快速记忆23种设计模式

一、设计模式的来源

设计模式（Design Pattern）是前辈们对代码开发经验的总结，是解决特定问题的一系列套路。它不是语法规定，而是一套用来提高代码可复用性、可维护性、可读性、稳健性以及安全性的解决方案。

1995 年，GoF（Gang of Four，四人组/四人帮）合作出版了《设计模式：可复用面向对象软件的基础》一书，共收录了 23 种设计模式，从此树立了软件设计模式领域的里程碑，人称「GoF设计模式」。

二、设计模式的六大原则（SOLID）

总原则——开闭原则（Open Closed Principle）

> 一个软件实体，如类、模块和函数应该**对扩展开放，对修改关闭**。

在程序需要进行拓展的时候，不能去修改原有的代码，而是要扩展原有代码，实现一个热插拔的效果。所以一句话概括就是：为了使程序的扩展性好，易于维护和升级。

想要达到这样的效果，我们需要使用接口和抽象类等。

1、单一职责原则（Single Responsibility Principle）

> 一个类应该只有一个发生变化的原因。

不要存在多于一个导致类变更的原因，也就是说每个类应该实现单一的职责，否则就应该把类拆分。

2、里氏替换原则（Liskov Substitution Principle）

> 所有引用基类的地方必须能透明地使用其子类的对象。

任何基类可以出现的地方，子类一定可以出现。里氏替换原则是继承复用的基石，只有当衍生类可以替换基类，软件单位的功能不受到影响时，基类才能真正被复用，而衍生类也能够在基类的基础上增加新的行为。

里氏代换原则是对“开-闭”原则的补充。实现“开闭”原则的关键步骤就是抽象化。而基类与子类的继承关系就是抽象化的具体实现，所以里氏替换原则是对实现抽象化的具体步骤的规范。里氏替换原则中，**子类对父类的方法尽量不要重写和重载**。因为父类代表了定义好的结构，通过这个规范的接口与外界交互，子类不应该随便破坏它。

3、依赖倒置原则（Dependence Inversion Principle）

> 1、上层模块不应该依赖底层模块，它们都应该依赖于抽象。
> 2、抽象不应该依赖于细节，细节应该依赖于抽象。

面向接口编程，依赖于抽象而不依赖于具体。写代码时用到具体类时，不与具体类交互，而与具体类的上层接口交互。

4、接口隔离原则（Interface Segregation Principle）

> 1、客户端不应该依赖它不需要的接口。
> 2、类间的依赖关系应该建立在最小的接口上。

每个接口中不存在子类用不到却必须实现的方法，如果不然，就要将接口拆分。使用多个隔离的接口，比使用单个接口（多个接口方法集合到一个的接口）要好。

5、迪米特法则（最少知道原则）(Law of Demeter)

> 只与你的直接朋友交谈，不跟“陌生人”说话。

一个类对自己依赖的类知道的越少越好。无论被依赖的类多么复杂，都应该将逻辑封装在方法的内部，通过public方法提供给外部。这样当被依赖的类变化时，才能最小的影响该类。

最少知道原则的另一个表达方式是：只与直接的朋友通信。类之间只要有耦合关系，就叫朋友关系。耦合分为依赖、关联、聚合、组合等。我们称出现为成员变量、方法参数、方法返回值中的类为直接朋友。局部变量、临时变量则不是直接的朋友。我们要求陌生的类不要作为局部变量出现在类中。

6、合成复用原则（Composite Reuse Principle）

> 尽量使用对象组合/聚合，而不是继承关系达到软件复用的目的。

合成或聚合可以将已有对象纳入到新对象中，使之成为新对象的一部分，因此新对象可以调用已有对象的功能。

三、设计模式的三大类

创建型模式（Creational Pattern）：对类的实例化过程进行了抽象，能够将软件模块中**对象的创建**和对象的使用分离。

（5种）工厂模式、抽象工厂模式、单例模式、建造者模式、原型模式

结构型模式（Structural Pattern）：关注于对象的组成以及对象之间的依赖关系，描述如何将类或者对象结合在一起形成更大的结构，就像**搭积木**，可以通过简单积木的组合形成复杂的、功能更为强大的结构。

（7种）适配器模式、装饰者模式、代理模式、外观模式、桥接模式、组合模式、享元模式

行为型模式（Behavioral Pattern）：关注于对象的行为问题，是对在不同的对象之间划分责任和算法的抽象化；不仅仅关注类和对象的结构，而且重点关注它们之间的**相互作用**。

（11种）策略模式、模板方法模式、观察者模式、迭代器模式、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

