**创建模式：**对类的实例化过程的抽象。一些系统在创建对象时，需要动态地决定怎样创建对象，创建哪些对象，以及如何组合和表示这些对象。创建模式描述了怎样构造和封装这些动态的决定。包含类的创建模式和对象的创建模式。

**结构模式：**描述如何将类或对象结合在一起形成更大的结构。分为类的结构模式和对象的结构模式。类的结构模式使用继承把类，接口等组合在一起，以形成更大的结构。类的结构模式是静态的。对象的结构模式描述怎样把各种不同类型的对象组合在一起，以实现新的功能的方法。对象的结构模式是动态的。

**行为模式：**对在不同的对象之间划分责任和算法的抽象化。不仅仅是关于类和对象的，并是关于他们之间的相互作用。类的行为模式使用继承关系在几个类之间分配行为。对象的行为模式则使用对象的聚合来分配行为。

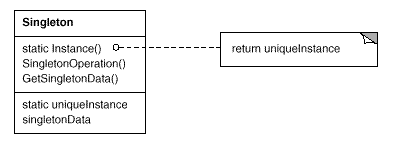
**设计模式使用排行：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 频率 | 所属类型 | 模式名称 | 模式 | 简单定义 |
| 5 | 创建型 | Singleton | 单件 | 保证一个类只有一个实例，并提供一个访问它的全局访问点。 |
| 5 | 结构型 | Composite | 组合模式 | 将对象组合成树形结构以表示部分整体的关系，Composite使得用户对单个对象和组合对象的使用具有一致性。 |
| 5 | 结构型 | FAÇADE | 外观 | 为子系统中的一组接口提供一致的界面，facade提供了一高层接口，这个接口使得子系统更容易使用。 |
| 5 | 结构型 | Proxy | 代理 | 为其他对象提供一种代理以控制对这个对象的访问 |
| 5 | 行为型 | Iterator | 迭代器 | 提供一个方法顺序访问一个聚合对象的各个元素，而又不需要暴露该对象的内部表示。 |
| 5 | 行为型 | Observer | 观察者 | 定义对象间一对多的依赖关系，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自动更新。 |
| 5 | 行为型 | Template Method | 模板方法 | 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，Template Method使得子类可以不改变一个算法的结构即可以重定义该算法得某些特定步骤。 |
| 4 | 创建型 | Abstract Factory | 抽象工厂 | 提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们的具体类。 |
| 4 | 创建型 | Factory Method | 工厂方法 | 定义一个用于创建对象的接口，让子类决定实例化哪一个类，Factory Method使一个类的实例化延迟到了子类。 |
| 4 | 结构型 | Adapter | 适配器 | 将一类的接口转换成客户希望的另外一个接口，Adapter模式使得原本由于接口不兼容而不能一起工作那些类可以一起工作。 |
| 4 | 结构型 | Decorator | 装饰 | 动态地给一个对象增加一些额外的职责，就增加的功能来说，Decorator模式相比生成子类更加灵活。 |
| 4 | 行为型 | Command | 命令 | 将一个请求封装为一个对象，从而使你可以用不同的请求对客户进行参数化，对请求排队和记录请求日志，以及支持可撤销的操作。 |
| 4 | 行为型 | State | 状态 | 允许对象在其内部状态改变时改变他的行为。对象看起来似乎改变了他的类。 |
| 4 | 行为型 | Strategy | 策略模式 | 定义一系列的算法，把他们一个个封装起来，并使他们可以互相替换，本模式使得算法可以独立于使用它们的客户。 |
| 3 | 创建型 | Builder | 生成器 | 将一个复杂对象的构建与他的表示相分离，使得同样的构建过程可以创建不同的表示。 |
| 3 | 结构型 | Bridge | 桥接 | 将抽象部分与它的实现部分相分离，使他们可以独立的变化。 |
| 3 | 行为型 | China of Responsibility | 职责链 | 使多个对象都有机会处理请求，从而避免请求的送发者和接收者之间的耦合关系 |
| 2 | 创建型 | Prototype | 原型 | 用原型实例指定创建对象的种类，并且通过拷贝这些原型来创建新的对象。 |
| 2 | 结构型 | Flyweight | 享元 | 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。 |
| 2 | 行为型 | Mediator | 中介者 | 用一个中介对象封装一些列的对象交互。 |
| 2 | 行为型 | Visitor | 访问者模式 | 表示一个作用于某对象结构中的各元素的操作，它使你可以在不改变各元素类的前提下定义作用于这个元素的新操作。 |
| 1 | 行为型 | Interpreter | 解释器 | 给定一个语言，定义他的文法的一个表示，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。 |
| 1 | 行为型 | Memento | 备忘录 | 在不破坏对象的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。 |

