设计模式是长期的实践出来的解决方案。目的是提高代码的复用性与兼容性和可维护性。

希望能够从重复造轮子走向模式复用。但这个很困难，实际开发中还是先实现功能，在找出代码的缺陷，不符合什么设计理念，去找有对应优势的设计模式拿来参考，再进行优化。

但是设计原则要时刻思考：

我们设计的类要专注一个职责，避免职责混乱。

软件要对扩展开放，对修改关闭，可以新增代码，无需修改原有代码。

子类替代父类，不会破坏程序的正确性，这个就要子类重写父类抽象方法要遵循父类的接口契约。

客户端只依赖需要的接口，不依赖不需要的接口，就是不需要的接口不要暴露给它。

高层模块不依赖底层模块，都只能依赖于抽象。

策略模式：

首先实现一个策略模式的基类，包含方法函数

策略A继承基类，重写方法函数

策略B继承基类，重写方法函数

然后需要一个第三的上下文切换类，集成这些策略，可使用Unique\_ptr去接收策略的初始化，这些要是用移动语义；并且提供设置策略的接口，用于后面更换策略；再提供一个执行函数，去调用传入的策略类的方法函数。

适配器模式：

目标类的接口：（我们想要的形式）

适配者类的接口：（现有的功能，但与我们想要的不兼容）

适配器类，去继承这两个了类，重写目标类的接口，在这里面去调用适配者类的接口，并进行修改。

这样客户端就只需要调用适配器类的接口就可以了。

比如说栈，这里就是在接口里实现了一个队列的反向，栈就是新接口，队列就是老的接口。

观察者模式：

主要是利用主题去维护观察者列表，当主体状态变化时自动调用所有观察者的更新方法，实现对象的松耦合通知机制。

观察者的基类

具体的观察者A

具体的观察者B

主题接口

具体主题：去管理观察者列表和状态，调用观察者暴露出来的方法

调用：具体主题加入观察者对象，确认什么事件去调用通知方法