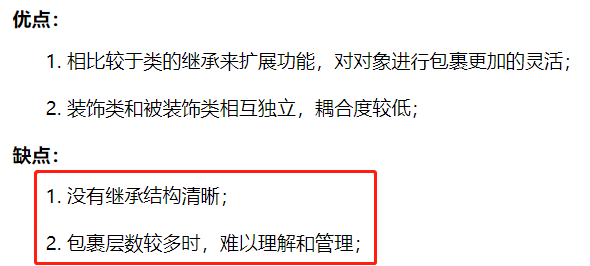
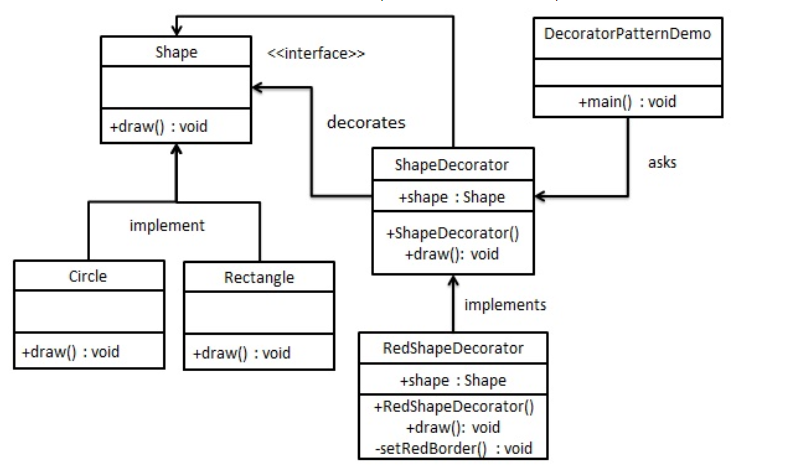
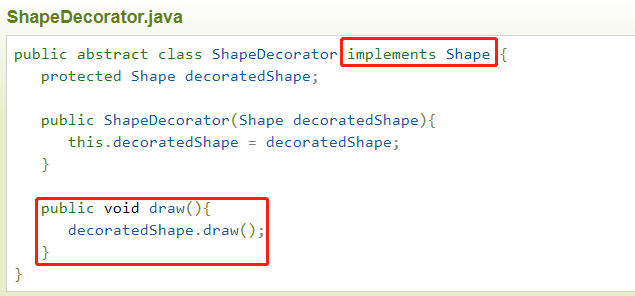
装饰器模式主要对现有的类对象进行包裹和封装，以期望在不改变类对象及其类定义的情况下，为对象添加额外功能。（为现有对象添加新功能）







