设计模式 --- 装饰模型 (转)

作者：ironMan
原文链接：https://ld246.com/article/1605714211495
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

前言

Step y Step

前一篇已经讲解了代理模式了，今天要讲解的就是**装饰模式**啦~

在看到FilterInputStream和FilterOutputStream时看到了之前常听见的**装饰模式**(对IO一定了解的同可能都会知道那么一句话：在IO用得最多的就是装饰模式了)！

其实无论是代理模式还是装饰模式。本质上我认为就是**对原有对象增强的方式**~

那么接下来就开始吧，如果文章有错误的地方请大家多多包涵，不吝在评论区指正哦~

声明：本文使用JDK1.8

一、对象增强的常用方式

很多时候我们可能**对Java提供给我们的对象不满意**，不能满足我们的功能。此时我们就想对Java原对进行增强，能够实现我们想要的功能就好~

一般来说，实现对象增强有三种方式：

- **继承**
  - 继承父类，子类扩展
- **装饰器模式**
  - 使用“包装”的方式来增强对象
- **代理模式**

1.1继承

最简单的方式就是继承父类，子类扩展来达到目的。虽然简单，但是这种方式的**缺陷非常大**：

- 一、如果 **父类是带有数据、信息、属性的话，那么子类无法增强。**
- 二、子类实现了之后 **需求无法变更**，增强的内容是**固定**的。

1.1.1第一点

第一点就拿以前在学JDBC的时候来说：

- 当时想要自己写一个简易的JDBC连接池，连接池由 List<Connection>来管理。显然我们的对象Connection，当写到close()方法的时候卡住了。
- 因为我们想要的功能是：调用 close()是让我们的Connection返回到“连接池”(集合)中，而不是闭掉。
- 此时我们 **不能**使用继承父类的方式来实现增强。因为Connection对象是由数据库厂商来实现的在得到Connection对象的时候绑定了各种信息(数据库的username、password、具体的数据库是啥等)。我们**子类继承Connection是无法得到对应的数据的**！就更别说调用close()方法了。

1.1.2第二点

第二点我也举个例子：

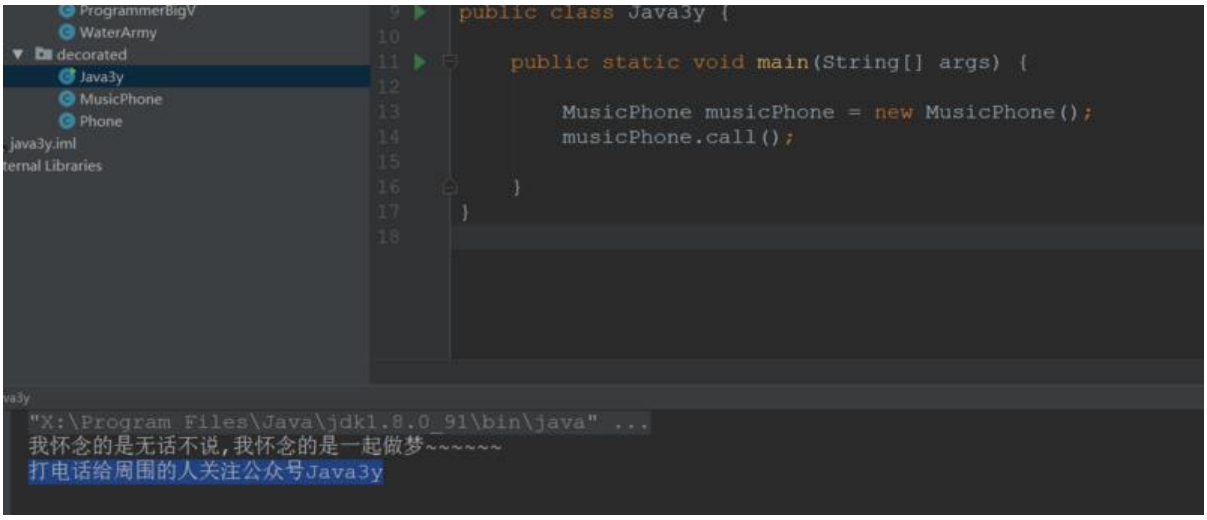
现在我设计一个电话类：

```
public class Phone {
    // 可以打电话
    public void call() {
        System.out.println("打电话给周围的人关注公众号Java3y");
    }
}
```

