# 1、工厂模式

在工厂模式中，我们在创建对象时不会对客户端暴露创建逻辑，并且是通过使用一个共同的接口来指向新创建的对象。工厂模式作为一种创建模式，一般在创建复杂对象时，考虑使用；在创建简单对象时，建议直接new完成一个实例对象的创建。

## 1.1、简单工厂模式

主要特点是需要在工厂类中做判断，从而创造相应的产品，当增加新产品时，需要修改工厂类。使用简单工厂模式，我们只需要知道具体的产品型号就可以创建一个产品。

缺点：工厂类集中了所有产品类的创建逻辑，如果产品量较大，会使得工厂类变的非常臃肿。

## 1.2、工厂方法模式

定义一个创建对象的接口，其子类去具体现实这个接口以完成具体的创建工作。如果需要增加新的产品类，只需要扩展一个相应的工厂类即可。

缺点：产品类数据较多时，需要实现大量的工厂类，这无疑增加了代码量。

## 1.3、抽象工厂模式

抽象工厂模式提供创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

当存在多个产品系列，而客户端只使用一个系列的产品时，可以考虑使用抽象工厂模式。

缺点：当增加一个新系列的产品时，不仅需要现实具体的产品类，还需要增加一个新的创建接口，扩展相对困难。

# 2、策略模式

策略模式是指定义一系列的算法，把它们单独封装起来，并且使它们可以互相替换，使得算法可以独立于使用它的客户端而变化，也是说这些算法所完成的功能类型是一样的，对外接口也是一样的，只是不同的策略为引起环境角色环境角色表现出不同的行为。

相比于使用大量的if...else，使用策略模式可以降低复杂度，使得代码更容易维护。

缺点：可能需要定义大量的策略类，并且这些策略类都要提供给客户端

# 3、适配器模式

适配器模式可以将一个类的接口转换成客户端希望的另一个接口，使得原来由于接口不兼容而不能在一起工作的那些类可以在一起工作。通俗的讲就是当我们已经有了一些类，而这些类不能满足新的需求，此时就可以考虑是否能将现有的类适配成可以满足新需求的类。适配器类需要继承或依赖已有的类，实现想要的目标接口。

缺点：过多地使用适配器，会让系统非常零乱，不易整体进行把握。比如，明明看到调用的是 A 接口，其实内部被适配成了 B 接口的实现，一个系统如果太多出现这种情况，无异于一场灾难。因此如果不是很有必要，可以不使用适配器，而是直接对系统进行重构。

# 4、单例模式

单例模式顾名思义，保证一个类仅可以有一个实例化对象，并且提供一个可以访问它的全局接口。实现单例模式必须注意一下几点：

单例类只能由一个实例化对象。

单例类必须自己提供一个实例化对象。

单例类必须提供一个可以访问唯一实例化对象的接口。

单例模式分为懒汉和饿汉两种实现方式。

## 4.1、懒汉单例模式

懒汉：故名思义，不到万不得已就不会去实例化类，也就是说在第一次用到类实例的时候才会去实例化一个对象。在访问量较小，甚至可能不会去访问的情况下，采用懒汉实现，这是以时间换空间。

## 4.2、饿汉单例模式

饿汉：饿了肯定要饥不择食。所以在单例类定义的时候就进行实例化。在访问量比较大，或者可能访问的线程比较多时，采用饿汉实现，可以实现更好的性能。这是以空间换时间。

# 5、原型模式

原型模式：用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象。通俗的讲就是当需要创建一个新的实例化对象时，我们刚好有一个实例化对象，但是已经存在的实例化对象又不能直接使用。这种情况下拷贝一个现有的实例化对象来用，可能会更方便。

以下情形可以考虑使用原型模式：

当new一个对象，非常繁琐复杂时，可以使用原型模式来进行复制一个对象。比如创建对象时，构造函数的参数很多，而自己又不完全的知道每个参数的意义，就可以使用原型模式来创建一个新的对象，不必去理会创建的过程。

当需要new一个新的对象，这个对象和现有的对象区别不大，我们就可以直接复制一个已有的对象，然后稍加修改。

当需要一个对象副本时，比如需要提供对象的数据，同时又需要避免外部对数据对象进行修改，那就拷贝一个对象副本供外部使用。

# 6、模板模式

模板模式：定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

当多个类有相同的方法，并且逻辑相同，只是细节上有差异时，可以考虑使用模板模式。具体的实现上可以将相同的核心算法设计为模板方法，具体的实现细节有子类实现。

缺点:每一个不同的实现都需要一个子类来实现，导致类的个数增加，使得系统更加庞大。

# 7、建造者模式

建造者模式：将复杂对象的构建和其表示分离，使得相同的构建过程可以产生不同的表示。

以下情形可以考虑使用建造者模式：

对象的创建复杂，但是其各个部分的子对象创建算法一定。

需求变化大，构造复杂对象的子对象经常变化，但将其组合在一起的算法相对稳定。

建造者模式的优点：

将对象的创建和表示分离，客户端不需要了解具体的构建细节。

增加新的产品对象时，只需要增加其具体的建造类即可，不需要修改原来的代码，扩展方便。

# 8、外观模式

外观模式：为子系统中的一组接口定义一个一致的界面；外观模式提供一个高层的接口，这个接口使得这一子系统更加容易被使用；对于复杂的系统，系统为客户端提供一个简单的接口，把负责的实现过程封装起来，客户端不需要连接系统内部的细节。

以下情形建议考虑外观模式：

设计初期阶段，应有意识的将不同层分离，层与层之间建立外观模式。

开发阶段，子系统越来越复杂，使用外观模式提供一个简单的调用接口。

一个系统可能已经非常难易维护和扩展，但又包含了非常重要的功能，可以为其开发一个外观类，使得新系统可以方便的与其交互。

优点：

实现了子系统与客户端之间的松耦合关系。

客户端屏蔽了子系统组件，减少了客户端所需要处理的对象数据，使得子系统使用起来更方便容易。

# 9、组合模式

组合模式：将对象组合成树形结构以表示“部分-整体”的层次结构，组合模式使得客户端对单个对象和组合对象的使用具有一直性。

既然讲到以树形结构表示“部分-整体”，那可以将组合模式想象成一根大树，将大树分成树枝和树叶两部分，树枝上可以再长树枝，也可以长树叶，树叶上则不能再长出别的东西。

以下情况可以考虑使用组合模式：

希望表示对象的部分-整体层次结构。

希望客户端忽略组合对象与单个对象的不同，客户端将统一的使用组合结构中的所有对象

更好的划分了设计层次，对于后期维护更加的容易。

# 10、代理模式

代理模式：为其它对象提供一种代理以控制这个对象的访问。在某些情况下，一个对象不适合或者不能直接引用另一个对象，而代理对象可以在客户端和目标对象之间起到中介作用。

优点：

职责清晰。真实的角色只负责实现实际的业务逻辑，不用关心其它非本职责的事务，通过后期的代理完成具体的任务。这样代码会简洁清晰。

代理对象可以在客户端和目标对象之间起到中介的作用，这样就保护了目标对象。

扩展性好。