###### AOP思想：横向重复，纵向抽取

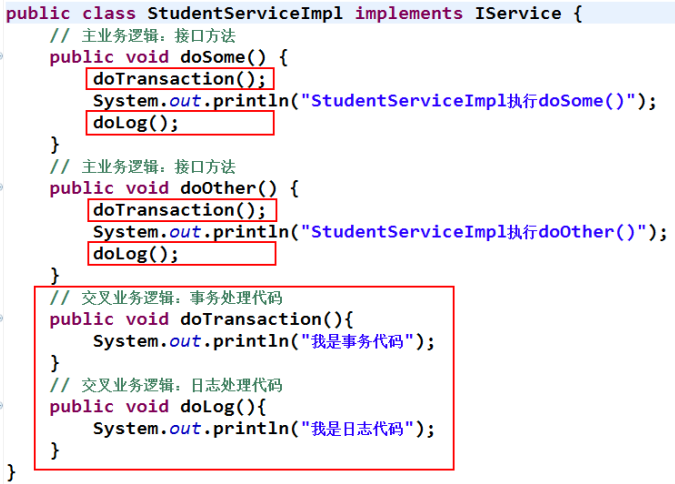
###### AOP的引入

**项目一：**

先定义好借口与一个实现类，该现实类中出了要实现接口的方法外，还要再写两个非业务方法。非业务方法也称为交叉业务逻辑：

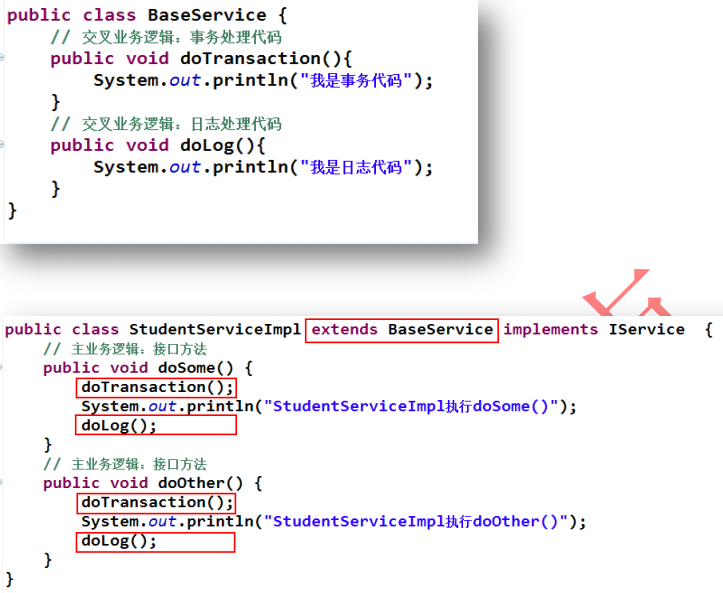
* doTransaction()：用于事物处理
* doLog()：用于日志处理

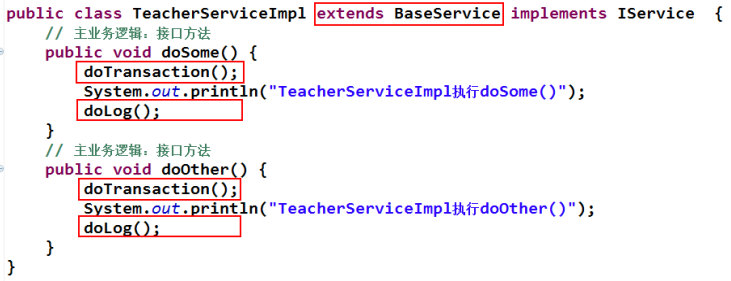
然后，再使接口方法调用它们。接口方法也称为主业务逻辑。



**项目二：**

此时提出一个问题：若有其它实现类同样也要调用这些事物、日志等处理方法怎么办？解决办法是，将这些日志、事物等方法包装到另外一个类中，让视线类继承这个类。

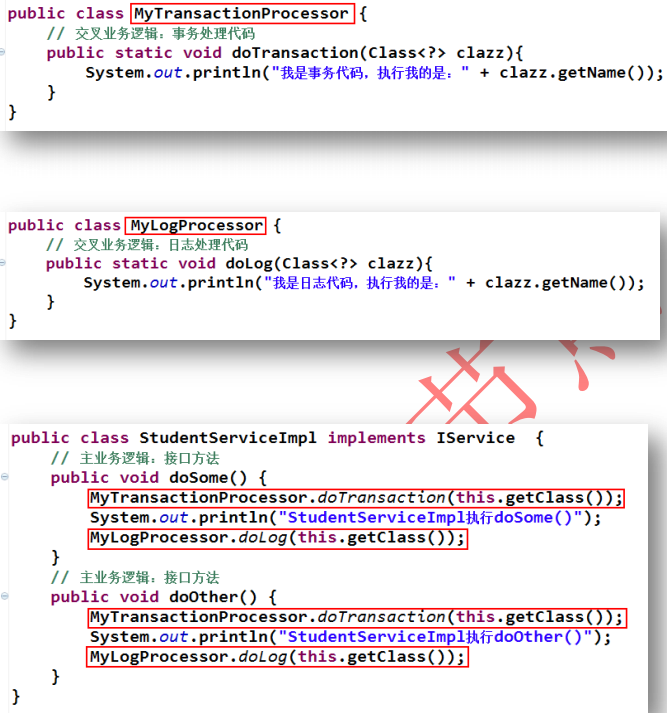




**项目三：**

以上的解决方案，也有一个弊端：若实现类还需要继承其他类，就不行了。因为Java不支持多继承。

当然，也可以有另一种解决方案：将这些业务交叉逻辑代码放到专门的工具类或处理类中，由主业务逻辑调用。



**项目四：**

以上的解决方案，还是存在弊端：交叉业务与主业务深度耦合在一起。当交叉业务逻辑较多时，在主业务代码中会出现大量的交叉业务逻辑代码调用语句，大大影响了主业务逻辑的可读性，降低了代码的可维护性，同时也增加了开发难度。

所以，可以采用动态代理方式。动态代理是OCP开发原则的一个重要体现：在不修改主业务逻辑的前提下，扩展和增强其功能。

Proxy.newProxyInstance(classLoader, Interface[] arr, InvocationHandler handler);