记忆口诀：行状责中模访解备观策命迭（形状折中模仿，戒备观测鸣笛）

四、23种设计模式

----------创建型模式----------

工厂模式

工厂模式（Factory Pattern）是Java中最常用的设计模式之一。

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。

定义一个创建对象的接口，让其子类自己决定实例化哪一个工厂类，工厂模式使其创建过程延迟到子类进行。

> 应用实例：您需要一辆汽车，可以直接从工厂里面提货，而不用去管这辆汽车是怎么做出来的，以及这个汽车里面的具体实现。而至于需要哪个牌子的汽车，就到哪个牌子的工厂。

抽象工厂模式

抽象工厂模式（Abstract Factory Pattern）是围绕一个超级工厂创建其他工厂。该超级工厂又称为其他工厂的工厂。

在抽象工厂模式中，接口是负责创建一个相关对象的工厂，不需要显式指定它们的类。每个生成的工厂都能按照工厂模式提供对象。

提供一个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。

> 应用实例：对于一个家庭来说，可能有商务女装、商

> 务男装、时尚女装、时尚男装，都是成套的，即一系列具体产品。假设一种情况，在您的家中，某一个衣柜（具体工厂）只能存放某一种这样的衣服（成套，一系列具体产品），每次拿这种成套的衣服时也自然要从这个衣柜中取出了。用 OO 的思想去理解，所有的衣柜（具体工厂）都是衣柜类的（抽象工厂）某一个，而每一件成套的衣服又包括具体的上衣（某一具体产品），裤子（某一具体产品），这些具体的上衣其实也都是上衣（抽象产品），具体的裤子也都是裤子（另一个抽象产品）。

单例模式

单例模式（Singleton Pattern）是Java中最简单的设计模式之一。

这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。

保证一个类仅有一个实例，并提供一个访问它的全局访问点。

> 应用实例：一个班级只能有一个班主任。

建造者模式（构建者模式）

建造者模式（Builder Pattern）使用多个简单的对象一步一步构建成一个复杂的对象。

一个Builder类会一步一步构造最终的对象。该Builder类是独立于其他对象的。

将一个复杂的构建与其表示相分离，使得同样的构建过程可以创建不同的表示。

> 应用实例：
> 1、去肯德基，汉堡、可乐、薯条、炸鸡翅等是不变的，而其组合是经常变化的，生成出所谓的“套餐”；
> 2、Java 中的 StringBuilder。

原型模式

原型模式（Prototype Pattern）是用于创建重复的对象，同时又能保证性能。

这种模式是实现了一个原型接口，该接口用于创建当前对象的克隆。当直接创建对象的代价比较大时，则采用这种模式。例如，一个对象需要在一个高代价的数据库操作之后被创建。我们可以缓存该对象，在下一个请求时返回它的克隆，在需要的时候更新数据库，以此来减少数据库调用。

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。

> 应用实例：
> 1、细胞分裂；
> 2、Java中的 Object clone() 方法。

----------结构型模式------

适配器模式

适配器模式（Adapter Pattern）是作为两个不兼容的接口之间的桥梁。

这种模式涉及到一个单一的类，该类负责加入独立的或不兼容的接口功能。

将一个类的接口转换成客户希望的另外一个接口。适配器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。

> 应用实例：
> 1、读卡器是作为内存卡和笔记本之间的适配器。您将内存卡插入读卡器，再将读卡器插入笔记本，这样就可以通过笔记本来读取内存卡；
> 2、美国电器110V，中国220V，就要有一个变压器将110V转化为220V。

装饰器模式

装饰器模式（Decorator Pattern）允许向一个现有的对象添加新的功能，同时又不改变其结构。

这种模式创建了一个装饰类，用来包装原有的类，并在保持类方法签名完整性的前提下，提供了额外的功能。

动态地给一个对象添加一些额外的职责。就增加功能来说，装饰器模式相比生成子类更为灵活。

> 应用实例：
> 1、孙悟空有72变，当他变成"庙宇"后，他的根本还是一只猴子，但是他又有了庙宇的功能；
> 2、将一个形状装饰上不同的颜色，同时又不改变形状。

代理模式

在代理模式（Proxy Pattern）中，一个类代表另一个类的功能。

在代理模式中，我们创建具有现有对象的对象，以便向外界提供功能接口。

为其他对象提供一种代理以控制对这个对象的访问。

> 应用实例：
> 1、Windows里面的快捷方式；
> 2、买火车票不一定在火车站买，也可以去代售点；
> 3、一张支票或银行存单是账户中资金的代理。支票在市场交易中用来代替现金，并提供对签发人账号上资金的控制；
> 4、Spring AOP。

