第二章 GoF 的 23 种设计模式的分类和功能

设计模式有两种分类方法，即根据模式的目的来分和根据模式的作用的范围来分。

1 根据目的来分

根据模式是用来完成什么工作来划分，这种方式可分为创建型模式、结构型模式和行为型模式 3 种。

1 .1 创建型模式

用于描述“怎样创建对象”，它的主要特点是“将对象的创建与使用分离”。GoF 中提供了单例、原型、工厂方法、抽象工厂、建造者等 5 种创建型模式。

1 .2 结构型模式

用于描述如何将类或对象按某种布局组成更大的结构，GoF 中提供了代理、适配器、桥接、装饰、外观、享元、组合等 7 种结构型模式。

1 .3 行为型模式

用于描述类或对象之间怎样相互协作共同完成单个对象都无法单独完成的任务，以及怎样分配职责。GoF 中提供了模板方法、策略、命令、职责链、状态、观察者、中介者、迭代器、访问者、备忘录、解释器等 11 种行为型模式。

2 根据作用范围来分

根据模式是主要用于类上还是主要用于对象上来分，这种方式可分为类模式和对象模式两种。

2 .1类模式

用于处理类与子类之间的关系，这些关系通过继承来建立，是静态的，在编译时刻便确定下来了。GoF 中的工厂方法、（**类**）适配器、模板方法、解释器属于该模式。

2 .2对象模式

用于处理对象之间的关系，这些关系可以通过组合或聚合来实现，在运行时刻是可以变化的，更具动态性。GoF 中除了以上 4 种，其他的都是对象模式。

表 1 介绍了这 23 种设计模式的分类。

表 1GoF 的 23 种设计模式的分类表

范围\目的	创建型模式	结构型模式	行为型模式
类模式	工厂方法	(类）适配器	模板方法、解释器
对象模式	单例 原型 抽象工厂 建造者	代理 (对象）适配器 桥接 装饰 外观 享元 组合	策略 命令 职责链 状态 观察者 中介者 迭代器 访问者 备忘录

3 GoF 的 23 种设计模式的功能

前面说明了 GoF 的 23 种设计模式的分类，现在对各个模式的功能进行介绍。

3 .1 单例（Singleton）模式

某个类只能生成一个实例，该类提供了一个全局访问点供外部获取该实例，其拓展是有限多例模式。

单例模式确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例单例模式。单例模式只应在有真正的“单一实例”的需求时才可使用。

举例：韦小宝有 7 个漂亮的老婆，她们的老公都是韦小宝，韦小宝就是他们家里的老公 Singleton，她们只要说道“老公”，都是指的同一个人，那就是韦小宝。

3 .2 原型（Prototype）模式

将一个对象作为原型，通过对其进行复制而克隆出多个和原型类似的新实例。

通过给出一个原型对象来指明所要创建的对象的类型，然后用复制这个原型对象的方法创建出更多同类型的对象。原始模型模式允许动态的增加或减少产品类，产品类不需要非得有任何事先确定的等级结构，原始模型模式适用于任何的等级结构。缺点是每一个类都必须配备一个克隆方法。

举例：跟 MM 用 QQ 聊天，一定要说些深情的话语了，我搜集了好多肉麻的情话，需要时只要 copy 出来放到 QQ 里面就行了，这就是我的情话 prototype 了。

3 .3 工厂方法（Factory Method）模式

定义一个用于创建产品的接口，由子类决定生产什么产品。

核心工厂类不再负责所有产品的创建，而是将具体创建的工作交给子类去做，成为一个抽象工厂角色，仅负责给出具体工厂类必须实现的接口，而不接触哪一个产品类应当被实例化这种细节。

举例：追 MM 少不了请吃饭了，麦当劳的鸡翅和肯德基的鸡翅都是 MM 爱吃的东西，虽然口味有所不同，但不管你带 MM 去麦当劳或肯德基，只管向服务员说“来四个鸡翅”就行了。麦当劳和肯德基就是生产鸡翅的 Factory 工厂模式：客户类和工厂类分开。消费者任何时候需要某种产品，只需向工厂请求即可。消费者无须修改就可以接纳新产品。缺点是当产品修改时，工厂类也要做相应的修改。如：如何创建及如何向客户端提供。

3 .4 抽象工厂（AbstractFactory）模式

提供一个创建产品族的接口，其每个子类可以生产一系列相关的产品。

客户类和工厂类分开。消费者任何时候需要某套产品集合时，只需向抽象工厂请求即可。抽象工厂会再向具体的工厂生产出符合产品集规格的产品.