Draw可以定义为抽象方法，让具体装饰类去实现这个方法

具体装饰类也就先实现对象的方法再基础上进行另外的补充

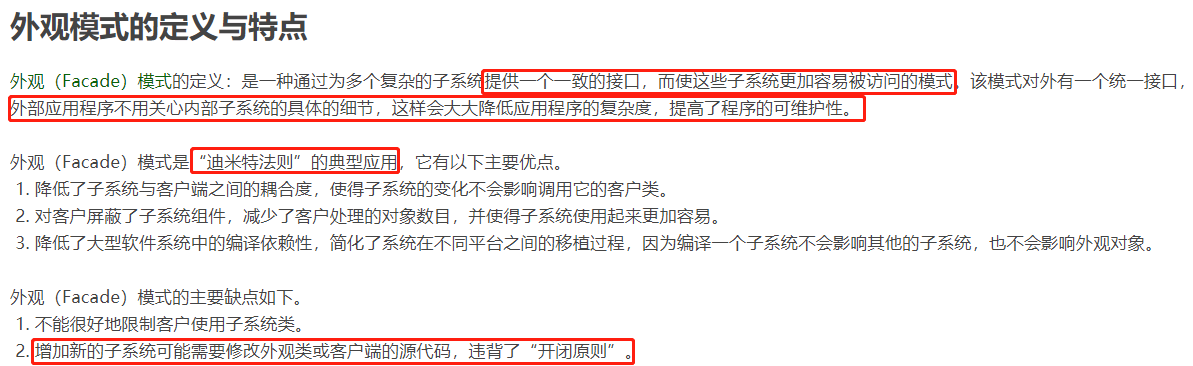


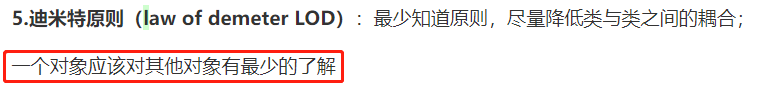
装饰器模式是在原有对象基础上进行补充添加

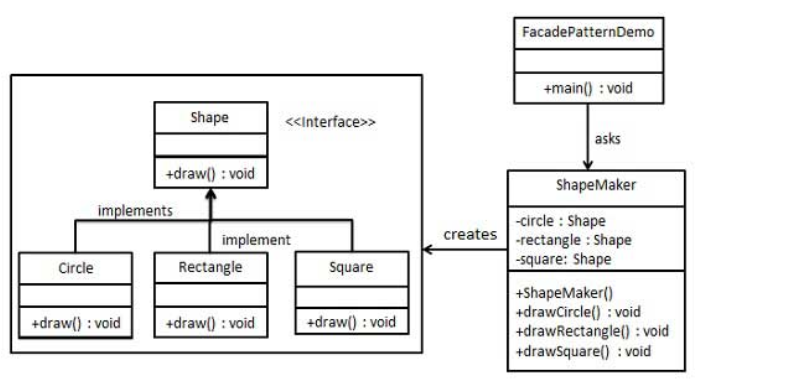
之前的榛果牛奶卡布奇诺采用装饰器模式更为合适，再咖啡的基础上进行添加材料操作

IO流就是装饰器模式，一层包一层

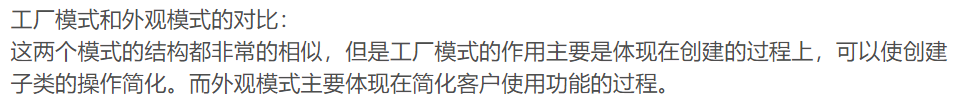
<https://www.runoob.com/design-pattern/decorator-pattern.html>