此时，我想**打电话之前能听彩铃**，于是我继承Phone类，实现我想要的功能。

```
public class MusicPhone extends Phone {

    // 听彩铃
    public void listenMusic() {
        System.out.println("我怀念的是无话不说,我怀念的是一起做梦~~~~~~");
    }

    @Override
    public void call() {

        // 在打电话之前听彩铃
        listenMusic();

        super.call();
    }
}
```

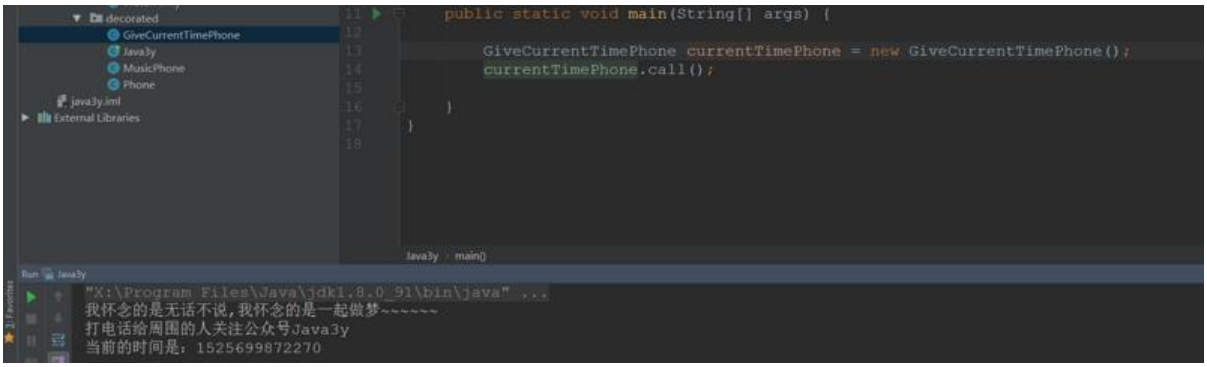
我们的功能就做好了：



● 此时，我又突然想实现多一个需求了，我想要听完电话之后告诉我一下当前的时间是多少。没事，们**又继承来增强**一下：

```
// 这里继承的是MusicPhone类
public class GiveCurrentTimePhone extends MusicPhone {

    // 给出当前的时间
    public void currentTime() {
        System.out.println("当前的时间是：" + System.currentTimeMillis());
    }

    @Override
    public void call() {
        super.call();

        // 打完电话提示现在的时间是多少啦
        currentTime();
    }
}
```

所以我们还是可以完成任务滴：



可是我需求现在**又想变**了：

● 我不想听彩铃了，只想听完电话通知一下时间就好了.....(可是我们的通知时间电话类是继承在听彩的电话类基础之上的),,,

● 我又有可能：我想在听电话之前报告一下时间，听完电话听音乐！...

● 如果 **需求变动很大**的情况下，而我们又用**继承的方式来实现**这样会导致一种现象：**类爆炸**(类数量增)！并且**继承的层次可能会比较多**~

所以，我们可以看到子类继承父类这种方式来扩展是**十分局限的，不灵活的**~

因此我们就有了**装饰模式**！

1.2装饰模式

首先我们来看看装饰模式是怎么用的吧。

1.2.1前提代码

电话接口：

```
// 一个良好的设计是抽取成接口或者抽象类的
public interface Phone {

    // 可以打电话
    void call();
}
```

具体的实现：

```
public class IphoneX implements Phone {


    @Override
    public void call() {
        System.out.println("打电话给周围的人关注公众号Java3y");
    }
}
```