**一 ： 单例模式(Singleton)**

单例模式：Singleton的作用是保证在应用程序中，一个类Class只有一个实例存在。并提供全局访问。

**结构：**



**账本类：1 单一实例 2 给多个对象共享 3 自己创建**

**网页计数器**  
public class LazySingleton

{  
     private static LazySingleton newInstance = null;

  private LazySingleton ()

{

}

public static synchronized  LazySingleton getInstance ()

{  
               if (newInstance == null)

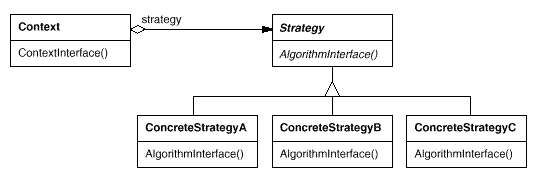
{  
                newInstance = new LazySingleton ();  
          }  
          return newInstance;  
}

}  
singleton限制了实例个数，有利于gc的回收。

二：**策略模式(Strategy)**

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。

**结构：**



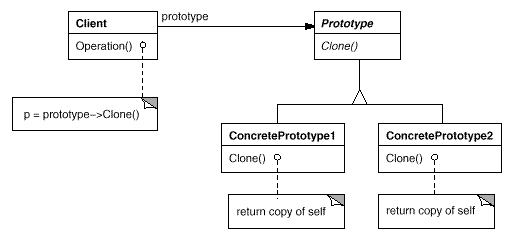
**使用QQ泡MM时使用外挂  客户端 ：ME 抽象类： 外挂 具体：策略（图片，笑话，名人名言）**

**图书销售算法（不同书本折扣的算法）**

**三：原型模式(Prototype)**

原型模式：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

**结构：**



**复印技术： 1 不是同一个对象 2 属同类**

**短消息（转发） 1-n个MM**

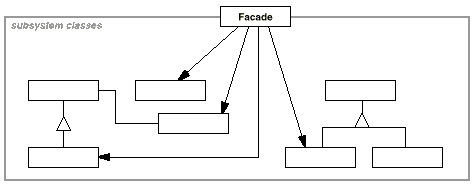
因为Java中的提供clone()方法来实现对象的克隆,所以Prototype模式实现一下子变得很简单.

**四：门面模式(Façade)**

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，减少复杂性。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。

1 门面角色 2 子系统角色

**结构:**



**Facade典型应用就是数据库JDBC的应用和Session的应用**

**五：备忘录模式(Memento)**

Memento模式：Memento对象是一个保存另外一个对象内部状态拷贝的对象，这样以后就可以将该对象恢复到原先保存的状态。模式的用意是在不破坏封装的条件下，将一个对象的状态捕捉住，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。模式所涉及的角色有三个，备忘录角色、发起人角色和负责人角色。

备忘录角色的作用：

（1） 将发起人对象的内部状态存储起来，备忘录可以根据发起人对象的判断来决定存储多少发起人对象的内部状态。

（2） 备忘录可以保护其内容不被发起人对象之外的任何对象所读取。

发起人角色的作用：

（1） 创建一个含有当前内部状态的备忘录对象。

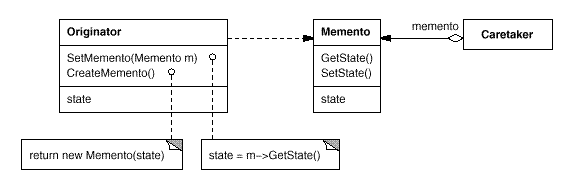
（2） 使用备忘录对象存储其内部状态。

负责人角色的作用：

（1） 负责保存备忘录对象。

（2） 不检查备忘录对象的内容

**结构：**



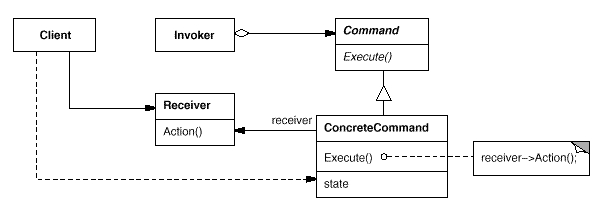
**备份系统时使用**

**GHOST**

**六 ： 命令模式（Command）**

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。

**结构：**



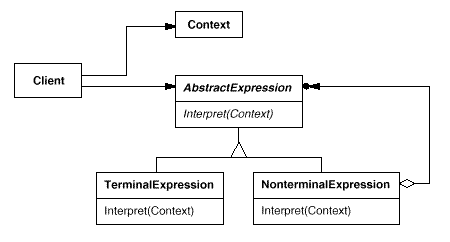
**MM（客户端）--àME（请求者）--à命令角色--à（具体命令）-à代理处（接收者）--àMM**

**上网 IE 输入 http地址 发送命令**

**七： 解释器(Interpreter)**

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。

**结构：**



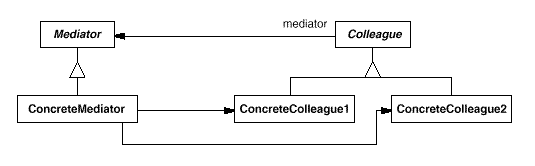
**编译原理之编译器**

**文言文注释：一段文言文，将它翻译成白话文**

**八：调停者模式(Mediator)**

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。

**结构：**

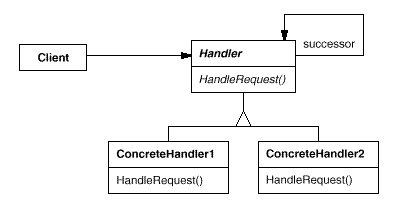


**法院和原告，被告的关系**

**九：责任链模式(CHAIN OF RESPONSIBLEITY)**

责任链模式：执行者的不确定性 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。

**结构：**



**典型的对象结构：**

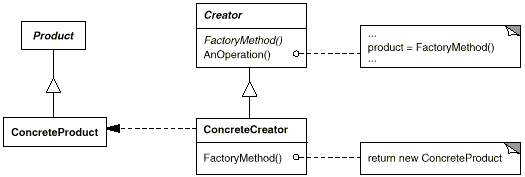


**喝酒时通过成语接龙决定谁喝酒(马到成功－功不可没－没完没了)**

**十：工厂模式（Factory）**

**工厂模式：**定义一个用于创建对象的接口，让接口子类通过工厂方法决定实例化哪一个类。

**结构：**

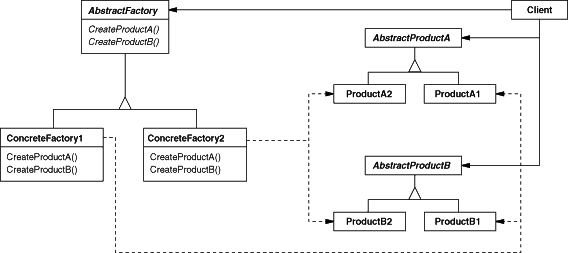


**水果园—〉（葡萄园，苹果园）--〉（葡萄，苹果）（各自生产）**

**十一：抽象工厂模式（Abstract Factory）**

抽象工厂模式：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。

**结构：**



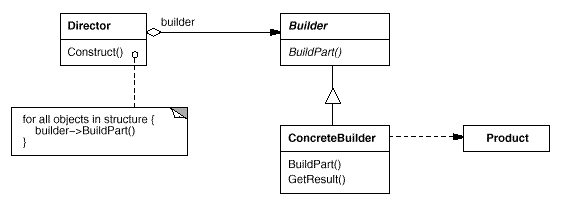
**女娲造人---〉（阴，阳）--〉（人，兽）----〉（男人，女人，公兽，母兽）(人和兽属于不同的产品类)**

**十二：建造模式（Builder）**

建造模式：将一个复杂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.Builder模式是一步一步创建一个复杂的对象,它允许用户可以只通过指定复杂对象的类型和内容就可以构建它们.用户不知道内部的具体构建细节.Builder模式是非常类似抽象工厂模式,细微的区别大概只有在反复使用中才能体会到。

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。

**结构：**



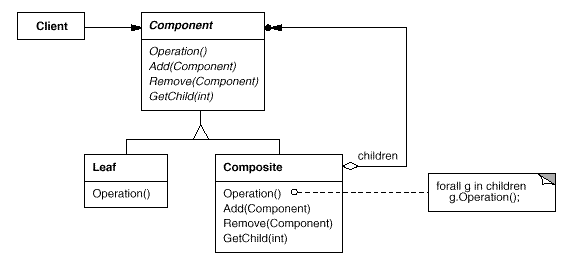
**交互图：**

**汽车制造**

**十三：合成模式（Composite）**

**合成模式：**将对象以树形结构组织起来,以达成“部分－整体” 的层次结构，使得客户端对单个对象和组合对象的使用具有一致性. 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。

**结构：**



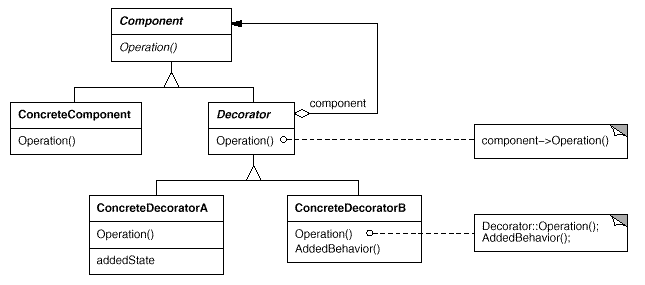
**windows的目录树（文件系统）**

**十四：装饰模式（DECORATOR）**

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。

使用Decorator的理由是:这些功能需要由用户动态决定加入的方式和时机.Decorator提供了"即插即用"的方法,在运行期间决定何时增加何种功能.

**结构：**



**在visio中文件可以使用背景进行装饰**

**变废为宝**

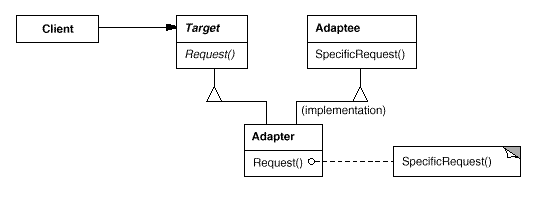
**十五：设计模式之Adapter(适配器)**

**适配器模式:**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  
将两个不兼容的类纠合在一起使用，属于结构型模式,需要Adaptee(被适配者)和Adaptor(适配器)两个身份.

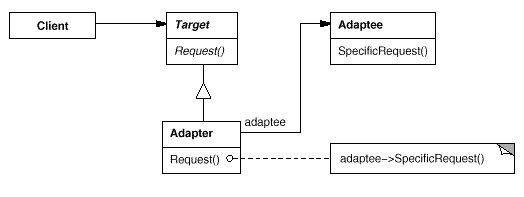
**为何使用?**  
我们经常碰到要将两个没有关系的类组合在一起使用,第一解决方案是：修改各自类的接口，但是如果我们没有源代码，或者，我们不愿意为了一个应用而修改各自的接口。 怎么办? 使用Adapter，在这两种接口之间创建一个混合接口(混血儿).

