链滴

自定义注解式切面不生效？看完这篇你就明白了！

作者：yxw839841231

原文链接：https://ld246.com/article/1632461146826

来源网站：链滴

日常开发中，切面的使用已经非常频繁了，而注解的方式因为其极强的便利性，也非常受欢迎。

问题产生

突然有一天，小伙伴问我说，他写了个自定义注解的切面，然后实际使用效果不生效。

主要有两个问题，自定义的注解，作用目标是类和方法，但在方法上生效在类上不生效。翻译成代码下：

比如自定义一个注解Zjoin，其@Target已经指明了是METHOD和TYPE。

```
@Target({ElementType.METHOD, ElementType.TYPE})
public @interface Zjoin {
    String value() default "default";
}
```

同时定义切点为：

```
@Pointcut("@annotation(com.studying.aopdemo.business.Zjoin)")
public void methodPointCut() {

}
```

使用的时候当然就很简单了。

方法上加注解：

```
@PostMapping("/index")
@Zjoin("")
public Object index() {
    System.out.println("\n\n\n");
    return "teacher";
```

```
}
```

类上加注解：

```
@RestController
@RequestMapping("/api/school")
@Zjoin("")
public class SchoolController{
    @PostMapping("/index")
    public Object index(){
        System.out.println("\n\n\n");
        return "school";
    }
}
```

然后问题就来，注解加在方法上的时候，能成功进入切面，而加在类上的时候，并没有产生任何作用咋回事？

理解切面的本质

首先要明白一点，Spring只支持方法级别的连接点，Spring的AspectJ是通过动态代理来实现的，其用是对方法的增强，而不是对类的增强，所以最终拦截的一定是方法级别而不是类级别。

但是，纵观Sprig框架本身，非常多的注解是写在类上的，那肯定是说明注解加在类上是可以被拦截，为什么我写的就不生效呢？

因为你写错了！

如何正确自定义切面注解？

我们自定义注解的时候，大概最常见的是三种情况。

1、在某个类的特定方法上加上注解，进行特定拦截；

2、要对整个类的所有方法进行拦截；

3、我的类太多了，想在某个父类里加上注解，然后集成该类的子类都能被拦截。

第1种就不用多说了，不会的要打屁股了，先来说说第2种。

上文已经提到，AspectJ实际上是作用于方法的，那如果我要实现对整个类进行拦截怎么玩？其答案设置@PonitCut的声明中。

第1种情况，我们声明@Pointcut是通过 @annotation 来指定具体的某个注解。这种情况下，注解在类上是不会生效的。

如果你要想加在类上生效，那就不能用annotation而应该是within，那么代码正确的写法如下：

```
@Pointcut("@within(com.studying.aopdemo.business.Zjoin)")
public void classPointCut() {

}
```

注意，为了区分两种类型的声明方式，采用了不用的切点方法名称，切点是可以有多个的。

也就是说，当你想针对方法级别和类级别的切面都生效时，可以设置多个切面去进行拦截。

比如，当定义了上述两个切点是，可以做如下拦截：

```
@Around("methodPointCut() || classPointCut()")
public Object handleControllerMethod(ProceedingJoinPoint joinPoint) throws Throwable{

}
```

而对于第三种情况，实际上，只要将注解放在父类中，子类继承父类就已经能实现了。

父类：

```
@Zjoin("")
public class BaseController {
}
```

子类：

```
@RestController
@RequestMapping("/api/student")
public class StudentController extends BaseController{
    @PostMapping("/index")
    public Object index(){
        return "student";
    }
}
```

而如果全局的切面拦截，当我们的Controller数量太多的时候，可以采用指定包的方式来实现，代码下：

```
@Pointcut("execution(public * com.studying.aopdemo.business.web..*(..))")
public void classPointCut() {

}
```

实际上，所有的问题，集中于我们如何去设置@Pointcut，不同的声明方式能实现不同的效果。

@Pointcut参数详解

execution：用于匹配方法执行的连接点；

within：用于匹配指定类型内的方法执行；

this：用于匹配当前AOP代理对象类型的执行方法；注意是AOP代理对象的类型匹配，这样就可能包引入接口也类型匹配；

target：用于匹配当前目标对象类型的执行方法；注意是目标对象的类型匹配，这样就不包括引入接也类型匹配；

args：用于匹配当前执行的方法传入的参数为指定类型的执行方法；

@within：用于匹配所有持有指定注解类型内的方法；

@target：用于匹配当前目标对象类型的执行方法，其中目标对象持有指定的注解；

@args：用于匹配当前执行的方法传入的参数持有指定注解的执行；

@annotation：用于匹配当前执行方法持有指定注解的方法；

bean：Spring AOP扩展的，AspectJ没有对于指示符，用于匹配特定名称的Bean对象的执行方法；

reference pointcut：表示引用其他命名切入点，只有@ApectJ风格支持，Schema风格不支持。

示例

最后附上一个完整的切面处理逻辑，分别实现特定方法拦截、类级别拦截和继承拦截。

自定义注解和两种切点声明见上文，处理逻辑如下：

```
@Around("methodPointCut() || classPointCut()")
public Object handleControllerMethod(ProceedingJoinPoint joinPoint) throws Throwable {
    System.out.println("进入切面");

    Class<?> targetCls = joinPoint.getTarget().getClass();
    Zjoin zjoin = targetCls.getAnnotation(Zjoin.class);

    Signature signature = joinPoint.getSignature();
    MethodSignature ms = (MethodSignature) signature;
    Method targetMethod = targetCls.getDeclaredMethod(ms.getName(), ms.getParameterTy
es());
    Zjoin zjoin2 = targetMethod.getAnnotation(Zjoin.class);
    String operation = zjoin2 == null ? zjoin.value() : zjoin2.value();

    log.info("当前注解作用域为:{}", operation);
    return joinPoint.proceed();
}
```