1.2.2包装模式实现

上面我们已经**拥有了一个接口还有一个默认实现。**包装模式是这样干的：

首先我们弄一个**装饰器**，它实现了接口，以**组合**的方式接收我们的**默认实现类。**

```
// 装饰器，实现接口
public abstract class PhoneDecorate implements Phone {

    // 以组合的方式来获取默认实现类
    private Phone phone;
    public PhoneDecorate(Phone phone) {
        this.phone = phone;
    }

    @Override
    public void call() {
        phone.call();
```

```
    }
}
```

有了装饰器以后，我们的扩展都可以**以装饰器为基础进行扩展**，继承装饰器来扩展就好了！

我们想要在**打电话之前听音乐**：

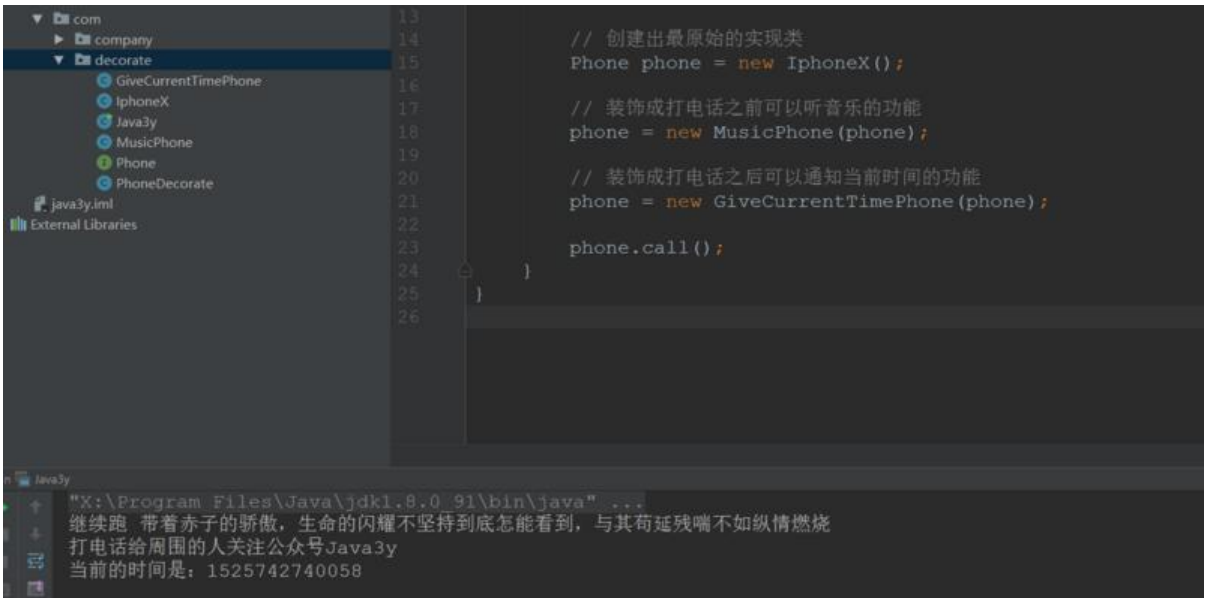
```
// 继承着装饰器来扩展
public class MusicPhone extends PhoneDecorate {

    public MusicPhone(Phone phone) {
        super(phone);
    }

    // 定义想要扩展的功能
    public void listenMusic() {

        System.out.println("继续跑 带着赤子的骄傲，生命的闪耀不坚持到底怎能看到，与其苟延残喘
如纵情燃烧");

    }

    // 重写打电话的方法
    @Override
    public void call() {

        // 在打电话之前听音乐
        listenMusic();
        super.call();
    }
}
```

现在我也想在**打完电话后通知当前的时间**，于是我们也**继承装饰类来扩展**：

```
// 这里继承的是MusicPhone装饰器类
public class GiveCurrentTimePhone extends PhoneDecorate  {


    public GiveCurrentTimePhone(Phone phone) {
        super(phone);
    }
```

