# 设计原则

## 单一职责原则

## 开放-封闭原则

## 依赖倒转原则

## 里氏代换原则

## 接口隔离原则

## 迪米特原则（最少知道原则）

# 创建型设计模式

## 简单工厂模式

## 工厂模式

工厂模式定义一个用于创建对象的接口，让子类决定实例化哪一个类。工厂方法使一个类的实例化延迟到子类。

## 抽象工厂模式

## 单例模式

## 建造者模式

## 原型模式

# 结构型设计模式

## 适配器模式

## 装饰者模式

1. 在设计中，应允许行为可以被扩展，而无需修改实现；
2. 除了继承，装饰者模式也可以让我们扩展行为，而装饰者模式包装具体组件
3. 装饰者会导致设计中出现小对象，若过度使用会使程序变得复杂，
4. 继承属于扩展形式之一，但不见得达到弹性设计的最佳方式
5. 每个组件都可单独使用或者被包装起来
6. 装饰者模式可以扩展组件的状态；
7. 装饰可以加上新的方法；
8. 可以为组件添加新的方法而修改装饰者，反之亦可。

## 代理模式

定义:

为其他对象提供一种代理以控制对这个对象的访问。

Subject抽象主题类：

抽象主题类，可以是一个接口，也可以是一个最普通的业务类型定义，无特殊要求。

RealSubject具体主题角色：

也叫被委托的角色，被代理的角色。它是冤大头，是业务逻辑的具体执行者。

Proxy代理主题类:

也叫委托类，代理类。它负责对真实角色的应用，把所有抽象主题类定义的方法限制委托给真实主题角色，并在真实主题角色处理前后做预处理工作和善后工作。

### 普通代理

在该模式下，调用者只知代理不知道真实的角色是谁，屏蔽了真实角色的变更对高层模块的影响，真实主题角色想怎么改就怎么改，对高层次模块没有任何影响,只要实现了接口所对应的方法，该模式非常适合扩展性高的场合。

用户设置代理IP地址，确保用户知道代理的存在。调用者只需要知道代理存在就好，而不用知道代理了谁，对真实角色的构造，调用进行项目组约定。

### 强制代理

强制代理的定义：从真实代理查找到代理角色，不允许直接访问真实角色。高层模块只要调用getProxy就可以访问真实角色的所有方法，它根本就不需要产生一个代理出来，代理的管理已经由真实角色自己完成。

实现方案：在真实角色中定义自己的代理者，每个流程的执行都首先判是否有代理存在，否则提示无法访问在代理角色中，代理的代理返回this；

### 动态代理

根据本代理的接口生成所有方法，也就是说**给定一个代理，动态代理会宣称自己已经实现该接口下，所有的方法。**

两条独立的发展路线，动态代理实现代理的职责，业务逻辑Subject实现相关的逻辑功能。两者之间没有必然的耦合关系。通知Advice从另一个切面切入，最终高层模块也就是Client进行耦合，最后完成封装任务。

实现阶段不用关心代理谁，而在运行阶段制定代理哪一个对象。AOP（Aspect Oriented Programming）面向横切面编程。核心就是动态代理。

其实其核心就是这个动态问题的解决，利用C++中的多态，回调等方案。

### 优点

1. 职责清晰（实现好内部结构就可以，具体客户要求有代理进行分化）
2. 高扩展性，具体主题随时变化，只要他实现了接口，无论如何都逃不出代理的手掌，所以代理无论如何都是可以使用的。
3. 智能化。

## 外观模式

## 桥接模式

## 组合模式

## 享元模式

# 行为型模式

## 策略模式

## 模板方法模式

## 观察者模式

## 迭代子模式

## 责任链模式

## 命令模式

## 备忘录模式

## 状态模式

## 访问者模式

## 中介者模式

## 解释器模式

# 其他

## 并发型模式

## 线程池模式