之前为大家讲解过代理机制的操作，属于**静态代理**，**特征是代理类和目标对象的类都是在编译期间确定下来，不利于程序的扩展**。同时，每一个代理类只能为一个接口服务，这样一来程序开发中必然产生过多的代理。最好可以通过一个代理类完成全部的代理功能。

* **动态代理**
* 动态代理是指**客户通过代理类来调用其他对象的方法**，并且是在程序运行时根据需要动态创建目标类的代理对象。
* 使用场合：
* 调试；
* 远程方法调用。
* **代理设计模式原理**：

是用一个代理将对象包装起来，然后用该代理对象取代原始对象，任何对原始对象的调用都要通过代理。代理对象决定是否以及何时将方法调用转到原始对象上。

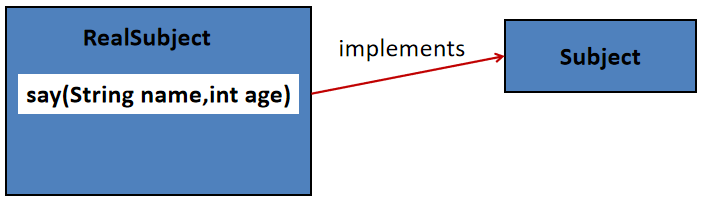
* **Proxy**：专们完成代理的操作类，是所有动态代理类的父类。通过此类为一个或多个接口动态地生成实现类。
* 提供用于创建动态代理类和动态代理对象的静态方法
* static Class<?> getProxyClass(ClassLoader loader,Class<?>...interfaces)创建一个动态代理类所对应的Class对象。
* static Object newProxyInstance(ClassLoader loader,

Class<?>[]interfaces,InvocationHandler h)

直接创建一个动态代理对象。（3个参数分别是类加载器、得到全部的接口、得到InvocationHandler接口的子类实例）

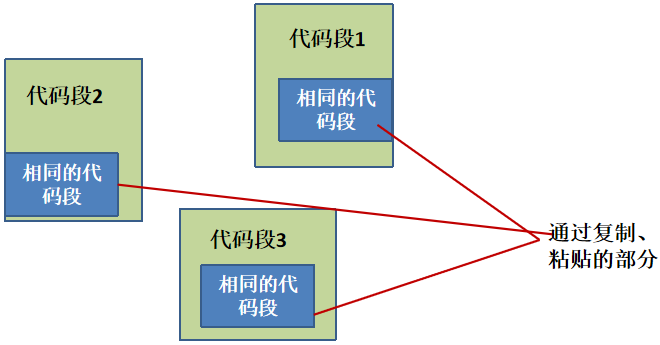
* **动态代理步骤**

1. **创建一个实现接口InvocationHandler的类**，它必须实**现invoke方法**，以完成代理的具体操作。
2. 创建被代理的类以及接口。



1. 通过Proxy的静态方法newProxyInstance(ClassLoader loader, Class[] interfaces, InvocationHandler h)创建一个Subject接口代理。
2. 通过Subject代理调用RealSubject实现类的方法。

* 动态代理与AOP（Aspect Orient Programming）
* 前面介绍的Proxy和InvocationHandler，很难看出这种动态代理的优势，下面介绍一种更实用的动态代理机制。



* 改进后的说明：代码段1、代码段2、代码段3和深色代码段分离开了，但代码段1、2、3又和一个特定的方法A耦合了！最理想的效果是：代码块1、2、3既可以执行方法A，又无须在程序中以硬编码的方式直接调用深色代码的方法。
* 使用Proxy生成一个动态代理时，往往并不会凭空产生一个动态代理，这样没有太大的意义。通常都是为指定的目标对象生成动态代理
* 这种动态代理在AOP中被称为AOP代理，AOP代理可代替目标对象，AOP代理包含了目标对象的全部方法。但AOP代理中的方法与目标对象的方法存在差异：AOP代理里的方法可以在执行目标方法之前、之后插入一些通用处理

