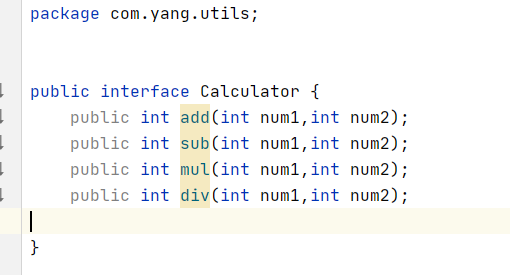
**Spring中面向切面编程AOP的底层实现使用的就是代理模式。这里记录我对代理模式的理解：**

比如现在有一个calculator计算机接口，功能有+、-、\*、/，其实现子类分别对其各项功能进行实现。

Calculator接口：

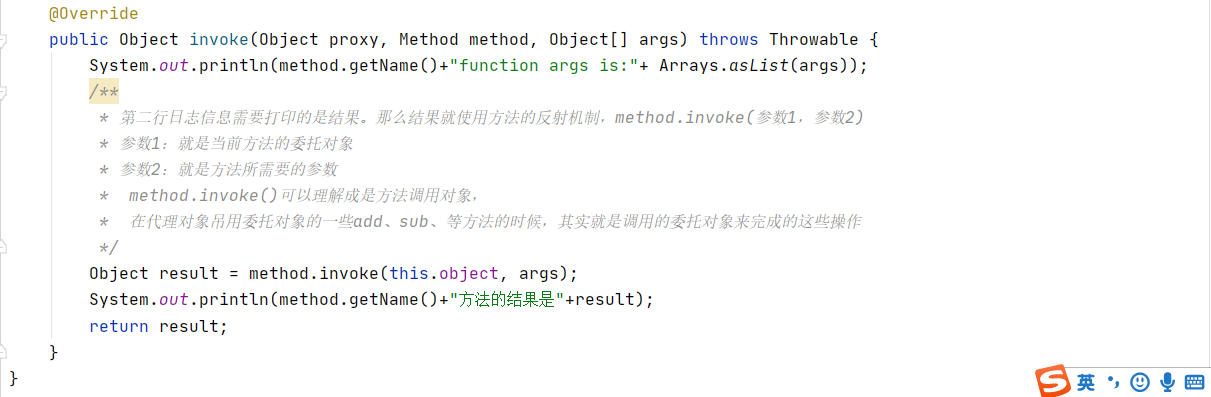


CalculatorImpl实现子类：



在其实现子类当中有打印出日志信息的功能，但是打印出日志信息的代码与逻辑处理代码完全耦合在一起，在后续的处理结果当中会非常的麻烦且不好维护，所以这时候就需要使用AOP面向切面编程将打印日志信息的代码抽离出来，与逻辑处理代码实现真正意义上的解耦合。

所谓代理模式，就是同业务逻辑代码相同，都是实现了某一个相同接口，拥有相同的方法。（例子：你想去买房子，但是现在各种各样的房子太多了，所以你需要一个中间商，他来按照你找房的规则来进行筛选房子，你只需要接受结果即可。那么他作为帮助你看房的中间商，他即也拥有这相同的看房功能。）只不过都需要手动来创建，现在在Spring框架中，就不需要手动来实现代理类了，框架会动态的（底层java反射机制）帮助你进行代理类的创建，而通过这种动态的代理模式完成的任务就是Spring的AOP面向切面编程。将日志信息与逻辑处理代码完全抽离出来。



具体的代码实现：首先这个MyInvocationHandler需要先实现InvocationHandler这个接口。然后实现invoke反射方法。在此之前，由于我们需要动态来建立代理类，所以需要先接受委托对象，将其作为成员变量object（不知道具体类型使用Object），然后写写一个bind绑定方法，在外部调用其的时候，就需要将委托类作为形参传入，然后返回的是一个代理对象，这里使用反射机制来返回代理对象。然后重写invoke方法，打印出本应存在与委托对象代码中原本的日志信息。

具体test测试：



主要是使用委托对象，然后利用MyInvocationHandler反射的代理对象来实现功能

这是底层的实现方法，在spring当中有更加简洁的手段来进行实现

创建LoggerAspact:

