设计模式

常见设计模式 Java GOF

设计模式
设计模式（Design pattern）代表了最佳的实践，通常被有经验的面向对象的软件开发人员所采用。设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案。这些解决方案是众多软件开发人员经过相当长的一段时间的试验和错误总结出来的。
本教程将通过 Java 实例，一步一步向您讲解设计模式的概念。

设计模式简介
设计模式（Design pattern）代表了最佳的实践，通常被有经验的面向对象的软件开发人员所采用。设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案。这些解决方案是众多软件开发人员经过相当长的一段时间的试验和错误总结出来的。
设计模式是一套被反复使用的、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了重用代码、让代码更容易被他人理解、保证代码可靠性。 毫无疑问，设计模式于己于他人于系统都是多赢的，设计模式使代码编制真正工程化，设计模式是软件工程的基石，如同大厦的一块块砖石一样。项目中合理地运用设计模式可以完美地解决很多问题，每种模式在现实中都有相应的原理来与之对应，每种模式都描述了一个在我们周围不断重复发生的问题，以及该问题的核心解决方案，这也是设计模式能被广泛应用的原因。

设计模式的使用
设计模式在软件开发中的两个主要用途。

Markdown 语法

- **开发人员的共同平台**
  设计模式提供了一个标准的术语系统，且具体到特定的情景。例如，单例设计模式意味着使用单个对象，这样所有熟悉单例设计模式的开发人员都能使用单个对象，并且可以通过这种方式告诉对方，程序使用的是单例模式。

- **最佳的实践**
  设计模式已经经历了很长一段时间的发展，它们提供了软件开发过程中面临的一般问题的最佳解决方案。学习这些模式有助于经验不足的开发人员通过一种简单快捷的方式来学习软件设计

设计模式的类型
根据设计模式的参考书 Design Patterns - Elements of Reusable Object-Oriented Software（中文译名：设计模式 - 可复用的面向对象软件元素） 中所提到的，总共有 23 种设计模式。这些模式可以分为三大类：创建型模式（Creational Patterns）、结构型模式（Structural Patterns）、行为型模式（Behavioral Patterns）。当然，我们还会讨论另一类设计模式：J2EE 设计模式。

序号	模式 & 描述	包括
1	**创建型模式** 这些设计模式提供了一种在创建对象的同时隐藏创建逻辑的方式，而不是使用新的运算符直接实例化对象。这使得程序在判断针对某个给定实例需要创建哪些对象时更加灵活。	• 工厂模式（Factory Pattern） • 抽象工厂模式（Abstract Factory Pattern） • 单例模式（Singleton Pattern） • 建造者模式（Builder Pattern） • 原型模式（Prototype Pattern）
2	**结构型模式** 这些设计模式关注类和对象的组合。继承的概念被用来组合接口和定义组合对象获得新功能的方式。	• 适配器模式（Adapter Pattern） • 桥接模式（Bridge Pattern） • 过滤器模式（Filter、Criteria Pattern） • 组合模式（Composite Pattern） • 装饰器模式（Decorator Pattern） • 外观模式（Facade Pattern） • 享元模式（Flyweight Pattern） • 代理模式（Proxy Pattern）
3	**行为型模式** 这些设计模式特别关注对象之间的通信。	• 责任链模式（Chain of Responsibility Pattern） • 命令模式（Command Pattern） • 解释器模式（Interpreter Pattern） • 迭代器模式（Iterator Pattern） • 中介者模式（Mediator Pattern） • 备忘录模式（Memento Pattern） • 观察者模式（Observer Pattern） • 状态模式（State Pattern） • 空对象模式（Null Object Pattern） • 策略模式（Strategy Pattern） • 模板模式（Template Pattern） • 访问者模式（Visitor Pattern）
4	**J2EE 模式** 这些设计模式特别关注表示层。这些模式是由 Sun Java Center 鉴定的。	• MVC 模式（MVC Pattern） • 业务代表模式（Business Delegate Pattern） • 组合实体模式（Composite Entity Pattern） • 数据访问对象模式（Data Access Object Pattern） • 前端控制器模式（Front Controller Pattern） • 拦截过滤器模式（Intercepting Filter Pattern） • 服务定位器模式（Service Locator Pattern） • 传输对象模式（Transfer Object Pattern）

