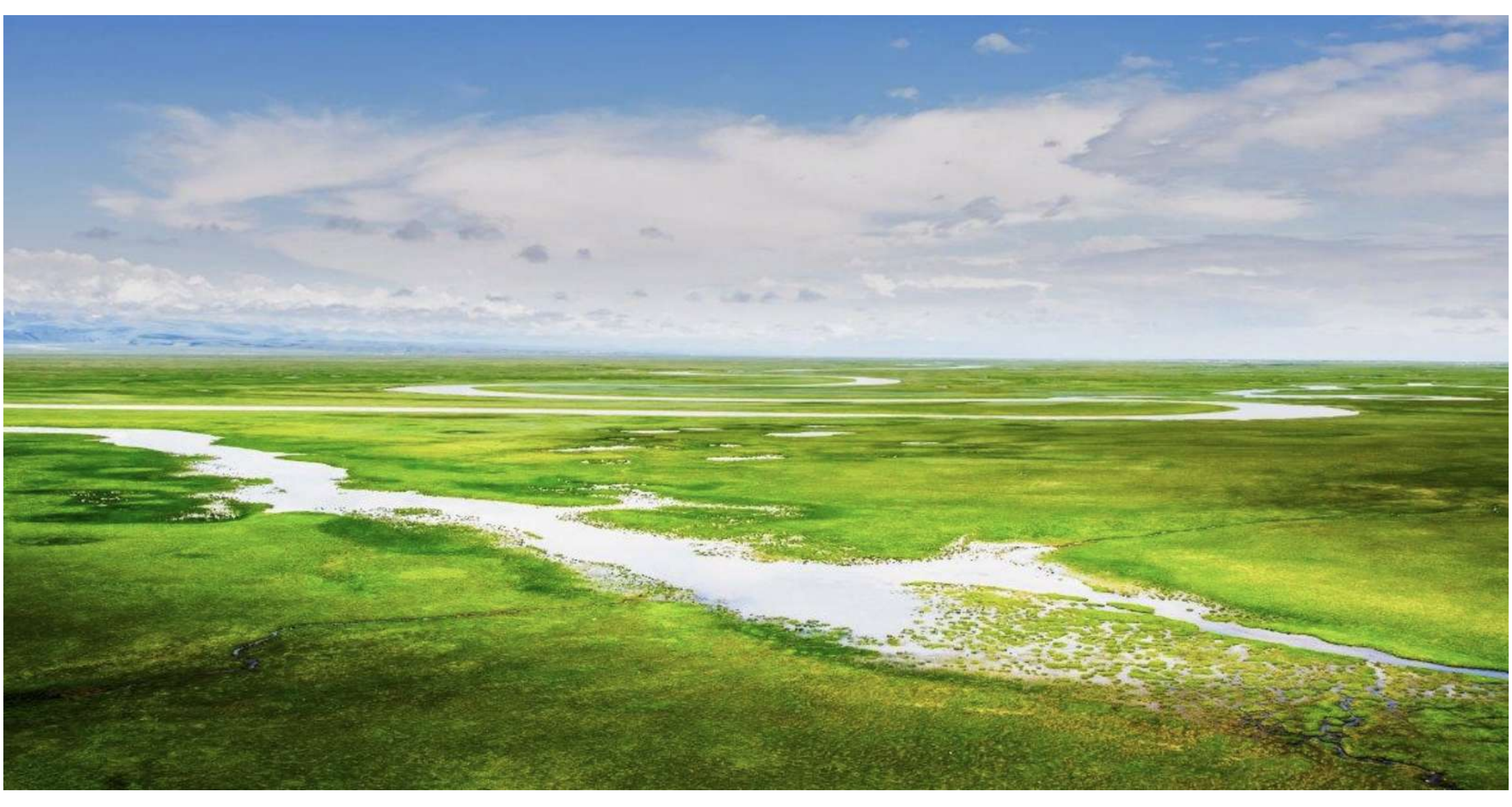


大家好，我是楼仔！

首先了解一下代理模式的定义。

> 为其他对象提供一种代理以控制这个对象的访问。

涉及角色及说明：

> Subject（抽象主题类）：接口或者抽象类，声明真实主题与代理的共同接口方法。
>
> RealSubject（真实主题类）：也叫做被代理类或被委托类，定义了代理所表示的真实对象，负责具体业务逻辑的执行，客户端可以通过代理类间接的调用真实主题类的方法。
>
> Proxy（代理类）：也叫委托类，持有对真实主题类的引用，在其所实现的接口方法中调用真实主题类中相应的接口方法执行。
>
> Client（客户端类）：使用代理模式的地方。

