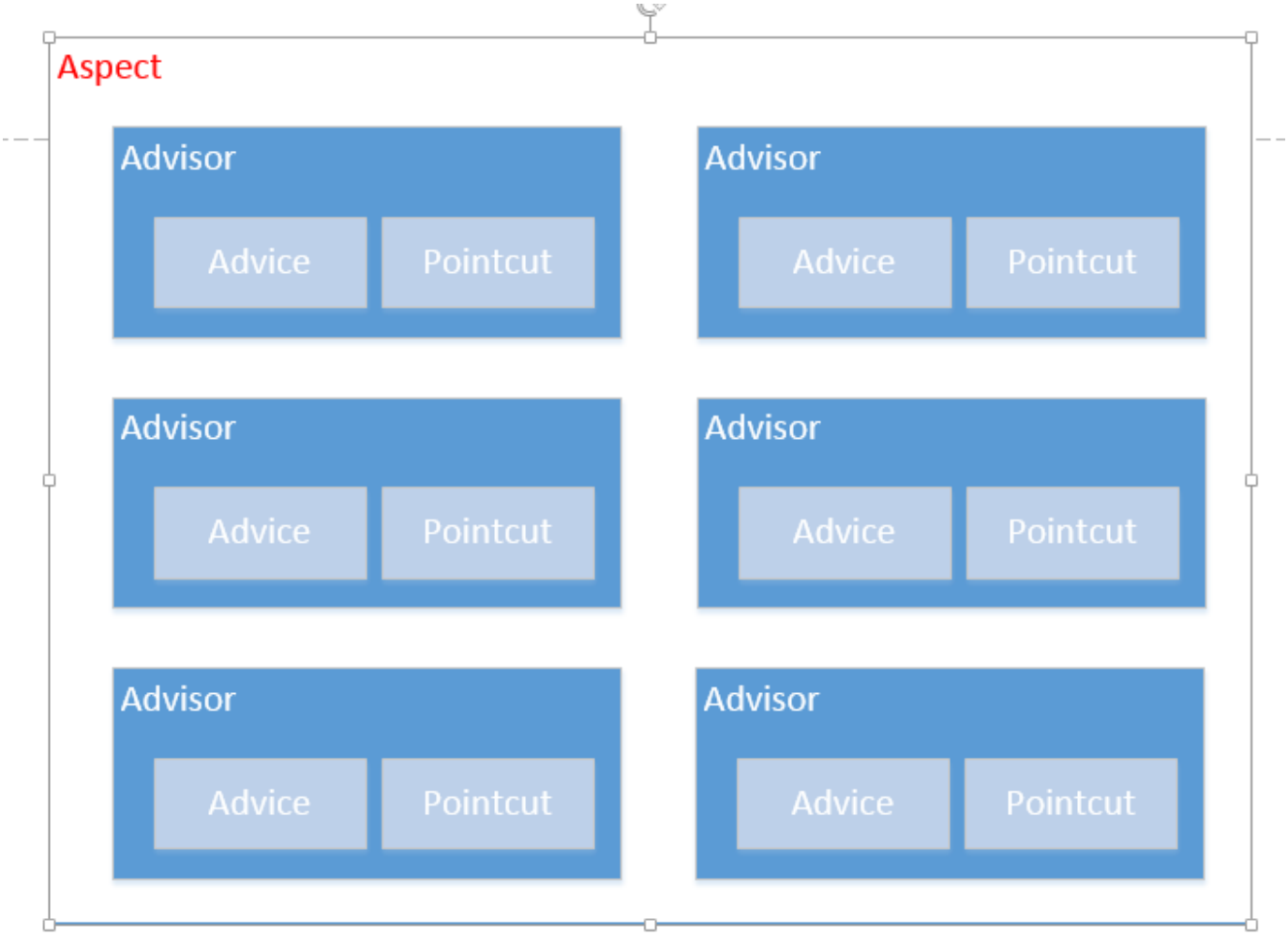
```java
@Aspect
public class ProfilingAspect {

    @Around("methodsToBeProfiled()")
    public Object profile(ProceedingJoinPoint pjp) throws Throwable {
        StopWatch sw = new StopWatch(getClass().getSimpleName());
        try {
            sw.start(pjp.getSignature().getName());
            return pjp.proceed();
        } finally {
            sw.stop();
            System.out.println(sw.prettyPrint());
        }
    }
                                这就是一个Advisor=Advice+Pointcut

    @Pointcut("execution(public * foo..*.*(..))")
    public void methodsToBeProfiled(){}
}
```

`Pointcut`对象，在程序执行到要增强的地方时，被封装好传入进来，要不要使用取决于自己（除非是`Around`），他代表了当时的上下文信息，就是一个在现场者。

我要定义一个切面(Aspect)，但是切面是虚的，只是定义用来代表一到多个Advisor，那我要先定义一个Adv

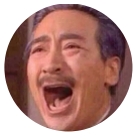
```
Aspect
    => n * Advisor
    => n * ( Advice + Pointcut) == n * (@Before/@After/@Around + Pointcut)
```

完毕!



java spring aop

阅读 2.6k • 更新于 2021-04-30



阿全啊

GC：很多对象年纪轻轻就会死

88 声望　**2** 粉丝

关注作者

3 条评论

得票 最新

撰写评论 ...

提交评论

评论支持部分 Markdown 语法：`**粗体** _斜体_ [链接](http://example.com) `代码` - 列表 > 引用`。你还可以使用 @ 来通知其他用户。

smilence：受教，看源码时，隐约感觉也是如此，看了这篇就清晰了

• 回复 • 2019-09-20

smilence：Aspect更像上下文对象(this)，Advice所指方法的this

• 回复 • 2019-09-20

阿全啊（作者）：@smilence 很高兴有所帮助，但我个人理解，其实 this 更应该是 JoinPoint，它代表了切面运行到那里时的上下文环境（包括对象），你可以从 JoinPoint 拿到执行时对象的信息。

• 回复 • 2019-11-24

继续阅读

Spring AOP IOC 之外 —— 类的命名

最近在阅读Spring的过程中，发现除了 AOP，IOC之外，还有许多被忽略的有助于阅读/理解源码的点，在此...

阿全啊　阅读 1.4k

Spring AOP(三) Advisor类架构

4　3