举例：追 MM 少不了请吃饭了，麦当劳的套餐和肯德基的套餐都是 MM 爱吃的东西，虽然口味有所不同，但不管你带 MM 去麦当劳或肯德基，只管向服务员说“两个 B 套餐”就行了。麦当劳和肯德基就是 B 套餐的 Abstract Factory, B 套餐里含有汉堡，鸡翅和饮料. 麦当劳或肯德基会根据 B 套餐的规格，让汉堡 Factory，鸡翅 Factory，饮料 Factory 分别生产对应 B 套餐的材料.

3 .5 建造者（Builder）模式

将一个复杂对象分解成多个相对简单的部分，然后根据不同需要分别创建它们，最后构建成该复杂对象。

将产品的内部表象和产品的生成过程分割开来，从而使一个建造过程生成具有不同的内部表象的产品对象。建造模式使得产品内部表象可以独立的变化，客户不必知道产品内部组成的细节。建造模式可以强制实行一种分步骤进行的建造过程。

举例：MM 最爱听的就是“我爱你”这句话了，见到不同地方的 MM,要能够用她们的方言跟她说这句话哦，我有一个多种语言翻译机，上面每种语言都有一个按键，见到 MM 我只要按对应的键，它就能够用相应的语言说出“我爱你”这句话了，国外的 MM 也可以轻松搞掂，这就是我的“我爱你”builder。（这一定比美军在伊拉克用的翻译机好卖）

3 .6 代理（Proxy）模式

为某对象提供一种代理以控制对该对象的访问。即客户端通过代理间接地访问该对象，从而限制、增强或修改该对象的一些特性。

代理模式给某一个对象提供一个代理对象，并由代理对象控制对源对象的引用。代理就是一个人或一个机构代表另一个人或者一个机构采取行动。某些情况下，客户不想或者不能够直接引用一个对象，代理对象可以在客户和目标对象直接起到中介的作用。客户端分辨不出代理主题对象与真实主题对象。代理模式可以并不知道真正的被代理对象，而仅仅持有一个被代理对象的接口，这时候代理对象不能够创建被代理对象，被代理对象必须有系统的其

他角色代为创建并传入。

举例：跟 MM 在网上聊天，一开头总是“hi,你好”，“你从哪儿来呀？”“你多大了？”“身高多少呀？”这些话，真烦人，写个程序做为我的 Proxy 吧，凡是接收到这些话都设置好了自动的回答，接收到其他的话时再通知我回答，怎么样，酷吧。

3 .7 适配器（Adapter）模式

将一个类的接口转换成客户希望的另外一个接口，使得原本由于接口不兼容而不能一起工作的那些类能一起工作。

适配器（变压器）模式：把一个类的接口变换成客户端所期待的另一种接口，从而使原本因接口原因不匹配而无法一起工作的两个类能够一起工作。适配类可以根据参数返还一个合适的实例给客户端。

举例：在朋友聚会上碰到了一个美女 Sarah，从香港来的，可我不会说粤语，她不会说普通话，只好求助于我的朋友 kent 了，他作为我和 Sarah 之间的 Adapter，让我和 Sarah 可以相互交谈了(也不知道他会不会耍我)

3 .8 桥接（Bridge）模式

将抽象与实现分离，使它们可以独立变化。它是用组合关系代替继承关系来实现，从而降低了抽象和实现这两个可变维度的耦合度。

桥梁模式：将抽象化与实现化脱耦，使得二者可以独立的变化，将他们之间的强关联变成弱关联，也就是指在一个软件系统的抽象化和实现化之间使用组合/聚合关系而不是继承关系，从而使两者可以独立的变化。

举例：早上碰到 MM，要说早上好，晚上碰到 MM，要说晚上好；碰到 MM 穿了件新衣服，要说你的衣服好漂亮哦，碰到 MM 新做的发型，要说你的头发好漂亮哦。不要问我“早上碰到 MM 新做了个发型怎么说”这种问题，自己用 BRIDGE 组合一下不就行了

3 .9 装饰（Decorator）模式

动态的给对象增加一些职责，即增加其额外的功能。

装饰模式以对客户端透明的方式扩展对象的功能，是继承关系的一个替代方案，提供比

继承更多的灵活性。动态给一个对象增加功能，这些功能可以再动态的撤消。增加由一些基本功能的排列组合而产生的非常大量的功能。

举例：Mary 过完轮到 Sarly 过生日，还是不要叫她自己挑了，不然这个月伙食费肯定玩完，拿出我去年在华山顶上照的照片，在背面写上“最好的礼物，就是爱你的隔壁老王”，再到街上礼品店买了个像框（卖礼品的 MM 也很漂亮哦），再找隔壁搞美术设计的 Mike 设计了一个漂亮的盒子装起来……，我们都是 Decorator，最终都在修饰我这个人呀，怎么样，看懂了吗?

