涉及到 java注解 和 aop 横切切面编程

**自定义注解**

1. 注解是一种元数据形式。即注解是属于java的一种数据类型，和类、接口、数组、枚举类似。
2. 注解用来修饰，类、方法、变量、参数、包。
3. 注解不会对所修饰的代码产生直接的影响。

**自定义注解的配置使用**

* **第一步，定义注解——相当于定义标记；**
* **第二步，配置注解——把标记打在需要用到的程序代码中；**
* **第三步，解析注解——在编译期或运行时检测到标记，并进行特殊操作。**

**注解其实就是一种标记，可以在程序代码中的关键节点（类、方法、变量、参数、包）上打上这些标记，然后程序在编译时或运行时可以检测到这些标记从而执行一些特殊操作**

注解的定义

注解通过 @interface 关键字进行定义。

public @interface TestAnnotation { }

元注解

元标签有 @Retention 翻译为持久力、保持力。即用来修饰自定义注解的生命力。

注解的生命周期有三个阶段：1、Java源文件阶段；2、编译到class文件阶 段；3、运行期阶段。

@Documented 是被用来指定自定义注解是否能随着被定义的java文件生成到JavaDoc文档当中

@Target 是专门用来限定某个自定义注解能够被应用在哪些Java元素上面的。

@Inherited 是指定某个自定义注解如果写在了父类的声明部分，那么子类的声明部分也能自动拥有该注解。@Inherited注解只对那些@Target被定义为ElementType.TYPE的自定义注解起作用。

**aop 横切切面编程**

aop：面向切面编程是软件编程思想发展到一定阶段的产物，是面向对象编程(OOP)的有益补充。AOP一般适用于具有横切逻辑的场合，如安全控制、事务管理、日志记录、性能统计等。面向切面编程简单地说就是在不改变源程序的基础上为代码段增加新的功能，对代码段进行增强处理。

aop的代理通过jdk动态代理，也可以通过cglib（谁个来波）实现，默认是通过jdk动态代理实现的。JDK动态代理需要接口的支持，如果没有接口只有类，则使用cglib实现。

@Before 前置通知，在方法执行之前执行

@After 后置通知，在方法执行之后执行（无论是否发生异常）还不能访问目标方法执行的结果

@AfterRunning 返回通知，在方法正常结束后 返回结果之后执行 可以访问方法的返回值

@AfterThrowing 异常通知，在方法抛出异常之后

@Around 环绕通知，围绕着方法执行

**Java反射**

1. *//第一种方式获取Class对象*
2. Student stu1 = new Student();*//这一new 产生一个Student对象，一个Class对象。*
3. Class stuClass = stu1.getClass();*//获取Class对象*

*2.//第二种方式获取Class对象*

1. Class stuClass2 = Student.class;
2. *//第三种方式获取Class对象*

Class stuClass3 = Class.forName("fanshe.Student");//注意此字符串必须是真实路径，就是带包名的类路径，包名.类名

1. *//1.加载Class对象*

Class class = Class.forName("类的全路径");

获取所有的构造方法(包括：私有、受保护、默认、公有)

conArray = class.getDeclaredConstructors();

获取所有公有构造方法

Constructor[] conArray = class.getConstructors();

获取公有、无参的构造方法

Constructor con = class.getConstructor(null);

Method[] conarray = class.getDeclaredMethods();//获取自身所有的方法(private、public、protected，和访问权限无关)，不包括继承的

**日志记录具体实现**

声明一个普通类 加上以下注解

@Component *//把类交给ioc容器管理*

@Aspect *//声明切面类*

通过aop的

*//前置通知 @before*

@Before(**"execution(\* org.fh.controller.\*.\*(..))"**)

方法参数中加入 JoinPoint

*//获取方法签名*

MethodSignature methodSignature = (MethodSignature)joinPoint.getSignature();

*//获取方法签名对象*

Method method = methodSignature.getMethod();

*//判断此方法是否需要记录日志*

*//判断方法上是否有日志注解*

logJilv annotation = method.getAnnotation(logJilv.**class**);

if(annotation != null){

}

有注解的话做日志记录