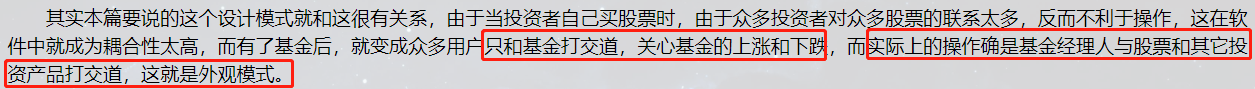


也就是需要图形对象的时候无需去调用具体的类，而通过maker可以获取到对象和工厂模式其实很像，UML图都很像

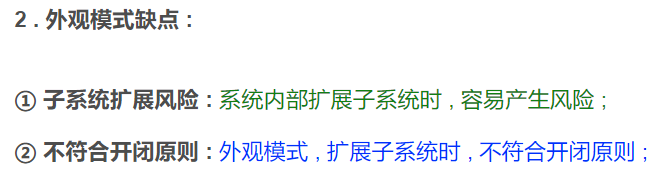


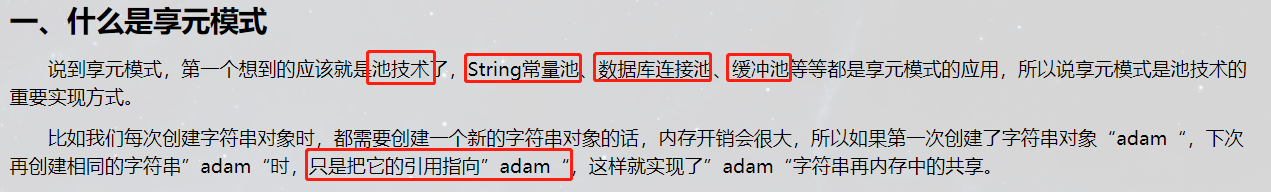
也就是工厂模式是创建型的，其实对象还没获取到，通过工厂模式获取相应对象

外观模式是结构型的，也就是功能型的，其实已有对象了，要用该对象实现某种功能



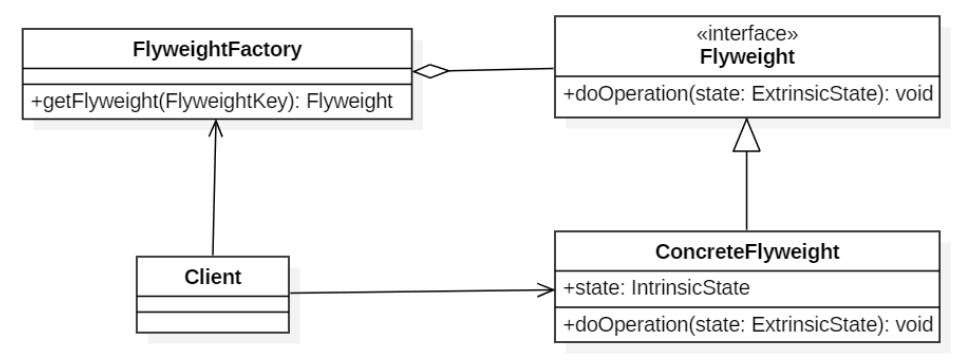
不用具体去操作股票，选择操作基金（缺点：要增加新的股票需要动到基金类，而不是所有的用户都想要新的股票）