3 .10 外观（Facade）模式

为多个复杂的子系统提供一个一致的接口，使这些子系统更加容易被访问。

外观模式又叫门面模式：外部与一个子系统的通信必须通过一个统一的门面对象进行。门面模式提供一个高层次的接口，使得子系统更易于使用。每一个子系统只有一个门面类，而且此门面类只有一个实例，也就是说它是一个单例模式。但整个系统可以有多个门面类。

举例：我有一个专业的单反相机，我就喜欢自己手动调光圈、快门，这样照出来的照片才专业，但 MM 可不懂这些，教了半天也不会。幸好相机有 Facade 设计模式，把相机调整到自动档，只要对准目标按快门就行了，一切由相机自动调整，这样 MM 也可以用这个相机给我拍张照片了。

3 .11 享元（Flyweight）模式

运用共享技术来有效地支持大量细粒度对象的复用。

FLYWEIGHT 在拳击比赛中指最轻量级。享元模式以共享的方式高效的支持大量的细粒度对象。享元模式能做到共享的关键是区分内蕴状态和外蕴状态。内蕴状态存储在享元内部，不会随环境的改变而有所不同。外蕴状态是随环境的改变而改变的。外蕴状态不能影响内蕴状态，它们是相互独立的。将可以共享的状态和不可以共享的状态从常规类中区分开来，将不可以共享的状态从类里剔除出去。客户端不可以直接创建被共享的对象，而应当使用一个工厂对象负责创建被共享的对象。享元模式大幅度的降低内存中对象的数量。

举例：每天跟 MM 发短信，手指都累死了，最近买了个新手机，可以把一些常用的句

子存在手机里，要用的时候，直接拿出来，在前面加上 MM 的名字就可以发送了，再不用一个字一个字敲了。共享的句子就是 Flyweight，MM 的名字就是提取出来的外部特征，根据上下文情况使用。

3 .12 组合（Composite）模式

将对象组合成树状层次结构，使用户对单个对象和组合对象具有一致的访问性。

组合模式又叫合成模式：合成模式将对象组织到树结构中，可以用来描述整体与部分的关系。合成模式就是一个处理对象的树结构的模式。合成模式把部分与整体的关系用树结构表示出来。合成模式使得客户端把一个个单独的成分对象和由他们复合而成的合成对象同等看待。

举例：Mary 今天过生日。“我过生日，你要送我一件礼物。”“嗯，好吧，去商店，你自己挑。”“这件 T 恤挺漂亮，买，这条裙子好看，买，这个包也不错，买。”“喂，买了三件了呀，我只答应送一件礼物的哦。”“什么呀，T 恤加裙子加包包，正好配成一套呀，小姐，麻烦你包起来。”“……”，MM 都会用 Composite 模式了，你会了没有?

3 .13 模板方法（TemplateMethod）模式

定义一个操作中的算法骨架，而将算法的一些步骤延迟到子类中，使得子类可以不改变该算法结构的情况下重定义该算法的某些特定步骤。

模板方法模式准备一个抽象类，将部分逻辑以具体方法以及具体构造子的形式实现，然后声明一些抽象方法来迫使子类实现剩余的逻辑。不同的子类可以以不同的方式实现这些抽象方法，从而对剩余的逻辑有不同的实现。先制定一个顶级逻辑框架，而将逻辑的细节留给具体的子类去实现。

举例：看过《如何说服女生上床》这部经典文章吗？女生从认识到上床的不变的步骤分为巧遇、打破僵局、展开追求、接吻、前戏、动手、爱抚、生米煮成熟饭八大步骤(Template method)，但每个步骤针对不同的情况，都有不一样的做法，这就要看你随机应变啦(具体实现)。

3 .14 策略（Strategy）模式

定义了一系列算法，并将每个算法封装起来，使它们可以相互替换，且算法的改变不会影响使用算法的客户。

策略模式针对一组算法，将每一个算法封装到具有共同接口的独立的类中，从而使得它们可以相互替换。策略模式使得算法可以在不影响到客户端的情况下发生变化。策略模式把行为和环境分开。环境类负责维持和查询行为类，各种算法在具体的策略类中提供。由于算法和环境独立开来，算法的增减，修改都不会影响到环境和客户端。