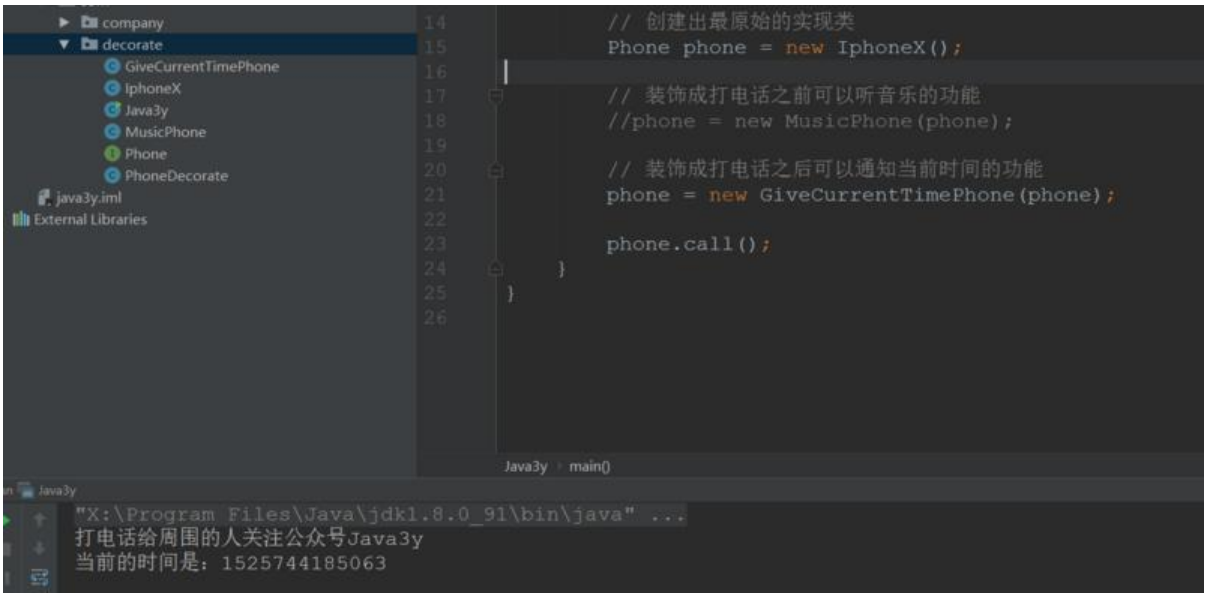
```
    // 自定义想要实现的功能：给出当前的时间
    public void currentTime() {
        System.out.println("当前的时间是：" + System.currentTimeMillis());
    }

    // 重写要增强的方法
    @Override
    public void call() {
        super.call();
        // 打完电话后通知一下当前时间
        currentTime();
    }
}
```

可以完成任务：



就**目前**这样看起来，比我直接继承父类要麻烦，而功能效果是一样的....我们继续往下看~~

此时，我不想在打电话之前听到彩铃了，很简单：我们**不装饰**它就好了！



此时，我想在打电话前报告一下时间，在打完电话之后听彩铃。

- 注意：虽然说要改动类中的代码， **但是**这种改动是合理的。因为我定义出的GiveCurrentTimePhone类和MusicPhone类本身从语义上就**没有规定**扩展功能的执行顺序
- 而继承不一样：先继承Phone->实现MusicPhone->再继承MusicPhone实现GiveCurrentTimePhone。这是**固定**的，从继承的逻辑上已经**写死了**具体的代码，是**难以改变**的。

```java
// 这里继承的是MusicPhone装饰器类
public class GiveCurrentTimePhone extends PhoneDecorate  {

    public GiveCurrentTimePhone(Phone phone) {
        super(phone);
    }

    // 自定义想要实现的功能：给出当前的时间
    public void currentTime() {
        System.out.println("当前的时间是：" + System.currentTimeMillis());
    }

    // 重写要增强的方法
    @Override
    public void call() {

        currentTime();
        super.call();
        // 打完电话后通知一下当前时间
        //currentTime();
    }
}
```

