# **[Java设计模式之代理模式（Proxy）](http://www.cnblogs.com/whoislcj/p/5693980.html)**

### **前言：**

      最近在研究Retrofit开源框架的时候，其主要核心代码是通过注解标示参数，动态代理模式实现具体接口，反射机制进行参数解析，最终实现发送请求。其实之前在学习Xutils源码的时候，Xutils 的task也是通过代理模式来访问的。为何要采用代理模式呢？有什么好处呢？抱着这些疑问！今天来学习总结一下。

### **什么是代理模式？**

       代理模式的定义：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原对象的引用。举例说明，就是一个人或者一个机构代表另一个人或者另一个机构采取行动。在一些情况下，一个客户不想或者不能够直接引用一个对象，而代理对象可以在客户端和目标对象之前起到中介的作用。

### **应用场景举例：**

       通过上面的代理模式描述我们可以知道，其目的就是为了控制对象引用，生活场景中我们以买车为例，如果我们要买一辆轿车必须通过汽车4S店，汽车4s店就是充当代理角色，其目的就是控制买车客户的买车行为，必须通过汽车4S店才能从汽车厂商买一辆车。

#### **1.）首先新建一个买车的接口**

public interface IBuyCar {

//买车

void buyCar();

}

#### **2.）声明一个要买车的客户，实现买车接口**

[IMG_256](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

public class Customer implements IBuyCar {

private int cash;//购车款

public int getCash() {

return cash;

}

public void setCash(int cash) {

this.cash = cash;

}

@Override

public void buyCar() {

Log.e("buyCar", "买一辆车花费了-->" + cash + "元");

}

}

[IMG_257](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

#### **3.）声明一个买车代理汽车4S店，同样也实现买车接口，必须接受客户下单**

[IMG_258](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

public class BuyCarProxy implements IBuyCar{

private Customer customer;//接收买车客户

public BuyCarProxy(Customer customer){

this.customer=customer;//接收买车客户 }

@Override

public void buyCar() {//实现为客户买车

customer.buyCar();

}

}

[IMG_259](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

#### **4.)创建一个客户端，模拟一次买车**

Customer customer=new Customer();

customer.setCash(120000);

BuyCarProxy buyCarProxy=new BuyCarProxy(customer);

buyCarProxy.buyCar();

IMG_260

#### **5.）通过代理模式实现权限控制**

  通过上面的例子，我们可能有个疑问，难道就不能直接去厂家买车吗？当然可以，如果在使用场景中实现类能满足要求时，我们当然可以直接实现类，但当实现类不能满足要求，要扩展需求，根据开闭原则你又不能修改实现类代码，这时你就用代理类。比如购买一辆车我们要对客户进行一个购车款审核，如果符合条件就买车，不符合要求我们就告知客户购车款不足。

[IMG_261](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

@Override

public void buyCar() {//实现为客户买车

int cash=customer.getCash();

if(cash<100000){

Log.e("buyCar","你的钱不够买一辆车");

return;

}

customer.buyCar();

}

[IMG_262](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

实现场景

[IMG_263](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

Customer customer=new Customer();

customer.setCash(120000);

BuyCarProxy buyCarProxy=new BuyCarProxy(customer);

buyCarProxy.buyCar();

Customer customer1 =new Customer();

customer1.setCash(90000);

BuyCarProxy buyCarProxy1 =new BuyCarProxy(customer1);

buyCarProxy1.buyCar();

[IMG_264](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

IMG_265

### **动态代理机制：**

以上讲的都是代理模式的静态实现，所谓静态代理就是自己要为要代理的类写一个代理类，或者用工具为其生成的代理类，总之，就是程序运行前就已经存在的编译好的代理类，这样有时候会觉得非常麻烦，也导致非常的不灵活，相比静态代理，动态代理具有更强的灵活性，因为它不用在我们设计实现的时候就指定某一个代理类来代理哪一个被代理对象，我们可以把这种指定延迟到程序运行时由JVM来实现。

举例：还是接着上面的例子

#### **1.）首先我们要声明一个动态代理类，实现InvocationHandler接口**

[IMG_266](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

public class DynamicProxy implements InvocationHandler {

// 被代理类的实例 Object obj;

// 将被代理者的实例传进动态代理类的构造函数中

public DynamicProxy(Object obj) {

this.obj = obj;

}

/\*\*

\* 覆盖InvocationHandler接口中的invoke()方法

\* 更重要的是，动态代理模式可以使得我们在不改变原来已有的代码结构

\* 的情况下，对原来的“真实方法”进行扩展、增强其功能，并且可以达到

\* 控制被代理对象的行为，下面的before、after就是我们可以进行特殊

\* 代码切入的扩展点了。

\*/

public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable {

/\*

\* before ：doSomething();

\*/

Object result = method.invoke(this.obj, args);

/\*

\* after : doSomething();

\*/

return result;

}

}

[IMG_267](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

#### **2.）具体实现**

[IMG_268](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

//我们要代理的真实对象

Customer customer = new Customer();

//我们要代理哪个真实对象，就将该对象传进去，最后是通过该真实对象来调用其方法的

InvocationHandler handler = new DynamicProxy(customer);

/\*

\* 通过Proxy的newProxyInstance方法来创建我们的代理对象，我们来看看其三个参数

\* 第一个参数 handler.getClass().getClassLoader() ，我们这里使用handler这个类的ClassLoader对象来加载我们的代理对象

\* 第二个参数customer.getClass().getInterfaces()，我们这里为代理对象提供的接口是真实对象所实行的接口，表示我要代理的是该真实对象，这样我就能调用这组接口中的方法了

\* 第三个参数handler， 我们这里将这个代理对象关联到了上方的 InvocationHandler 这个对象上

\*/

IBuyCar buyCar = (IBuyCar) Proxy.newProxyInstance(handler.getClass().getClassLoader(), customer.getClass().getInterfaces(), handler);

buyCar.buyCar();

[IMG_269](http://www.cnblogs.com/whoislcj/p/javascript:void(0);)

#### **3.）动态代理好处**

使用Java动态代理机制的好处：

1、减少编程的工作量：假如需要实现多种代理处理逻辑，只要写多个代理处理器就可以了，无需每种方式都写一个代理类。

2、系统扩展性和维护性增强，程序修改起来也方便多了(一般只要改代理处理器类就行了)。

### **总结：**

   通过上面的应用例子我们学习了代理模式的具体使用场景。