举例：跟不同类型的 MM 约会，要用不同的策略，有的请电影比较好，有的则去吃小吃效果不错，有的去海边浪漫最合适，但目的都是为了得到 MM 的芳心，我的追 MM 锦囊中有好多 Strategy 哦。

3 .15 命令（Command）模式

将一个请求封装为一个对象，使发出请求的责任和执行请求的责任分割开。

命令模式：命令模式把一个请求或者操作封装到一个对象中。命令模式把发出命令的责任和执行命令的责任分割开，委派给不同的对象。命令模式允许请求的一方和发送的一方独立开来，使得请求的一方不必知道接收请求的一方的接口，更不必知道请求是怎么被接收，以及操作是否执行，何时被执行以及是怎么被执行的。

举例：俺有一个 MM 家里管得特别严，没法见面，只好借助于她弟弟在我们俩之间传送信息，她对我有什么指示，就写一张纸条让她弟弟带给我。这不，她弟弟又传送过来一个 COMMAND，为了感谢他，我请他吃了碗杂酱面，哪知道他说：“我同时给我姐姐三个男朋友送 COMMAND，就数你最小气，才请我吃面。”，:-(

3 .16 职责链（Chain of Responsibility）模式

把请求从链中的一个对象传到下一个对象，直到请求被响应为止。通过这种方式去除对象之间的耦合。

责任链模式：在责任链模式中，很多对象由每一个对象对其下家的引用而接起来形成一条链。请求在这个链上传递，直到链上的某一个对象决定处理此请求。客户并不知道链上的

哪一个对象最终处理这个请求，系统可以在不影响客户端的情况下动态的重新组织链和分配责任。处理者有两个选择：承担责任或者把责任推给下家。一个请求可以最终不被任何接收端对象所接受。

举例：晚上去上英语课，为了好开溜坐到了最后一排，哇，前面坐了好几个漂亮的 MM 哎，找张纸条，写上“Hi,可以做我的女朋友吗？如果不愿意请向前传”，纸条就一个接一个的传上去了，糟糕，传到第一排的 MM 把纸条传给老师了，听说是个老处女呀，快跑!

3 .17 状态（State）模式

允许一个对象在其内部状态发生改变时改变其行为能力。

状态模式允许一个对象在其内部状态改变的时候改变行为。这个对象看上去象是改变了它的类一样。状态模式把所研究的对象的行为包装在不同的状态对象里，每一个状态对象都属于一个抽象状态类的一个子类。状态模式的意图是让一个对象在其内部状态改变的时候，其行为也随之改变。状态模式需要对每一个系统可能取得的状态创立一个状态类的子类。当系统的状态变化时，系统便改变所选的子类。

举例：跟 MM 交往时，一定要注意她的状态哦，在不同的状态时她的行为会有不同，比如你约她今天晚上去看电影，对你没兴趣的 MM 就会说“有事情啦”，对你不讨厌但还没喜欢上的 MM 就会说“好啊，不过可以带上我同事么？”，已经喜欢上你的 MM 就会说“几点钟？看完电影再去泡吧怎么样？”，当然你看电影过程中表现良好的话，也可以把 MM 的状态从不讨厌不喜欢变成喜欢哦。

3 .18 观察者（Observer）模式

多个对象间存在一对多关系，当一个对象发生改变时，把这种改变通知给其他多个对象，从而影响其他对象的行为。

观察者模式：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上发生变化时，会通知所有观察者对象，使他们能够自动更新自己。

举例：想知道咱们公司最新 MM 情报吗？加入公司的 MM 情报邮件组就行了，隔壁老

王负责搜集情报，他发现的新情报不用一个一个通知我们，直接发布给邮件组，我们作为订阅者（观察者）就可以及时收到情报啦。

3 .19 中介者（Mediator）模式

定义一个中介对象来简化原有对象之间的交互关系，降低系统中对象间的耦合度，使原有对象之间不必相互了解。

又名调停者模式：调停者模式包装了一系列对象相互作用的方式，使得这些对象不必相互明显作用。从而使他们可以松散耦合。当某些对象之间的作用发生改变时，不会立即影响其他的一些对象之间的作用。保证这些作用可以彼此独立的变化。调停者模式将多对多的相互作用转化为一对多的相互作用。调停者模式将对象的行为和协作抽象化，把对象在小尺度的行为上与其他对象的相互作用分开处理。