特定方法拦截：

```
@RestController
@RequestMapping("/api/teacher")
public class TeacherController {
    @PostMapping("/index")
    @Zjoin("方法级别：TeacherController.index()")
    public Object index() {
        System.out.println("特定方法拦截\n\n\n");
        return "teacher";
    }
}
```

类级别拦截：

```
@RestController
@RequestMapping("/api/school")
@Zjoin("类级别：SchoolController")
public class SchoolController{
    @PostMapping("/index")
    public Object index(){
        System.out.println("类级别拦截\n\n\n");
        return "school";
```

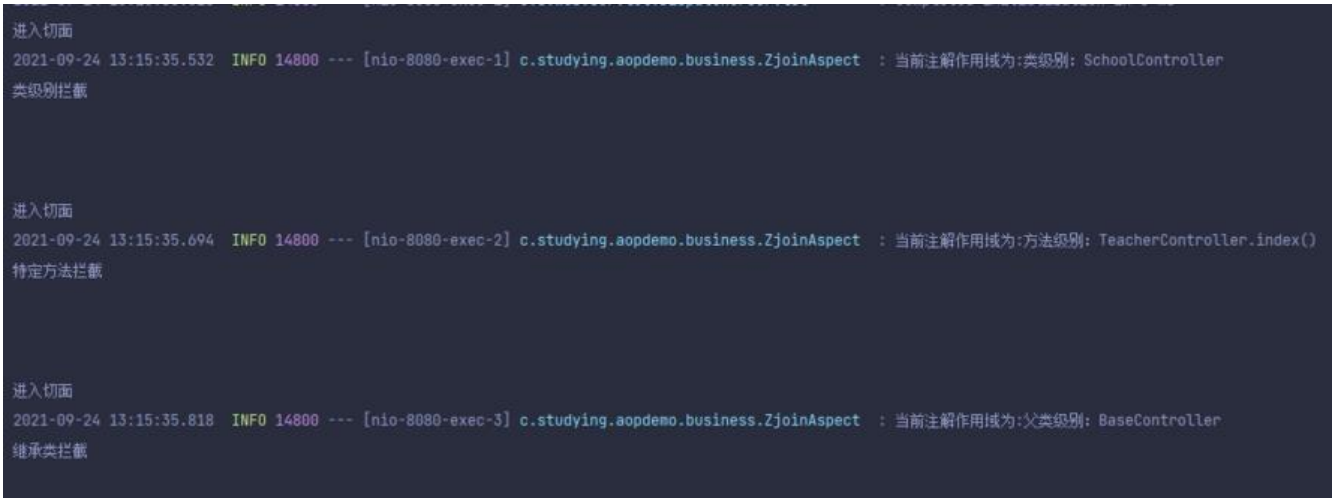
```
    }

}
```

继承类拦截：

```
@RestController
@RequestMapping("/api/student")
public class StudentController extends BaseController{
    @PostMapping("/index")
    public Object index(){
        System.out.println("继承类拦截\n\n\n");
        return "student";
    }
}
```

效果如下：



当然，如果想要更为直观的体现切面逻辑，可以再增加一个切面处理逻辑：

```
@Around("classPointCut()")
public Object handleController(ProceedingJoinPoint joinPoint) throws Throwable {
System.out.println("进入类级别切面");
Class<?> targetCls = joinPoint.getTarget().getClass();
Zjoin zjoin = targetCls.getAnnotation(Zjoin.class);
//1.2.4 获取自定义注解中operation属性的值
String operation = zjoin.value();
log.info("当前注解作用域为:{}", operation);
return joinPoint.proceed();
}
```

那么将看到如下输出效果

```
进入类级别切面
2021-09-24 13:22:09.201  INFO 23232 --- [nio-8080-exec-1] c.studying.aopdemo.business.ZjoinAspect  : 当前注解作用域为:类级别: SchoolController
进入切面
2021-09-24 13:22:09.203  INFO 23232 --- [nio-8080-exec-1] c.studying.aopdemo.business.ZjoinAspect  : 当前注解作用域为:类级别: SchoolController
类级别拦截



进入切面
2021-09-24 13:22:09.376  INFO 23232 --- [nio-8080-exec-2] c.studying.aopdemo.business.ZjoinAspect  : 当前注解作用域为:方法级别: TeacherController.index()
特定方法拦截



进入类级别切面
2021-09-24 13:22:09.651  INFO 23232 --- [nio-8080-exec-5] c.studying.aopdemo.business.ZjoinAspect  : 当前注解作用域为:父类级别: BaseController
进入切面
2021-09-24 13:22:09.652  INFO 23232 --- [nio-8080-exec-5] c.studying.aopdemo.business.ZjoinAspect  : 当前注解作用域为:父类级别: BaseController
继承类拦截
```

原文链接：自定义注解式切面不生效？看完这篇你就明白了！