前言：spring aop框架仅支持方法级别的连接点， 而aspcej支持字段，方法，构造器的连接点；

本代码用的时aspectJ 框架？貌似在切点在注解上时，注解在类上时，不会触发通知

代码：

Pom 引入依赖，即可使用

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-aop</artifactId>  
 <exclusions>  
 <exclusion>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-logging</artifactId>  
 </exclusion>  
 </exclusions>  
</dependency>

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by Fernflower decompiler)  
//  
  
package com.geekplus.test.common.datasource;  
  
import java.lang.reflect.Method;  
import org.aspectj.lang.JoinPoint;  
import org.aspectj.lang.annotation.After;  
import org.aspectj.lang.annotation.Aspect;  
import org.aspectj.lang.annotation.Before;  
import org.aspectj.lang.annotation.Pointcut;  
import org.aspectj.lang.reflect.MethodSignature;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.stereotype.Component;  
  
@Configuration("spring.aop.proxy-target-class=true ")  
@Component  
@Aspect  
public class DataSourceInterceptor {  
 public DataSourceInterceptor() {  
 }  
  
 public void setDataSourceArtemis(JoinPoint point) {  
 DatabaseContextHolder.setDataSource(DataSourceType.artemis);  
 }  
  
 public void setDataSourceGeektest(JoinPoint point) {  
 DatabaseContextHolder.setDataSource(DataSourceType.geektest);  
 }  
  
 @Pointcut("@annotation(com.geekplus.test.common.datasource.DataSource)")  
 public void dataSource() {  
 }  
  
 @Before("dataSource()")  
 public void intercept(JoinPoint point) throws Exception {  
 Class<?> target = point.getTarget().getClass();  
 MethodSignature signature = (MethodSignature)point.getSignature();  
 Class[] var4 = target.getInterfaces();  
 int var5 = var4.length;  
  
 for(int var6 = 0; var6 < var5; ++var6) {  
 Class<?> clazz = var4[var6];  
 this.resolveDataSource(clazz, signature.getMethod());  
 }  
  
 this.resolveDataSource(target, signature.getMethod());  
 }  
  
 private void resolveDataSource(Class<?> clazz, Method method) throws Exception {  
 Class<?>[] types = method.getParameterTypes();  
 if (clazz.isAnnotationPresent(DataSource.class)) {  
 DataSource source = (DataSource)clazz.getAnnotation(DataSource.class);  
 DatabaseContextHolder.setDataSource(source.dbtype());  
 }  
  
 Method annoMetheod = clazz.getMethod(method.getName(), types);  
 if (annoMetheod != null && annoMetheod.isAnnotationPresent(DataSource.class)) {  
 DataSource source = (DataSource)annoMetheod.getAnnotation(DataSource.class);  
 DatabaseContextHolder.setDataSource(source.dbtype());  
 }  
  
 }  
  
 @After("dataSource())")  
 public void afterSwitchDS(JoinPoint point) {  
 DatabaseContextHolder.clearDataSource();  
 }  
}

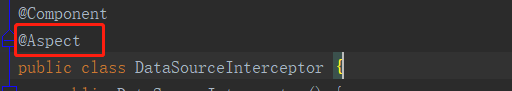
知识点：

切面是由通知，切点，连接点组成的

1. 表示切面普通类（不是某个接口的实现类）作为为可执行的切面时需要加上这句，否则抛出java.lang.ClassCastException: com.sun.proxy.$Proxy2 cannot be cast to com.Imp.userImp

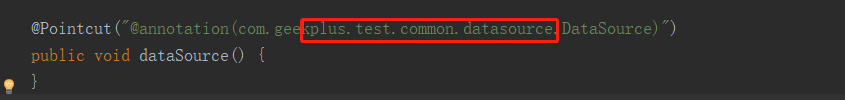


2.标识为切面



3.切点：决定“在哪做”决定在哪些“连接点”上应用通知

（就是要把动作加在哪个地方）这里加在全类名注解DataSource上，命名切点叫dataSource（），表示凡是代码中用到@dataSource注解的方法，都会在调用方法的时候触发通知，作出动作（前/后/环绕）