举例：四个 MM 打麻将，相互之间谁应该给谁多少钱算不清楚了，幸亏当时我在旁边，按照各自的筹码数算钱，赚了钱的从我这里拿，赔了钱的也付给我，一切就 OK 啦，俺得到了四个 MM 的电话。

3 .20 迭代器（Iterator）模式

提供一种方法来顺序访问聚合对象中的一系列数据，而不暴露聚合对象的内部表示。

又名迭代子模式：迭代子模式可以顺序访问一个聚集中的元素而不必暴露聚集的内部表象。多个对象聚在一起形成的总体称之为聚集，聚集对象是能够包容一组对象的容器对象。迭代子模式将迭代逻辑封装到一个独立的子对象中，从而与聚集本身隔开。迭代子模式简化了聚集的界面。每一个聚集对象都可以有一个或一个以上的迭代子对象，每一个迭代子的迭代状态可以是彼此独立的。迭代算法可以独立于聚集角色变化。

举例：我爱上了 Mary，不顾一切的向她求婚。

Mary：“想要我跟你结婚，得答应我的条件”

我：“什么条件我都答应，你说吧”

Mary：“我看上了那个一克拉的钻石”

我：“我买，我买，还有吗？”

Mary： “我看上了湖边的那栋别墅”

我： “我买，我买，还有吗？”

……

3 .21 访问者（Visitor）模式

在不改变集合元素的前提下，为一个集合中的每个元素提供多种访问方式，即每个元素有多个访问者对象访问。

访问者模式的目的是封装一些施加于某种数据结构元素之上的操作。一旦这些操作需要修改的话，接受这个操作的数据结构可以保持不变。访问者模式适用于数据结构相对未定的系统，它把数据结构和作用于结构上的操作之间的耦合解脱开，使得操作集合可以相对自由的演化。访问者模式使得增加新的操作变的很容易，就是增加一个新的访问者类。访问者模式将有关的行为集中到一个访问者对象中，而不是分散到一个个的节点类中。当使用访问者模式时，要将尽可能多的对象浏览逻辑放在访问者类中，而不是放到它的子类中。访问者模式可以跨过几个类的等级结构访问属于不同的等级结构的成员类。

举例：情人节到了，要给每个 MM 送一束鲜花和一张卡片，可是每个 MM 送的花都要针对她个人的特点，每张卡片也要根据个人的特点来挑，我一个人哪搞得清楚，还是找花店老板和礼品店老板做一下 Visitor，让花店老板根据 MM 的特点选一束花，让礼品店老板也根据每个人的特点选一张卡，这样就轻松多了。

3 .22 备忘录（Memento）模式

在不破坏封装性的前提下，获取并保存一个对象的内部状态，以便以后恢复它。

备忘录对象是一个用来存储另外一个对象内部状态的快照的对象。备忘录模式的用意是在不破坏封装的条件下，将一个对象的状态捉住，并外部化，存储起来，从而可以在将来合适的时候把这个对象还原到存储起来的状态。

举例：同时跟几个 MM 聊天时，一定要记清楚刚才跟 MM 说了些什么话，不然 MM

发现了会不高兴的哦，幸亏我有个备忘录，刚才与哪个 MM 说了什么话我都拷贝一份放到备忘录里面保存，这样可以随时察看以前的记录啦。

3 .23 解释器（Interpreter）模式

提供如何定义语言的文法，以及对语言句子的解释方法，即解释器。

给定一个语言后，解释器模式可以定义出其文法的一种表示，并同时提供一个解释器。客户端可以使用这个解释器来解释这个语言中的句子。解释器模式将描述怎样在有了一个简单的文法后，使用模式设计解释这些语句。在解释器模式里面提到的语言是指任何解释器对象能够解释的任何组合。在解释器模式中需要定义一个代表文法的命令类的等级结构，也就是一系列的组合规则。每一个命令对象都有一个解释方法，代表对命令对象的解释。命令对象的等级结构中的对象的任何排列组合都是一个语言。

举例：俺有一个《泡 MM 真经》，上面有各种泡 MM 的攻略，比如说去吃西餐的步骤、去看电影的方法等等，跟 MM 约会时，只要做一个 Interpreter，照着上面的脚本执行就可以了。

必须指出，这 23 种设计模式不是孤立存在的，很多模式之间存在一定的关联关系，在大的系统开发中常常同时使用多种设计模式。