理解：

> 代理模式属于结构型模式。
>
> 代理模式也叫委托模式。
>
> 生活中，比如代购、打官司等等，实际上都是一种代理模式。

代理模式可以分为静态代理和动态代理。

静态代理

创建抽象主题：

```
public interface penguin {
    public void beating();
}
```

创建真实主题：

```
public class littlePenguin implements penguin {
    @Override
    public void beating() {
        System.out.println("打豆豆");
    }
}
```

创建静态代理类：

```
public class penguinProxy {
    private penguin m_penguin;
    public penguinProxy(penguin p) {
        this.m_penguin = p;
    }
    public void beating() {
        System.out.println("打豆豆前");
        m_penguin.beating();
        System.out.println("打豆豆后");
    }
}
```

使用姿势：

```
public static void main(String args[]) {
    penguin penguin1 = new littlePenguin();
    penguinProxy proxy = new penguinProxy(penguin1);
    proxy.beating();
}
```

输出：

```
打豆豆前
打豆豆
打豆豆后
```

静态代理很简单，就是不直接调用littlePenguin中的方法，都是通过代理类来处理。

这个示例是我参考网上写的，不过我觉得这个示例有一个问题，就是在main()中先new一个littlePenguin，然后赋值给代理类penguinProxy，我觉得如果需要对littlePenguin完全进行封装，就应该在代理类中搞一个littlePenguin单例，因为对littlePenguin的使用，不应该暴露到main中，要不然搞这个代理，就没有太大意义，当然这个只是我的个人理解哈。

动态代理

创建动态代理类：

```
public class penguinProxy implements InvocationHandler {
    private Object obj; // 被代理的对象
    public penguinProxy(Object obj) {
        this.obj = obj;
    }
    @Override
    public Object invoke(Object proxy, Method method, Object[] args)throws Throwable {
        System.out.println("海外动态代理调用方法:" + method.getName());
        Object result = method.invoke(obj, args);
        return result;
    }
}
```

使用姿势：

```
public static void main(String args[]) {
    penguin penguin1 = new littlePenguin();
    // 创建动态代理
    penguinProxy proxy = new penguinProxy(penguin1);
    //获取ClassLoader
    ClassLoader loader = penguin1.getClass().getClassLoader();
    //通过Proxy创建代理实例，实际上通过反射来实现的
    penguin penguin2 = (penguin) Proxy.newProxyInstance(loader, new Class[]{penguin.class}, proxy);
    penguin2.beating();
}
```

输出：

```
海外动态代理调用方法:beating
打豆豆
```

动态代理的方式很灵活，后面会在剖析两者的区别。从个人理解，这个应该只有java才能使用这种动态代理的方式吧，不知道PHP和Go是否可以支持，欢迎大家一起讨论哈~~

静态代理 VS 动态代理

静态代理的缺点：

> 静态代理如果接口新增一个方法，除了所有实现类（真实主题类）需要实现这个方法外，所有代理类也需要实现此方法。增加了代码维护的复杂度。
> 代理对象只服务于一种类型的对象，如果要服务多类型的对象。必须要为每一种对象都进行代理，静态代理在程序规模稍大时就无法胜任了。

动态代理的优点：

> 可以通过一个代理类完成全部的代理功能，接口中声明的所有方法都被转移到调用处理器一个集中的方法中处理(InvocationHandler.invoke)。当接口方法数量较多时，我们可以进行灵活处理，而不需要像静态代理那样每一个方法进行中转。
> 动态代理的应用使我们的类职责更加单一，复用性更强。

动态代理的缺点：

> 不能对类进行代理，只能对接口进行代理，如果我们的类没有实现任何接口，那么就不能使用这种方式进行动态代理（因为$Proxy()这个类集成了Proxy,Java的集成不允许出现多个父类）。

优缺点分析

优点：

> 代理作为调用者和真实主题的中间层,降低了模块间和系统的耦合性。可以以一个小对象代理一个大对象,达到优化系统提高运行速度的目的。代理对象能够控制调用者的访问权限，起到了保护真实主题的作用。

缺点：

> 由于在调用者和真实主题之间增加了代理对象，因此可能会造成请求的处理速度变慢。
> 实现代理模式需要额外的工作（有些代理模式的实现非常复杂），从而增加了系统实现的复杂度。

适用场景

当一个对象不能或者不想直接访问另一个对象时，可以通过一个代理对象来间接访问。为保证客户端使用的透明性，委托对象和代理对象要实现同样的接口。
被访问的对象不想暴露全部内容时，可以通过代理去掉不想被访问的内容。

根据适用范围，代理模式可以分为以下几种：

- 远程代理：为一个对象在不同的地址空间提供局部代表，这样系统可以将Server部分的事项隐藏。
- 虚拟代理：如果要创建一个资源消耗较大的对象，可以先用一个代理对象表示，在真正需要的时候才真正创建。
- 保护代理：用代理对象控制对一个对象的访问，给不同的用户提供不同的访问权限。
- 智能引用：在引用原始对象的时候附加额外操作，并对指向原始对象的引用增加引用计数。