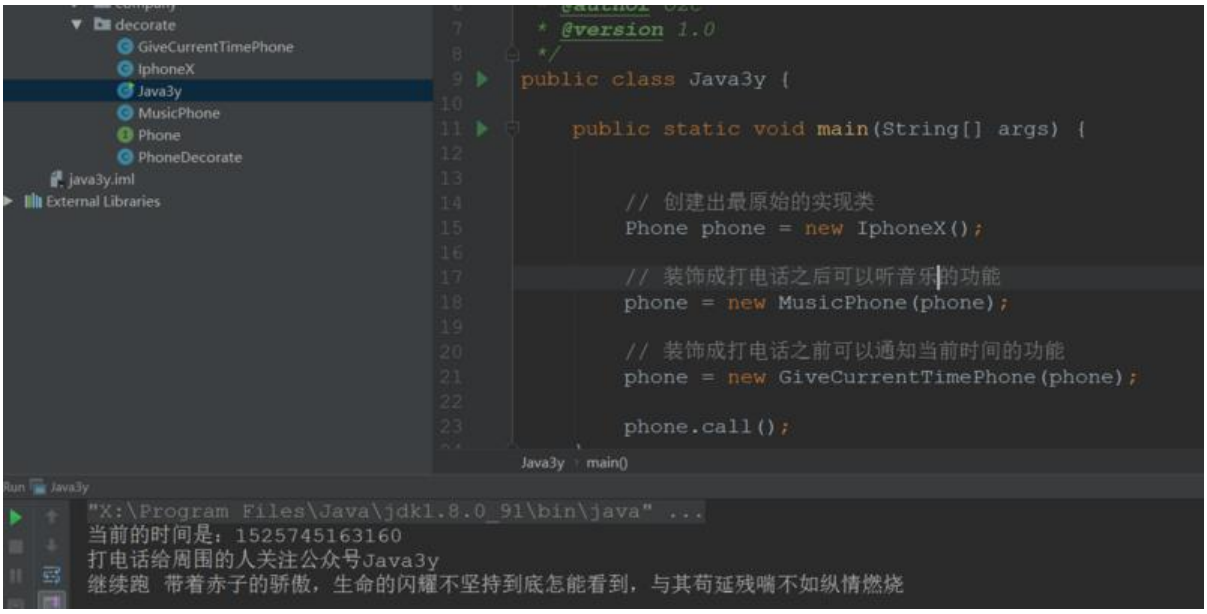
换到前面去

```java
// 继承着装饰器来扩展
public class MusicPhone extends PhoneDecorate {

    public MusicPhone(Phone phone) {
        super(phone);
    }

    // 定义想要扩展的功能
    public void listenMusic() {

        System.out.println("继续跑 带着赤子的骄傲，生命的闪耀不坚持到底
燃烧");

    }

    // 重写打电话的方法
    @Override
    public void call() {

        // 在打电话之前听音乐
        //listenMusic();
        super.call();

        listenMusic();
    }
}
```

换到后面去

原文链接：设计模式 --- 装饰模型 (转)

所以我们还是可以很简单地完成功能：



二、装饰模式讲解

可能有的同学在看完上面的代码之后，还是迷迷糊糊地不知道装饰模式是怎么实现“装饰”的。下面就再来解析一下：

- 第一步：我们有一个Phone接口，该接口定义了Phone的功能
- 第二步：我们有一个最简单的实现类iPhoneX
- 第三步：写一个装饰器抽象类PhoneDecorate，以 **组合**(构造函数传递)的方式接收我们最简单的现类iPhoneX。其实装饰器抽象类的作用就是**代理**(核心的功能还是由最简单的实现类iPhoneX来做，不过在**扩展**的时候可以**添加一些没有的功能而已**)。
- 第四步：想要扩展什么功能，就继承PhoneDecorate装饰器抽象类，将想要增强的对象(最简单的现类iPhoneX或者已经被增强过的对象)传进去，完成我们的扩展！

再来看看下面的图，就懂了！

往往我们的代码可以省略起来，成了这个样子(**是不是和IO的非常像**！)

```
// 先增强听音乐的功能，再增强通知时间的功能
    Phone phone = new GiveCurrentTimePhone(new MusicPhone(new IphoneX()));
```