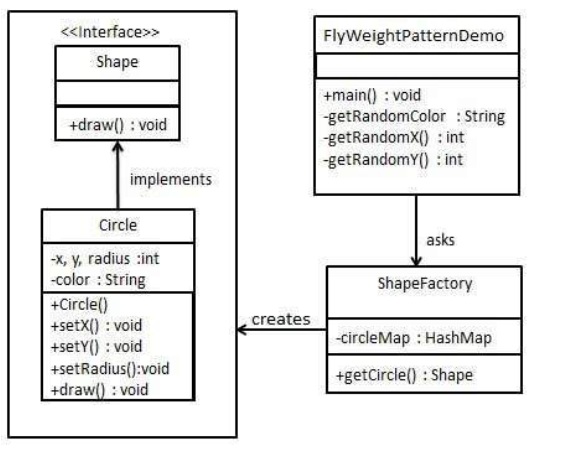


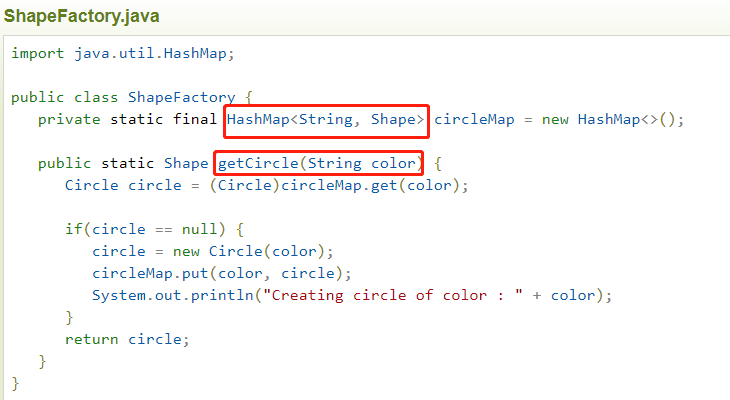


享即为共享的意思，也就是使用过程中的多个对象其实是一个对象进行的变化

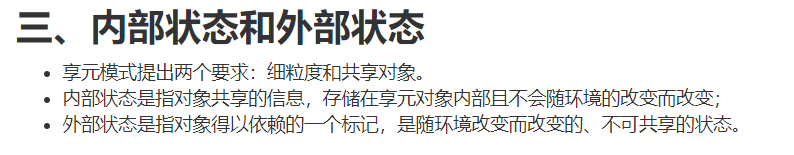


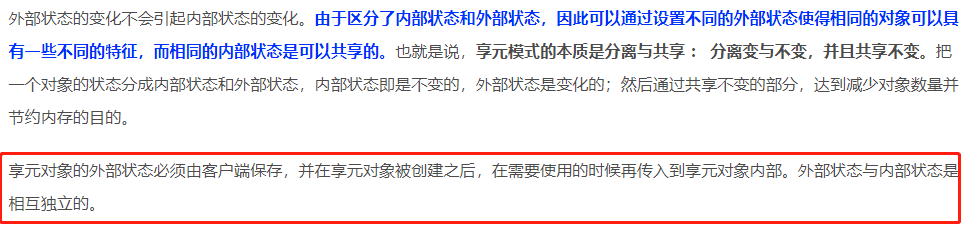






怎么共享，即为在工厂类里存放一个map集合，color是外部状态。





<https://www.runoob.com/design-pattern/flyweight-pattern.html>

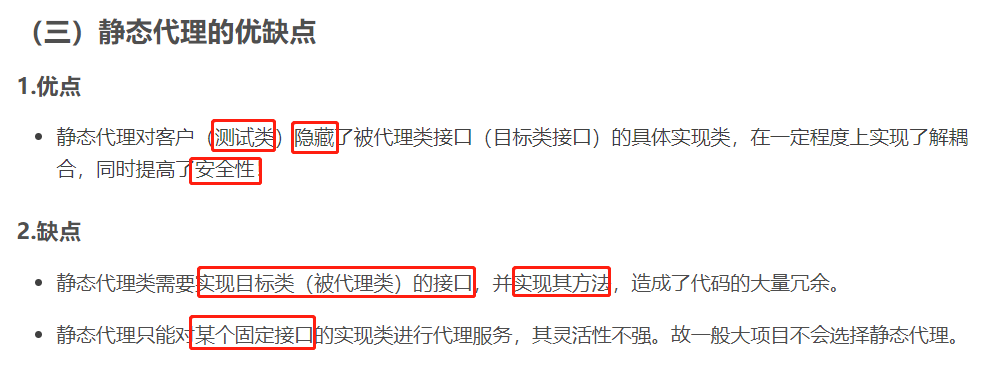
单例模式只有一个对象，而享元模式有多个对象

若是直径确定为1M，多个不同颜色的圆的直径都不变，直径就为内部状态