后记

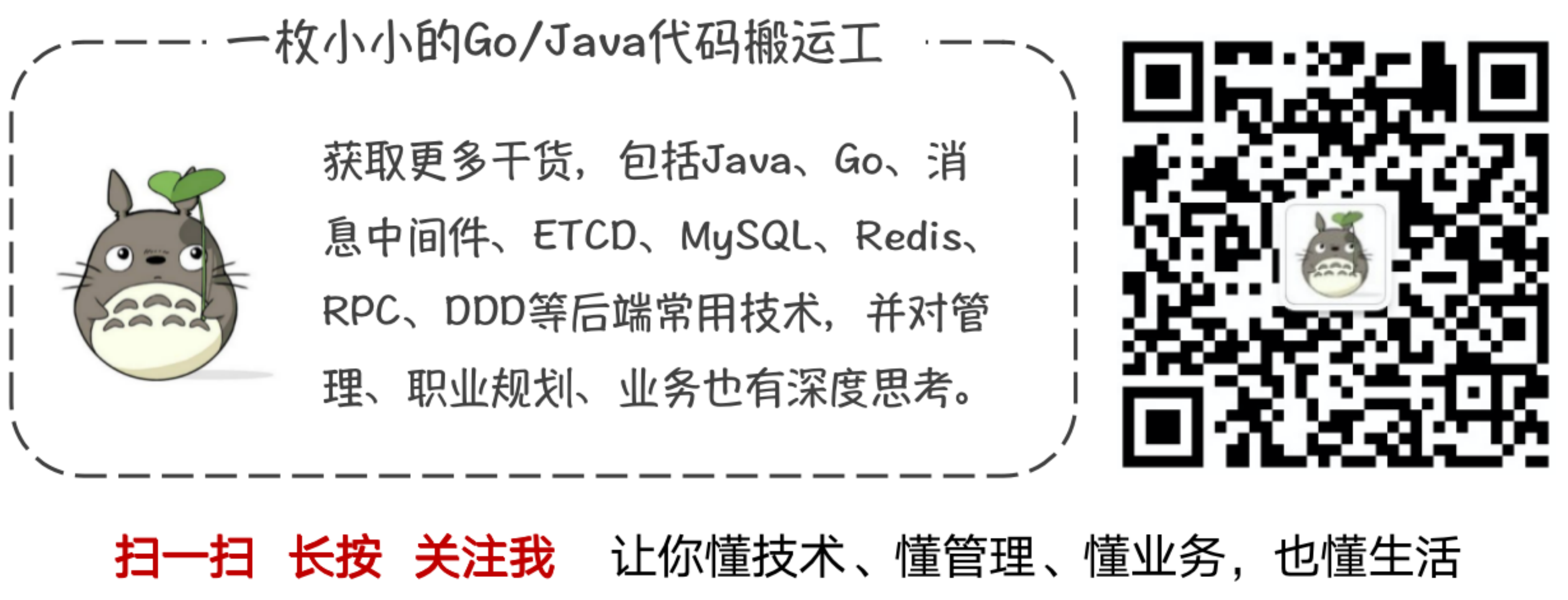
这个是设计模式系列文章中的最后一篇，可能大家会问“设计模式不是有23种么，你这才写了7种”。其实我写这个系列，并不是想把所有的设计模式都写出来，因为这样的文章网上太多了，我主要是想结合自己做过的项目，把之前遇到的设计模式总结出来，然后再结合之前的项目示例给大家讲解，因为只有自己看到过，或者用过，对这个设计模式的理解才能深刻。

其实对于这个“代理模式”，我本来是不想写的，因为之前我没有用过，或者我之前见过，但是现在忘了，因为这个可能会用的比较多，所以就加进来，如果以后项目中遇到了，我再补充实际的项目场景，所以这个设计模式是我的系列文章中，唯一一个没有在我项目x实际场景中运用到的，文章中的知识也是摘抄网上的，便于后续查阅。

后面的设计模式，大家可能也会经常用到，比如适配器、门面模式、装饰者等，后面我就不再特意去写，如果在后续的项目中，我有遇到其它的设计模式，我肯定还会继续写这个系列，其实写文章不是目的，主要是将遇到的设计模式，结合实际的项目进行总结，这样理解就会更加深刻！

学习交流

可以扫下面二维码，关注「楼仔」公众号。



长按二维码，回复 **「加群」**，欢迎一起学习交流哈~~ 👏🏻👏🏻👏🏻

> 尽信书则不如无书，因个人能力有限，难免有疏漏和错误之处，如发现 bug 或者有更好的建议，欢迎批评指正，不吝感激。