也可以这么写切点

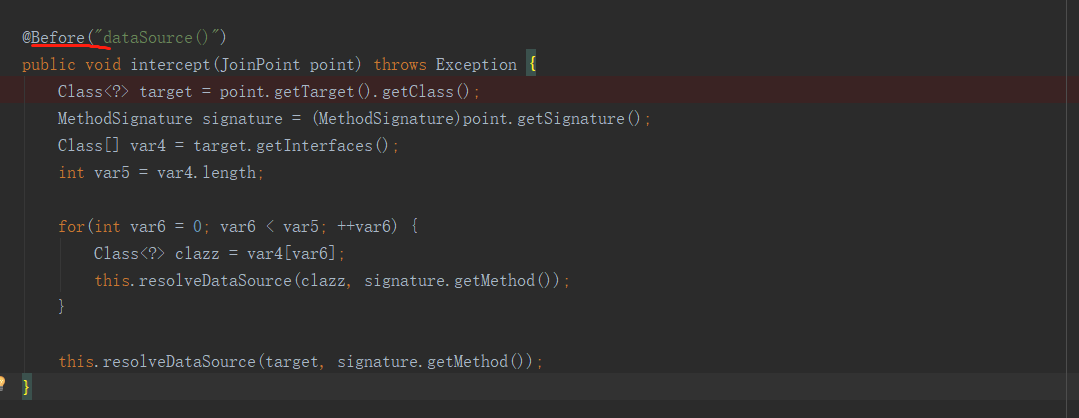
1. *\* 定义一个切入点表达式,用来确定哪些类需要代理*
2. *\* execution(\* aopdemo.\*.\*(..))代表aopdemo包下所有类的所有方法都会被代理*
3. *\*/*
4. @Pointcut("execution(\* aopdemo.\*.\*(..))")
5. public void declareJoinPointerExpression() {}

描述切点的几种方式

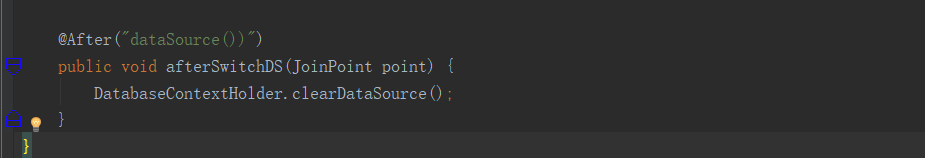
|  |  |
| --- | --- |
| **AspectJ**指示器 | 描　　述 |
| arg() | 限制连接点匹配参数为指定类型的执行方法 |
| @args() | 限制连接点匹配参数由指定注解标注的执行方法 |
| execution() | 用于匹配是连接点的执行方法 |
| this() | 限制连接点匹配AOP代理的bean引用为指定类型的类 |
| target | 限制连接点匹配目标对象为指定类型的类 |
| @target() | 限制连接点匹配特定的执行对象，这些对象对应的类要具有指定类型的注解 |
| within() | 限制连接点匹配指定的类型 |
| @within() | 限制连接点匹配指定注解所标注的类型（当使用Spring AOP时，方法定义在由指定的注解所标注的类里） |
| @annotation | 限定匹配带有指定注解的连接点 在类上注解时，类加载的时候不会执行通知，仅支持方法上的注解通知 |

4前置通知（切面在切点之前的要执行的代码 ）（ 传入JoinPoint连接点）

通知决定“做什么”“何时做”