下面用一个图片来整体描述一下设计模式之间的关系：

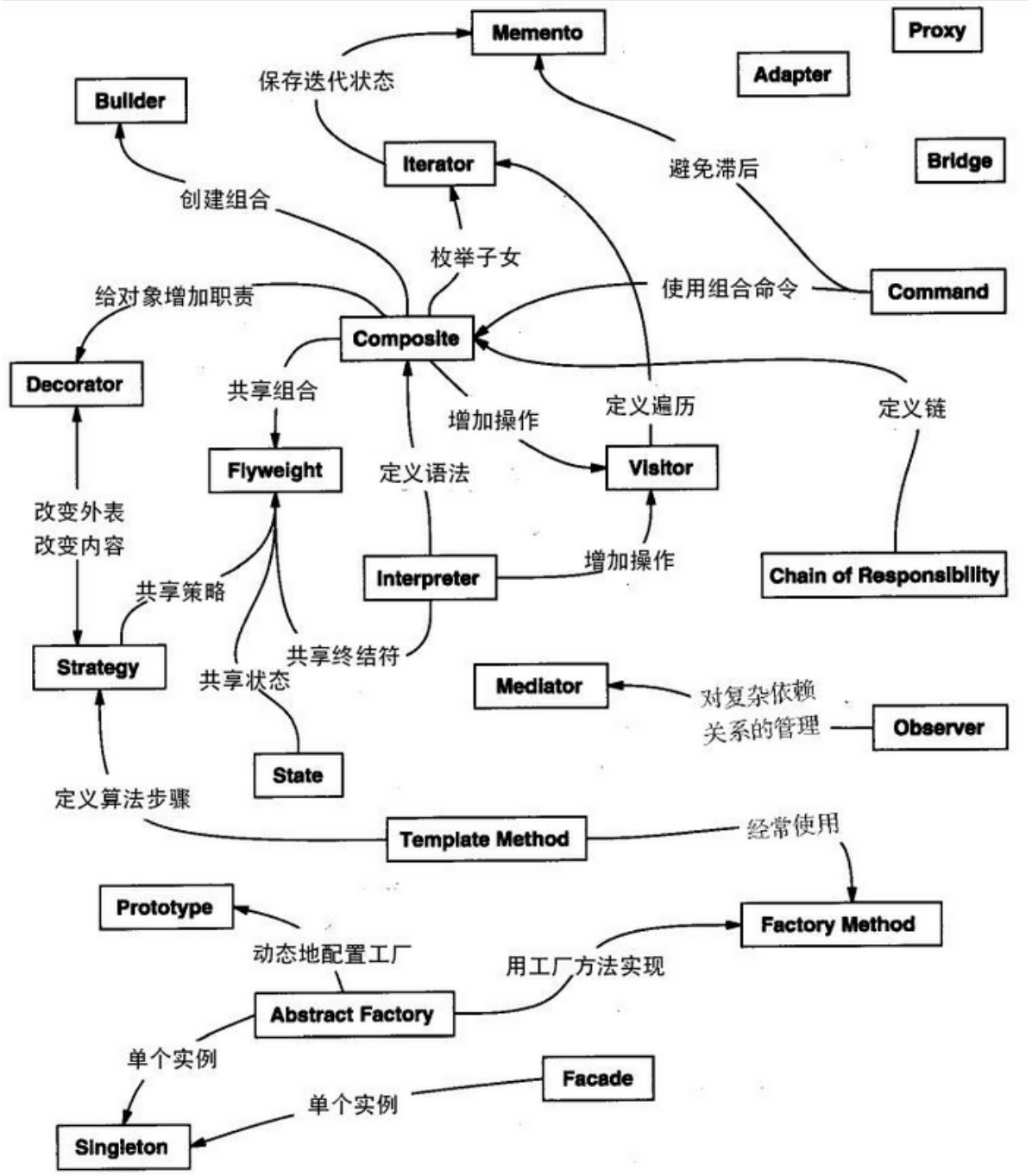

图　设计模式之间的关系

设计模式的六大原则

1、开闭原则（Open Close Principle）

开闭原则的意思是：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，实现一个热插拔的效果。简言之，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，后面的具体设计中我们会提到这点。

