* 设计模式 – 代理模式：

给某一个对象提供一个代理对象，并由代理对象控制对源对象的引用。代理就是一个人或一个机构代表另一个人或者一个机构采取行动。某些情况下，客户不想或者不能够直接引用一个对象，代理对象可以在客户和目标对象直接起到中介的作用。

在某些情况下，一个客户不想或者不能直接引用另一个对象，而代理对象可以在客户端 和目标对象之间起到中介作用。

代理模式的作用是：为其他对象提供一种代理以控制对这个对象的访问。

如图：客户端在调用实际的目标对象之前，先经过代理对象，代理对象实现了目标对象的所有接口，所以，通过对代理对象的调用，能把方法的调用委派给目标对象（在代理对象的方法内部，调用了目标对象的方法）。



代理模式分为： 静态代理 和 动态代理(利用反射实现)。

1. 静态代理。在企业开发中很少用到。

2、动态代理。使用动态代理，不需要再去为目标对象编写静态代理类了，我们只需要 通过JDK或者是第三方框架来动态的声称这个代理对象的字节码就可以了。

动态代理有两种：JDK动态代理和CGLIB动态代理。

* 代理模式所涉及到的角色：

抽象角色：提供一种真实角色和代理角色所共有的接口

真实角色：定义了代理角色要代理的真实对象

代理角色：含有对真实角色的引用，通常在客户端调用时代理对象都要在真实角色对象 的基础上增加一些真实角色所不能胜任的工作，而不只是简单的含有真实角色的引用

代理角色要和真实角色实现共同的接口。

代理类中要有真实角色的引用，因为真正去做事情的是真实角色。

代理角色可以实现附加的功能。

* 静态代理：

缺点：这种模式的一个代理类只能为一个接口的对象进行代理。

有了反射，我们通过运行时在变化，这时就可以出来动态代理了。编译的时候就写好了 固定的类，运行的时候给传不同的参数，它可以代替你不同的代理类。一个就可以代理 无数个了，这个思想非常重要。

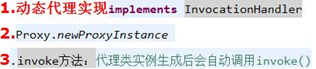
* 动态代理：

所谓动态代理，即通过代理类的代理，接口和实现类之间可以不直接发生联系，而可以在运行期（Runtime）实现动态关联.。

动态代理是在运行时，由java虚拟机生成的代理类。

Java在JDK 1.3之后加入协助开发动态代理功能的类，我們不必为特定对象与方法写特定的代理，使用动态代理，可以使得一个handler服务于各个对象。

JDK动态代理由java.lang.reflect.Proxy和java.lang.reflect.InvocationHandler实现。



如何使用 Java 动态代理。具体有如下四步骤：

1)通过实现 InvocationHandler 接口创建自己的调用处理器；

2)通过为Proxy类指定ClassLoader对象和一组 interface 来创建动态代理类；

3)通过反射机制获得动态代理类的构造函数，其唯一参数类型是调用处理器接口类型；

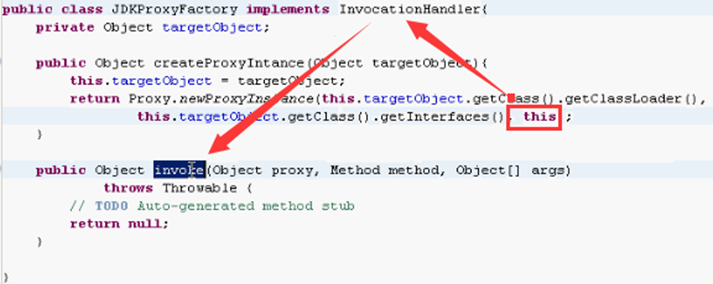
4)通过构造函数创建动态代理类实例，构造时调用处理器对象作为参数被传入。

* JDK动态代理：

使用JDK提供的代理类Proxy，它专门用于为目标对象动态创建代理对象。

使用java.lang.reflect.Proxy这个类的前提是，目标对象必须要实现一个接口Java.lang.

reflect.InvocationHandler，并实现它的invoke方法。invoke方法中有三个参数，第一个 是代理对象，第二个是拦截到的方法，第三个是方法的输入参数。