结果是一样的：

2.1装饰模式的优缺点

优点：

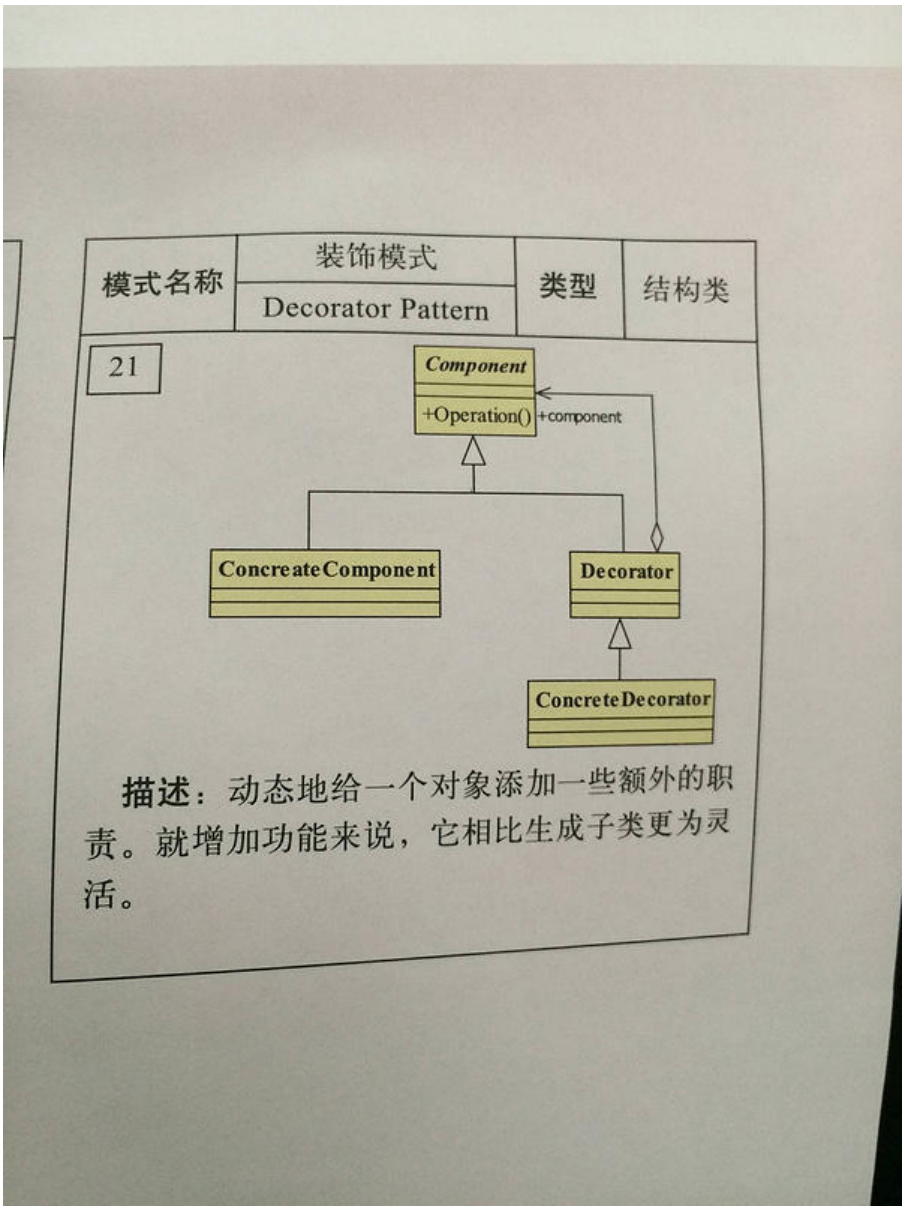
- 装饰类和被装饰类是可以 **独立**的，低耦合的。互相都不用知道对方的存在
- 装饰模式是继承的一种 **替代**方案，**无论包装多少层，返回的对象都是is-a的关系**(上面的例子：包完还是Phone类型)。
- 实现动态扩展，只要 **继承了装饰器**就可以**动态**扩展想要的功能了。