静态代理，目标对象和代理对象的个数是一对一的

代理可以实现在保持目标对象代码不变的情况下，对对象访问前后进行一些操作

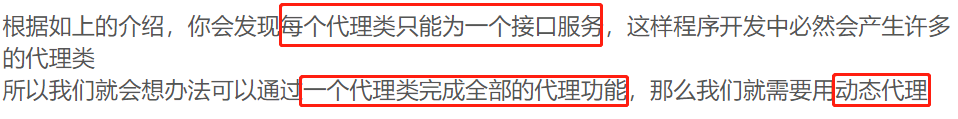
拦截器就是用了代理设计模式的原理



静态需要实现接口

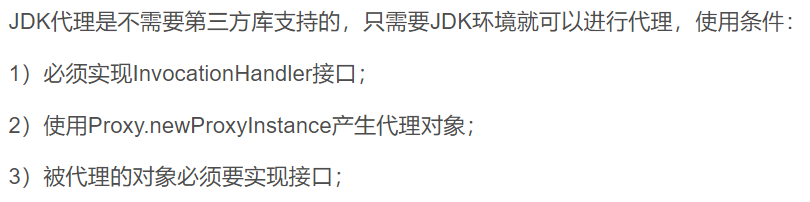


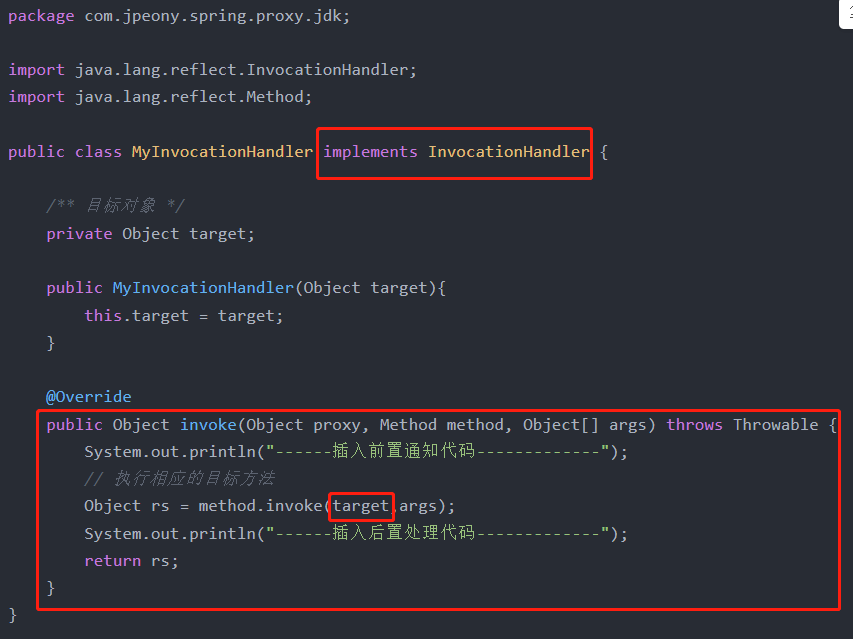
扩展性不强



动态代理是在运行时，通过反射机制实现动态代理，并且能够代理各种类型的对象

动态代理又分为JDK动态代理和CGLIB动态代理，其中CDLIB需要配置maven环境



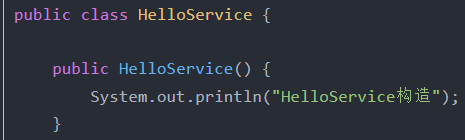




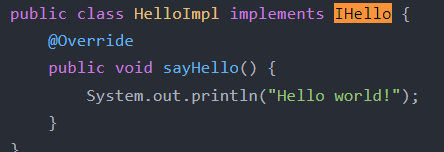
Proxy.newProxyInstance传入类加载器，目标类类型和代理类实例