2、里氏代换原则（Liskov Substitution Principle）

里氏代换原则是面向对象设计的基本原则之一。 里氏代换原则中说，任何基类可以出现的地方，子类一定可以出现。LSP 是继承复用的基石，只有当派生类可以替换掉基类，且软件单位的功能不受到影响时，基类才能真正被复用，而派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，所以里氏代换原则是对实现抽象化的具体步骤的规范。

3、依赖倒转原则（Dependence Inversion Principle）

这个原则是开闭原则的基础，具体内容：针对对接口编程，依赖于抽象而不依赖于具体。

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

这个原则的意思是：使用多个隔离的接口，比使用单个接口要好。它还有另外一个意思是：降低类之间的耦合度。由此可见，其实设计模式就是从大型软件架构出发、便于升级和维护的软件设计思想，它强调降低依赖，降低耦合。

5、迪米特法则，又称最少知道原则（Demeter Principle）

最少知道原则是指：一个实体应当尽量少地与其他实体之间发生相互作用，使得系统功能模块相对独立。

6、合成复用原则（Composite Reuse Principle）

合成复用原则是指：尽量使用合成/聚合的方式，而不是使用继承。

Android进阶之大话设计模式

常用的设计模式有以下八种：单例、工厂、观察者、代理、命令、适配器、合成、访问者

单例模式：目的是为了让系统中只有一个调用对象，缺点是单例使其他程序过分依赖它，而且不同单例运行在不同进程中，使得维护困难；

工厂模式：生产固定的一些东西，如抽象类，缺点是产品修改麻烦；如喜欢动作片和爱情片的人分别向服务器发出同一个请求，就可以得到他们想看的影片集，相当于不同对象进行同一请求，需求均得到满足。

观察者模式：就是多个对象对一个对象进行监控，如缓存；

代理模式：自己的事交给别人去做，分别返回结果即可，如异步线程；

命令模式：调用对象与作用对象之间分离，由中间件来协调两者之间的工作，如控制器；

适配器模式：将一个接口变成用户所需要的接口，如baseadapter可以适配listview和spinner，因为它们有相同的接口

合成模式：将一对多的关系转换成一对整体的关系，如listview与适配器；

访问者模式：对不同的对象采取不同的处理，如instanceof。

创建型模式

1、FACTORY—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。

2、BUILDER—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）

建造模式：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。

3、FACTORY METHOD—请MM去麦当劳吃汉堡，不同的MM有不同的口味，要每个都记住是一件烦人的事情，我一般采用Factory Method模式，带着MM到服务员那儿，说“要一个汉堡”，具体要什么样的汉堡呢，让MM直接跟服务员说就行了。

工厂方法模式：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。

4、PROTOTYPE—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。（100块钱一份，你要不要）

原始模型模式：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。

5、SINGLETON—俺有6个漂亮的老婆，她们的老公都是我，我就是我们家里的老公Sigleton，她们只要说道“老公”，都是指的同一个人，那就是我(刚才做了个梦啦，哪有这么好的事)

单例模式：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。

结构型模式

6、ADAPTER—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)

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。

7、BRIDGE—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

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，也就是说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。

8、COMPOSITE—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，MM，麻烦你包起来。”“……”，MM都会用Composite模式了，你会了没有？

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。

9、DECORATOR—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“最好的的礼物，就是爱你的Fita”，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？

装饰模式：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。

10、FACADE—我有一个专业的Nikon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。

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。

11、FLYWEIGHT—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。

享元模式：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。

12、PROXY—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。

代理模式：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。

行为模式

13、CHAIN OF RESPONSIBLEITY—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!

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。

14、COMMAND—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。”，:-(

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。系统支持命令的撤消。

15、INTERPRETER—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。

解释器模式：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。

16、ITERATOR—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？”

Mary：“你的小弟弟必须要有50cm长”

我脑袋嗡的一声，坐在椅子上，一咬牙：“我剪，我剪，还有吗？”

……

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。

17、MEDIATOR—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。

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偶合。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。

18、MEMENTO—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。

备忘录模式：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。

19、OBSERVER—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，tom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

观察者模式：观察者模式定义了一种一队多的依赖关系，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。

20、STATE—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。

状态模式：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。

21、STRATEGY—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，单目的都是为了得到MM的芳心，我的追MM锦囊中有好多Strategy哦。

策略模式：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。

22、TEMPLATE METHOD——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、进去八大步骤(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)；

模板方法模式：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。

23、VISITOR—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特点选一张卡，这样就轻松多了；

访问者模式：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。