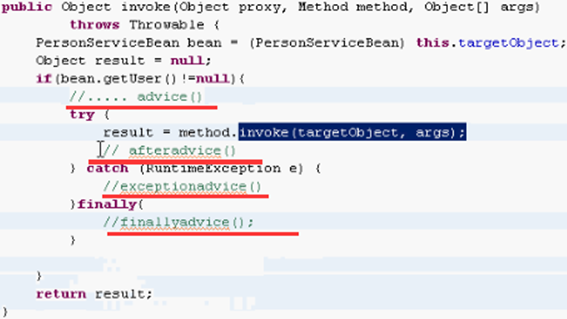
Proxy类的newProxyInstance方法，需要目标对象类加载路径、目标对象的所有接口和 InvocationHandler接口实现类对象三个参数，用于创建代理对象的实例。

当我们对代理对象的业务方法进行调用的时候，调用操作会被InvocationHandler接口的实现类对象拦截到，并执行它的invoke方法，就像上图中的this一样。

在这个invoke方法里面，如果我们要访问代理对象的话，就必须把这个方法的调用再 委派给目标对象。



以上，我们实现了创建代理对象，并对方法实现了拦截（进入invoke方法），可以在invoke方法中实现一些业务逻辑，比如AOP的实现，如下图：



JDK动态代理的原理：当客户端调用代理对象的时候，代理对象就会执行invoke方法， 在invoke方法里面，再把方法的调用委派给目标对象。

* CGLIB动态代理：

现实开发中，可能情况比较复杂，有可能目标类没有实现接口。这时候，我们可以用一 个第三方的创建代理对象的一个框架 – CGLIB。

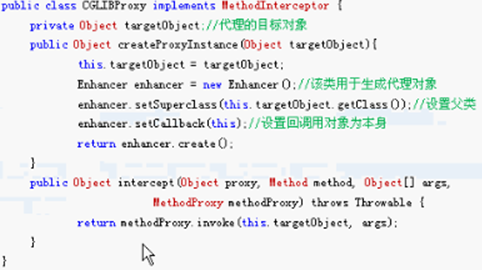


使用CGLIB创建代理对象，不要求你的目标类必须实现接口。

CGLIB创建代理类，实际上是继承了目标类，对目标类里面的所有非final修饰符的所有方法进行覆盖（重写），在原有的方法里面添加一些需要的业务代码。（创建出来的代理对象是目标类的子类）

与JDK的动态代理一样，当代理对象被调用，回调方法就会被执行。

回调方法中的参数：第一个是代理对象本身，第二个是回调方法，第三个是方法的参数， 第四个是方法的代理对象。

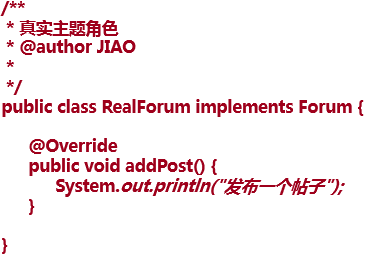


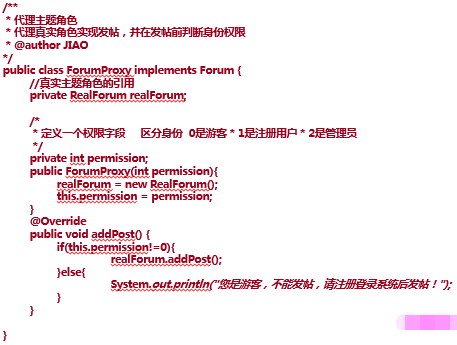
* JDK的动态代理和cglib动态代理的区别：

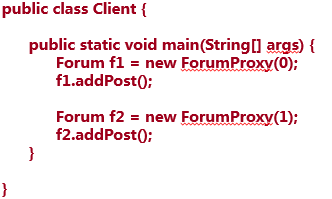
JDK动态代理需要接口的支持，如果没有接口只有类， 则使用cglib来实现。

* 静态代理Demo：









* JDK的动态代理Demo：