<https://blog.csdn.net/yhl_jxy/article/details/80586785>

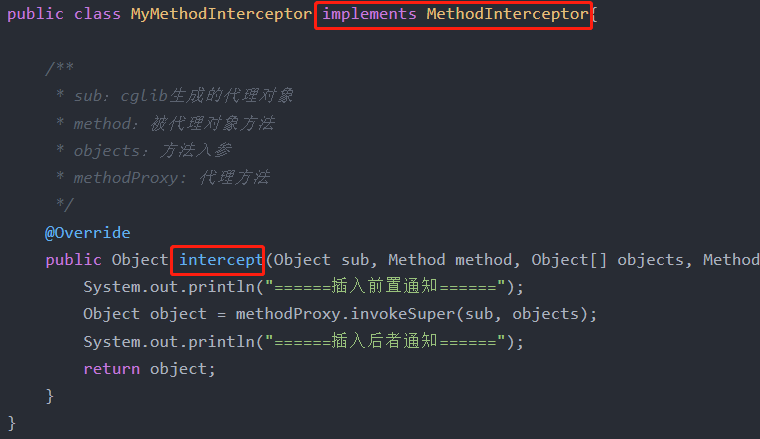
CGLIB动态代理业务类并没有实现任何接口



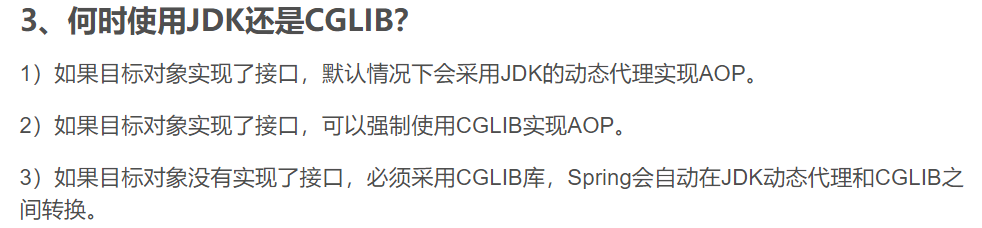
JDK的业务类需要实现接口

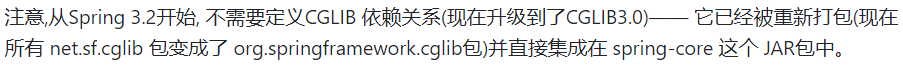


CGLIB动态代理实现MethodInterceptor接口并重写intercept方法



<https://blog.csdn.net/yhl_jxy/article/details/80633194>

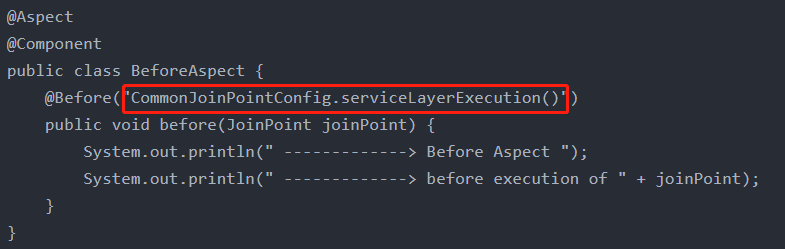




平常项目用代理多用注解

Pointcut: 切点，用于定义哪个方法会被拦截，例如 execution(\* cn.springcamp.springaop.service..(…))

Aspect: 切面配合Before，After，Around使用



还用一种常见的方法是自定义一个注释，在注释前后操作。进行日志，入库等处理



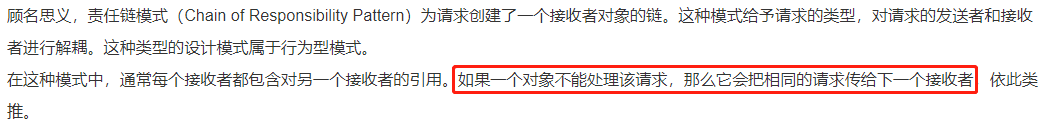
<https://blog.csdn.net/grq15203514615/article/details/87935827>



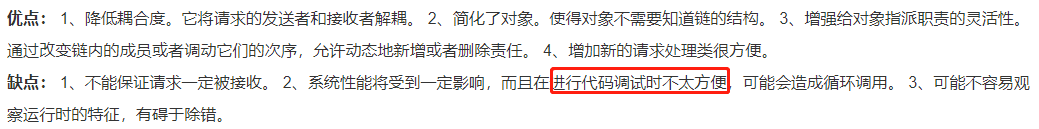
创建型为不是简单new一个对象，设计模式创建对象时更加灵活。

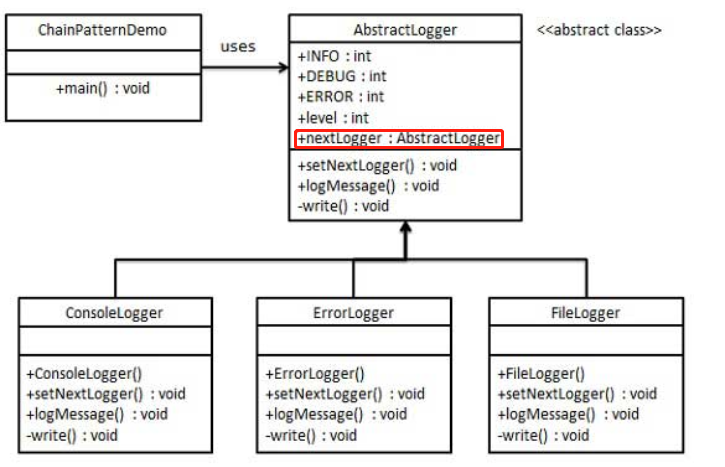
结构型为已有一些对象，通过设计模式使得多个对象变成一个整体功能更为强大或还是一个对象但是能满足特定的需求