注意事项：

1、和适配器模式的区别：适配器模式主要改变所考虑对象的接口，而代理模式不能改变所代理类的接口。

2、和装饰器模式的区别：装饰器模式为了增强功能，而代理模式是为了加以控制。

外观模式

外观模式（Facade Pattern）隐藏系统的复杂性，并向客户端提供了一个客户端可以访问系统的接口。

这种模式涉及到一个单一的类，该类提供了客户端请求的简化方法和对现有系统类方法的委托调用。

为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

> 应用实例：
> 去医院看病，可能要去挂号、门诊、划价、取药，让患者或患者家属觉得很复杂，如果有提供接待人员，只让接待人员来处理，就很方便。

桥接模式

桥接模式（Bridge Pattern）是用于把抽象化与实现化解耦，使得二者可以独立变化。它通过提供抽象化和实现化之间的桥接结构，来实现二者的解耦。

这种模式涉及到一个作为桥接的接口，使得实体类的功能独立于接口实现类。这两种类型的类可被结构化改变而互不影响。

将抽象部分与实现部分分离，使它们都可以独立的变化。

又称为柄体（Handle and Body）模式或接口（Interface）模式。

> 应用实例：
> 1、猪八戒从天蓬元帅转世投胎到猪，转世投胎的机制将尘世划分为两个等级，即：灵魂和肉体，前者相当于抽象化，后者相当于实现化。生灵通过功能的委派，调用肉体对象的功能，使得生灵可以动态地选择；
> 2、墙上的开关，可以看到的开关是抽象的，不用管里面具体怎么实现的；
> 3、如果要绘制不同的颜色，如红色、绿色、蓝色的矩形、圆形、椭圆、正方形，我们需要根据实际需要对形状和颜色进行组合，那么颜色、形状就是抽象部分，组合后的就是实现部分。

注意事项：对于两个独立变化的维度，使用桥接模式再适合不过了。

组合模式

组合模式（Composite Pattern），又叫部分整体模式，是用于把一组相似的对象当作一个单一的对象。组合模式依据树形结构来组合对象，用来表示部分以及整体层次。这种类型的设计模式属于结构型模式，它创建了**对象组的树形结构**。

这种模式创建了一个包含自己对象组的类。该类提供了修改相同对象组的方式。

将对象组合成树形结构以表示"部分-整体"的层次结构。组合模式使得用户对单个对象和组合对象的使用具有一致性。

> 应用实例：
> 1、算术表达式包括操作数、操作符和另一个操作数，其中，另一个操作数也可以是操作数、操作符和另一个操作数。
> 2、在JAVAAWT和SWING中，对于Button和Checkbox是树叶，Container是树枝。

享元模式

享元模式（Flyweight Pattern）主要用于减少创建对象的数量，以减少内存占用和提高性能。这种类型的设计模式属于结构型模式，它提供了减少对象数量从而改善应用所需的对象结构的方式。

享元模式尝试重用现有的同类对象，如果未找到匹配的对象，则创建新对象。

运用共享技术有效地支持大量细粒度的对象。

> 应用实例：
> 1、Java中的String，如果有则返回，如果没有则创建一个字符串保存在字符串缓存池里面；
> 2、数据库的数据池。

----------行为型模式----------

策略模式

在策略模式（Strategy Pattern）中，一个类的行为或其算法可以在运行时更改。

在策略模式中，我们创建表示各种策略的对象和一个行为随着策略对象改变而改变的context对象。策略对象改变context对象的执行算法。

定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换。

> 应用实例：
> 1、诸葛亮的锦囊妙计，每一个锦囊就是一个策略；
> 2、旅行的出游方式，选择骑自行车、坐汽车，每一种旅行方式都是一个策略。

模板模式

在模板模式（Template Pattern）中，一个抽象类公开定义了执行它的方法的方式/模板。它的子类可以按需要重写方法实现，但调用将以抽象类中定义的方式进行。

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。

> 应用实例：
> 1、在造房子的时候，地基、走线、水管都一样，只有在建筑的后期才有加壁橱加栅栏等差异；
> 2、西游记里面菩萨定好的81难，这就是一个顶层的逻辑骨架；
> 3、spring中对Hibernate的支持，将一些已经定好的方法封装起来，比如开启事务、获取Session、关闭Session等，程序员不重复写那些已经规范好的代码，直接丢一个实体就可以保存。