**如何使用?**  
实现Adapter方式,其实"think in Java"的"类再生"一节中已经提到,有两种方式：组合(composition)和继承(inheritance).

**结构：**



**对象结构：**



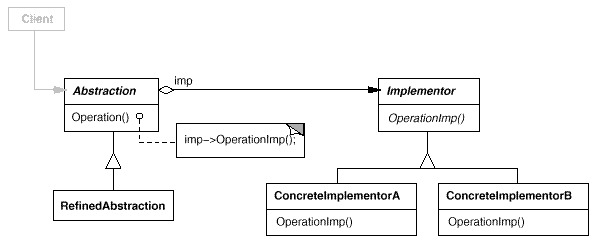
**充电器（手机和220V电压）**

**jdbc-odbc桥**

**十六：桥梁模式（Bridge）**

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。

**结构：**

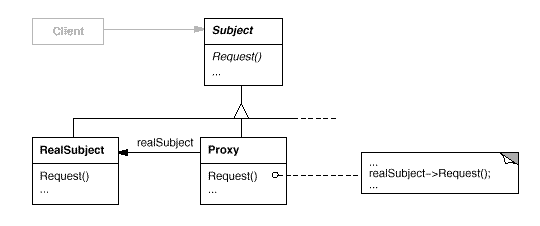


**jdbc驱动程序**

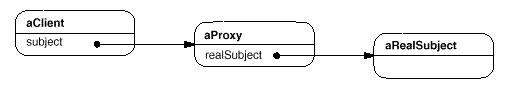
**十七：代理模式（Proxy）**

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。

**结构：**



**运行时的代理结构：**



**用代理服务器连接出网**

**销售代理（厂商）律师代理（客户）**

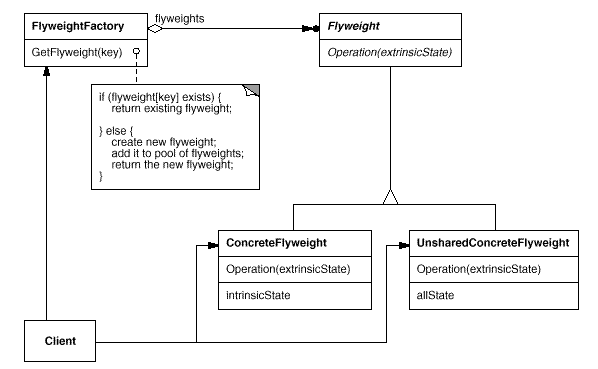
**foxmail**

**枪手**

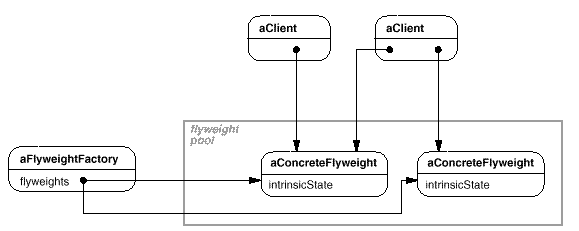
**十八：享元模式（Flyweight）**

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。

**结构：**



**共享方法：**

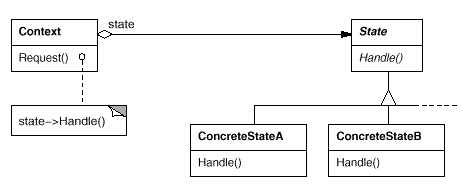


**字体的26个字母和各自的斜体等**

**十九：状态模式（State）**

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。

**结构：**



**人心情不同时表现不同有不同的行为**

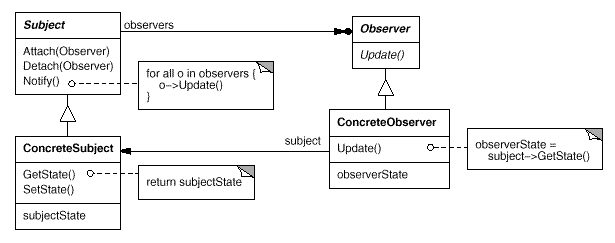
**编钟**

**登录login logout**

**二十：观察者模式（Observer）**

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。发布订阅。

**结构：**



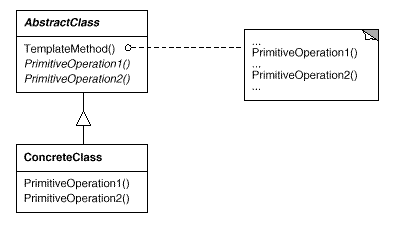
**公司邮件系统everyone@sina.com的应用。当公司员工向这个邮箱发邮件时会发给公司的每一个员工。如果设置了Outlook则会及时收到通知。**

**接收到短消息**

**二十一：模板方法模式（Template）**

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。

**结构：**

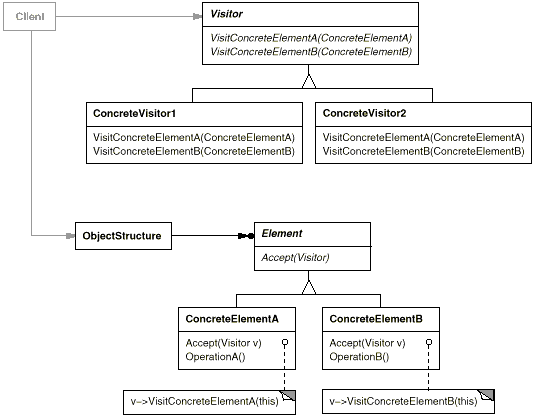


**使用网页设计时使用的模板架构网页（骨架） 算法的各个逻辑系统**

**二十二：访问者模式（Visitor）**

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。

**结构：**

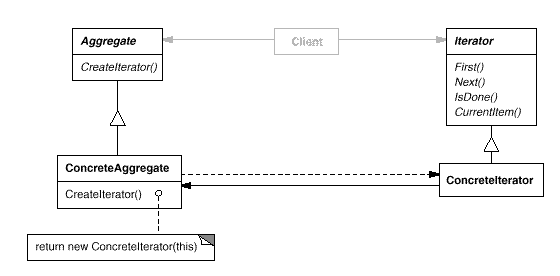


**电脑销售系统： 访问者（自己）---〉电脑配置系统（主板，CPU，内存。。。。。。）**

**二十三：迭代子模式（Iterator）**

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。

**结构：**



**查询数据库，返回结果集（map， list， set）**

**二十四：MVC模式**

MVC模式：它强制性的使应用程序的输入、处理和输出分开。使用MVC应用程序被分成三个核心部件：模型、视图、控制器。它们各自处理自己的任务。相互通信。

MVC还使用了的设计模式，如：用来指定视图缺省控制器的Factory Method和用来增加视图滚动的Decorator。但是MVC的主要关系还是由Observer、Composite和Strategy三个设计模式给出的。

