只用于记录自己能理解使用场景的设计模式

**单例模式**：

以Spring为例，一个controller可能会依赖多个service层对象；如果每次调用controller层的方法时，都需要去从0创建，那么就很可能会顺带创建多个service层对象🡪导致创建对象的开销变大

而单例模式就是为了将“创建开销大”的对象缓存起来使用

**策略模式**：

假设我需要根据爬虫源不同，准备不同的爬虫器，最简单的方法是用if else判断，但这样写出来的代码可读性不好，并且每次修改/添加爬虫器时都需要修改原有的代码，违背了“**开闭原，即对拓展开放，对修改关闭**”；但如果定义一个爬虫器接口a，并为每个爬虫源都配置一个爬虫器，同时根据爬虫源信息调用对应的爬虫器，在添加/修改爬虫器时，就不需要对调用爬虫器的代码进行修改🡪提高了代码的拓展性和可读性

需要if else进行大量判断的场景，可以考虑用策略模式优化

**模板方法：**

举例：用一个父类，将实体类中一些通用属性（比如id，create\_time，update\_time等等）/通用方法封装好

实体类子类继承父类后，就不需要重新写一遍通用属性/方法

**观察者模式**：

分为三个部分，“事件”，“事件监听者”，“事件发布者”；事件发布者发布事件后，事件监听者根据事件内容执行对应的行为

如：Spring在初始化过程中会在run方法执行到某个阶段时，让事件发布者发布事件，对该事件感兴趣的监听器会创建日志实例