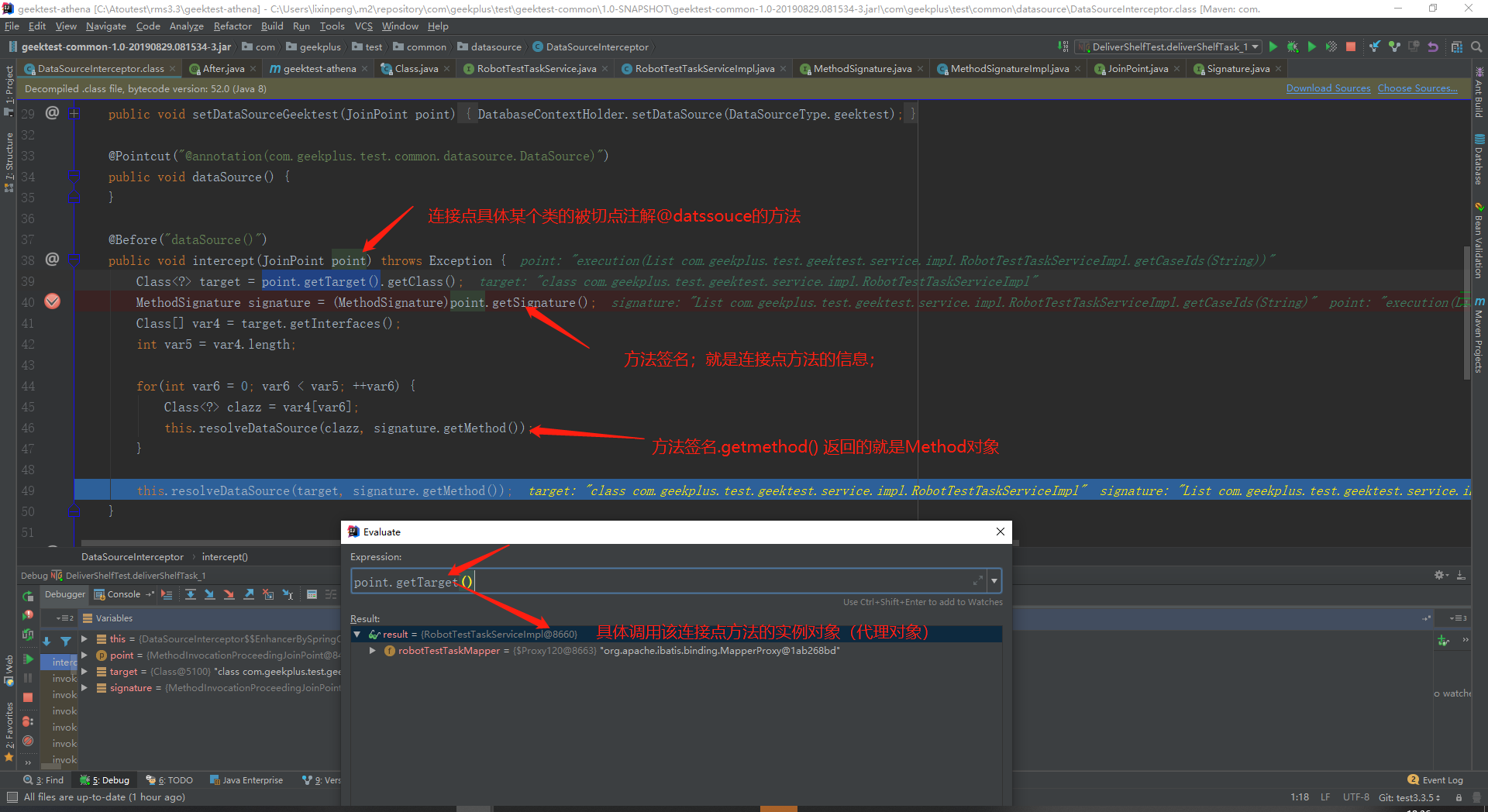
5后置通知（切面在切点之后的要执行的代码）



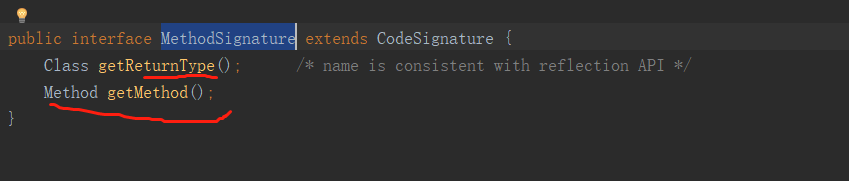
6.joinPoint“连接点”，所有能被通知的点，包括所有的类和所有的方法。 而具体要应用在哪些点，“切点“决定’

Joinpoint的属性方法

|  |  |
| --- | --- |
| **方法名** | **功能** |
| Signature getSignature(); | **获取封装了署名信息的对象,在该对象中可以获取到目标方法名,所属类的Class等信息** |
| Object[] getArgs(); | **获取传入目标方法的参数对象** |
| Object getTarget(); | **获取被代理的对象 本例中就是注解@datasouce 所在的类对象student** |
| Object getThis(); | **获取代理对象** |



其中返回的signature强制转化为MethodSignature对象，可以获取方法签名的方法对象menthod 和所属的类Class



7．代码分析，

此方法检查类或者方法是否被注解，如果注解，就拿出注解的值给别的程序用



引申自定义注解