行为型为已有多个对象，设计模式方便对象间的相互调用与通信

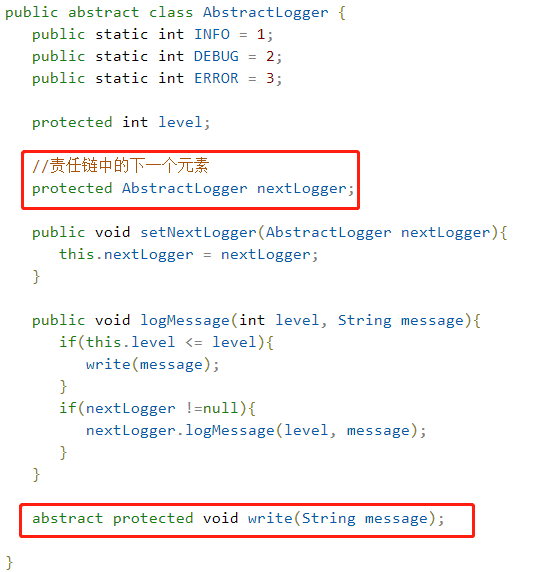


责任链模式也就是一个对象不能完整处理这个请求，通过链表给下一个对象处理





设计起来也不会太复杂，就是类里面有一个链表，通过某种规则让链表中的下一个元素处理请求

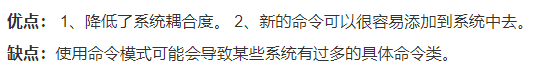


这个logMessage也可以算是低配版的模板设计模式

<https://www.runoob.com/design-pattern/chain-of-responsibility-pattern.html>



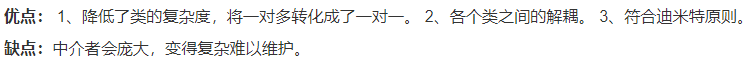
解耦，就行饭店吃饭。**客户就是发出命令的人**，**服务员**的菜单就是**命令调用者**，**后台工作人员**是**具体命令类**（安排菜单中的某个菜给某个厨师做），**厨师**就是**接收者**了，具体执行命令的对象



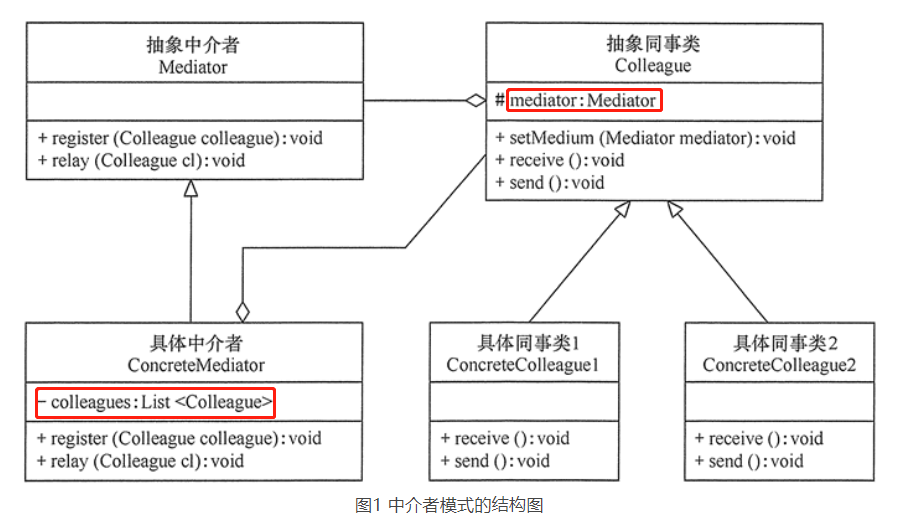
<https://www.runoob.com/design-pattern/command-pattern.html>

<https://www.cnblogs.com/meet/p/5116430.html>

中介者模式（Mediator Pattern）是用来降低多个对象和类之间的通信复杂性。这种模式提供了一个中介类，该类通常处理不同类之间的通信，并支持松耦合，使代码易于维护。中介者模式属于行为型模式。



中介者是为了让两个类不直接发生关系而通过中介者联系，中介者并不偏向任何一方，双方通过中介者相互和另一方发生关系关系是双向的。



<http://c.biancheng.net/view/1393.html>

也就是中介者中保存多个同事对象，随时可以操作同事的方法，同事又有着中介者的对象，可以拜托中介者去操作其他中介者

所以同事间的多对多联系交由中介者就可以变成与中介者的两个多对一联系