缺点：

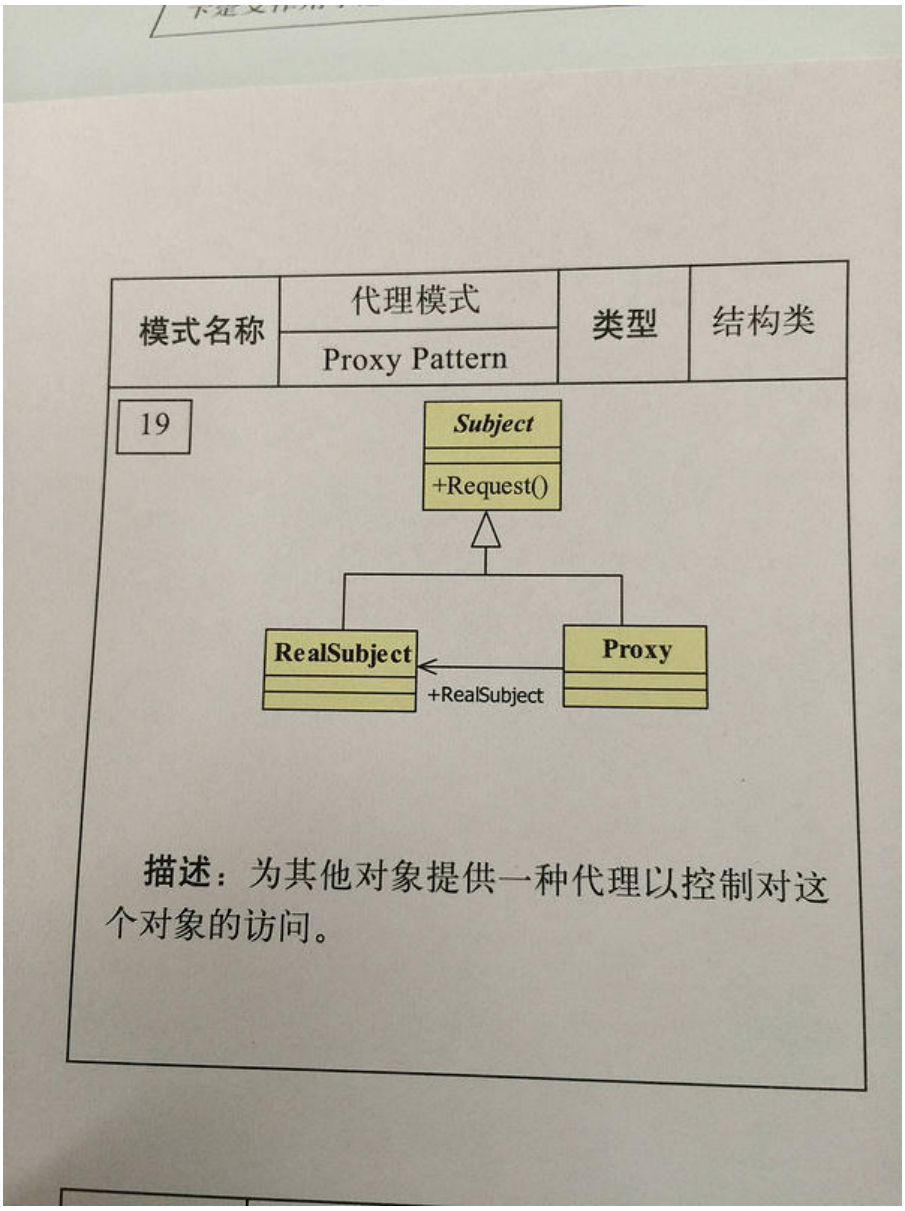
- 多层装饰是比较复杂的，提高了系统的复杂度。不利于我们调试~

三、总结

最后来补充一下包装模式和代理模式的类图：



原文链接：设计模式 --- 装饰模型 (转)



对象增强的三种方式：

- 继承
- 包装模式
- 代理模式

那么只要遇到Java提供给我们的API不够用，我们增强一下就行了。在写代码时，某个类被写死了，能不够用，增强一下就可以了！

原文地址：https://segmentfault.com/a/1190000014771830