观察者模式

当对象间存在一对多关系时，则使用观察者模式（ObserverPattern）。比如，当一个对象被修改时，则会自动通知它的依赖对象。

定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

> 应用实例：
> 1、拍卖的时候，拍卖师观察最高标价，然后通知给其他竞价者竞价；
> 2、西游记里面悟空请求菩萨降服红孩儿，菩萨洒了一地水招来一个老乌龟，这个乌龟就是观察者，他观察菩萨洒水这个动作。

迭代器模式

迭代器模式（Iterator Pattern）是Java和.Net编程环境中非常常用的设计模式。这种模式用于顺序访问集合对象的元素，不需要知道集合对象的底层表示。

迭代器模式属于行为型模式。

提供一种方法顺序访问一个聚合对象中各个元素，而又无须暴露该对象的内部表示。

> 应用实例：JAVA中的iterator。

责任链模式

顾名思义，责任链模式（Chain of Responsibility Pattern）为请求创建了一个接收者对象的链。这种模式给予请求的类型，对请求的发送者和接收者进行解耦。

在这种模式中，通常每个接收者都包含对另一个接收者的引用。如果一个对象不能处理该请求，那么它会把相同的请求传给下一个接收者，依此类推。

避免请求发送者与接收者耦合在一起，让多个对象都有可能接收请求，将这些对象连接成一条链，并且沿着这条链传递请求，直到有对象处理它为止。

> 应用实例：红楼梦中的"击鼓传花"。

命令模式

命令模式（Command Pattern）是一种数据驱动的设计模式。请求以命令的形式包裹在对象中，并传给调用对象。调用对象寻找可以处理该命令的合适的对象，并把该命令传给相应的对象，该对象执行命令。

将一个请求封装成一个对象，从而使您可以用不同的请求对客户进行参数化。

> 应用实例：电视机是请求的接收者，遥控器是请求的发送者，遥控器上有一些按钮，不同的按钮对应电视机的不同操作。抽象命令角色由一个命令接口来扮演，有三个具体的命令类实现了抽象命令接口，这三个具体命令类分别代表三种操作：打开电视机、关闭电视机和切换频道。

备忘录模式

备忘录模式（Memento Pattern）保存一个对象的某个状态，以便在适当的时候恢复对象。

在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。

> 应用实例：
> 1、后悔药；
> 2、打游戏时的存档；
> 3、Windows里的ctri+z；
> 4、IE中的后退；
> 5、数据库的事务管理。

状态模式

在状态模式（State Pattern）中，类的行为是基于它的状态改变的。

在状态模式中，我们创建表示各种状态的对象和一个行为随着状态对象改变而改变的context对象。

允许对象在内部状态发生改变时改变它的行为，对象看起来好像修改了它的类。

> 应用实例：
> 1、打篮球的时候运动员可以有正常状态、不正常状态和超常状态；
> 2、曾侯乙编钟中，'钟是抽象接口'，'钟A'等是具体状态，'曾侯乙编钟'是具体环境（Context）。

访问者模式

在访问者模式（Visitor Pattern）中，我们使用了一个访问者类，它改变了元素类的执行算法。通过这种方式，元素的执行算法可以随着访问者改变而改变。根据模式，元素对象已接受访问者对象，这样访问者对象就可以处理元素对象上的操作。

主要将数据结构与数据操作分离。

主要解决：稳定的数据结构和易变的操作耦合问题。

> 应用实例：您在朋友家做客，您是访问者，朋友接受您的访问，您通过朋友的描述，然后对朋友的描述做出一个判断，这就是访问者模式。

中介者模式

中介者模式（Mediator Pattern）是用来降低多个对象和类之间的通信复杂性。这种模式提供了一个中介类，该类通常处理不同类之间的通信，并支持松耦合，使代码易于维护。

用一个中介对象来封装一系列的对象交互，中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。

> 应用实例：
> 1、中国加入WTO之前是各个国家相互贸易，结构复杂，现在是各个国家通过WTO来互相贸易；
> 2、机场调度系统；

> 3、MVC框架，其中C（控制器）就是M（模型）和V（视图）的中介者。

解释器模式

解释器模式（Interpreter Pattern）提供了评估语言的语法或表达式的方式。这种模式实现了一个表达式接口，该接口解释一个特定的上下文。这种模式被用在SQL解析、符号处理引擎等。

给定一个语言，定义它的文法表示，并定义一个解释器，这个解释器使用该标识来解释语言中的句子。

> 应用实例：编译器、运算表达式计算。

思考题：

设计模式其实还有两类：并发型模式、线程池模式，它们分别是什么？

原文及参考资料：

六大设计原则(SOLID)

23种设计模式汇总整理

图解23种设计模式

23种设计模式及案例

23种设计模式的应用场景分别是哪些？

如有错误，请更正指出，谢谢！