设计模式总结笔记

- **设计模式 3 大类：**

  创建型模式、结构型模式和行为型模式。

- **面向对象 6 大原则：**

1. 单一职责原则：就一个类而言，应该仅有一个引起它变化的原因。(ASD)软件设计真正要做的许多内容，就是发现职责，并把那些职责相互分离。其实要求判断是否应该分离出类来，也不难，那就是如果你能想到多于一个的动机去改变一个类，那么这个类就有多于一个的职责，就应该考虑类的职责分离。
2. 开放-封闭原则：对扩展开放，对修改封闭。软件实体(类、模块或函数)应该是能扩展的，但是不可修改。
3. 依赖倒置原则：抽象不应该依赖于细节，细节必须依赖于抽象。高程模块不应该依赖与底层模块，两者都应该依赖与抽象。 针对接口编程，不应该针对实现编程。
4. 里氏代换原则：一个软件里，把所有的父类替换成它们的子类，程序的行为没有变化。子类型必须能够替换掉它们的父类型。
5. 迪米特发则：如果两个类不必直接通信，那么这两个类就不应该发生直接的相互作用，如果其中的一个类要调用另一个类的某一个方法，可以通过第三方转发这个调用。
6. 合成/聚合复用原则：尽量使用合成/聚合，尽量不要使用类继承。聚合是一种弱的拥有关系，体现的是A对象可以包含B对象，但B对象不是A对象的一部分。合成是一种强的拥有关系， 体现了严格的部分和整体的关系，部分和整体的生命周期一样。

- **GoF 23 个设计模式**

1、简单工厂模式： 将具有相同行为的对象封装成一个类，使用一个工厂类创建它们。

2、策略模式： 将不同的算法或规则封装成不同的类，使用context类来管理它们。策略模式就是

用来封装算法的。

3、代理模式： 1)远程代理：比如WebService; 2)虚拟代理：为了优化性能，代理需要开销很大的对象； 3）安全代理：需要控制真实对象的访问权限； 4)智能代理：代理额外处理些事情。

4、装饰模式：动态地给一个对象添加一些额外的职责，就增加功能来说，装饰模式比生成子类更加灵活。

5、工厂模式：定义一个创建对象的接口，让子类决定实例化哪一个类。工厂模式使一个类的实例化延迟到其子类之中。

6、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。 原型模式其实就是从一个对象再创建另一个可定制的对象，而且不需要知道任何创建的细节。

7、模板方法模式： 定义一个操作中的算法骨架，将一些步骤延迟到子类中，模板方法模式可以让一个子类不用改变一个算法的结构即可重定义该算法的某些特定步骤。

8、外观模式：facade，为子系统的一组接口提供一个一致的界面，此模式定义了一个高层接口，这一接口使得这一系统更加容易使用。在设计阶段，应该有意识将两个不同的层分离，比如经典的三层架构，就需要考虑在数据访问层和业务逻辑层、业务逻辑层和表示层之间建立外观facade结构。其次，在开发阶段，子系统由于不断的重构、演化变得越来越复杂，大多数模式在使用时也会产生很多很多的小类。这时增加外观facade可以提供一个简单的接口，减少他们之间的依赖。第三，在维护一个遗留的大型系统时，可能这个系统已经非常难以维护和扩展了，但因为它们包含非常重要的功能，新的需求开发必须要依赖于它。此时，用外观模式facade也是非常合适的。

9、建造者模式，将一个复杂的对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。与模板模式有点类似，不同之处是有个Director指挥者类，builder类。buider类可以建造不同的构件，director类可以按不同的顺序装配构件。

10、观察者模式，定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象。这个主题对象在状态发生改变时，会通知所有观察者对象，使他们能够自动更新自己。观察者和主题对象不应该相互依赖，而应该有一个抽象基类。

委托和事件模型就是 观察者模式的一种改进。委托就是一种引用方法的类型，一旦为委托分配了方法，委托将与该方法具有完全相同的方法。委托方法的使用可以像任何其他方法一样，具有参数和返回值。委托可以看作是对函数的抽象，是函数的“类”，委托的实例将代表一个函数。

11、抽象工厂模式：提供一个创建一系列相关或相互依赖的对象的接口，而无需指定他们具体的类。

区别： 简单工厂模式，一个工厂类创建多个具有相同基类的子类的实例。

工厂模式，工厂类只提供创建接口，由子工厂类决定所创建的具体子类。

抽象工厂模式： 工厂类提供多个创建接口，可以创建一系列相关的类的实例。

简单工厂模式可以使用反射，依赖注入。

12、状态模式：当一个对象的内在状态改变时，允许改变其行为，这个对象看起来像改变了其类。状态模式主要解决的是当控制一个对象状态转换的条件表达式过于复杂时的情况。把状态的判断逻辑转移到表示不同状态的一系列类当中，可以把复杂的判断逻辑简化。

13、适配器模式：Adapter，将一个类的接口转换成客户所需要的另一种接口。Adapter模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

使用条件：系统的数据和行为都正确，但接口不符时，我们应该考虑使用适配器，目的是使控制范围之外的一个原有对象与某个接口匹配。适配器模式主要是用于希望使用现有类，但接口又与复用环境不一致的情况。

14、备忘录模式：在不破坏代码封装性的前提下，获取一个对象内部的状态，并在该对象之外保存这些状态，这样以后就可以将对象恢复到以前保存的状态。

15、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。组合模式使用户对单个对象和组合对象的使用具有一致性。

16、迭代器模式：

17、单实例模式：

18、桥接模式：Bridge，将抽象部分与它的实现部分分离，使它们都可以独立地变化。即将对象的实现(或行为)独立成一个类，对象通过引用实现类(或行为类)而拥有行为。

在我们需要从多角度去分类实现对象时，使用继承会造成大量类的增加，不能满足开放-封闭原则时，就需要考虑桥接模式了。

19、命令模式：将请求封装为一个对象，从而使你可用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可撤销的操作。

敏捷开发原则告诉我们，不要为代码添加一些基于猜测的、实际不需要的功能，只有在实际需要时再重构代码。

20、职责链模式：使多个对象都有机会处理请求，从而避免请求的发送者和接收者之间的耦合关系。将这些对象连成一条链，并沿着这条链传递请求，直到有一个对象处理它为止。

21、中介者模式：用一个中介对象来封装一系列对象的交互。中介者使各对象之间不需要显式引用，从而使其耦合松散，而且可心独立改变它们之间的交互。

22、享元模式：Flyweight，运用共享技术，有效地支持大量细粒度对象。

23、解释器模式：Interpret，给定一个语言，定义它的方法的一种表示，并定义一个解释器，这个解

释器使用该表示来解释语言中的句子。

24、访问者模式：Visitor，表示一个作用于某对象结构中各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。

访问者的目的是把处理从数据结构中分离出来，当系统具有比较稳定的数据结构和易于变化的算法时，使用访问者模式是比较合适的。