一、设计模式（Design pattern）

是一套被反复使用、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可重用代码、让代码更容易被他人理解、保证代码可靠性；
- 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脉络，如同大厦的结构一样。

二、设计模式分为三大类：

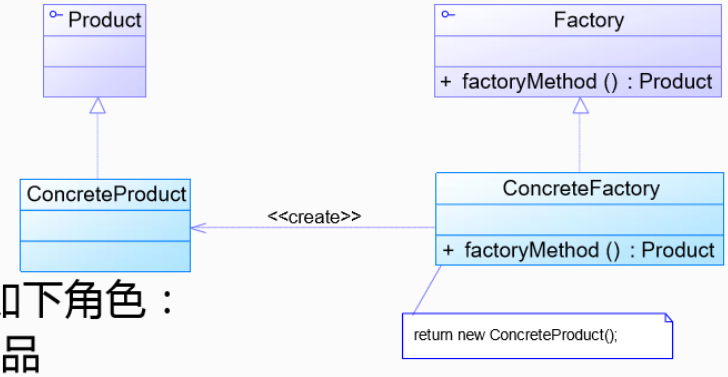
- 创建型模式主要用于创建对象。共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。
- 结构型模式主要用于处理类或对象的组合。共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。
- 行为型模式主要用于描述类或对象怎样交互和怎样分配职责。共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

1.工厂方法模式（Factory Method）

也叫虚拟构造器模式，它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，使一个类的实例化延迟到其子类。

在工厂方法模式中，工厂父类负责定义创建产品对象的公共接口，而工厂子类则负责生成具体的产品对象，这样做的目的是将产品类的实例化操作延迟到工厂子类中完成，即通过工厂子类来确定究竟应该实例化哪一个具体产品类。

工厂方法模式结构



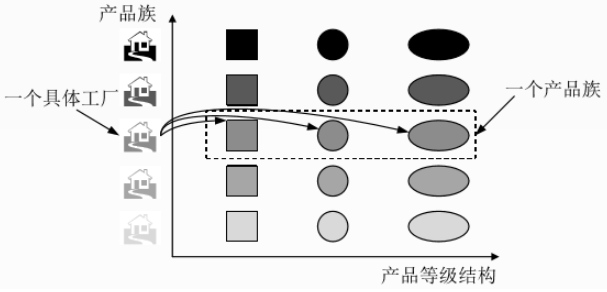
工厂方法模式包含如下角色：

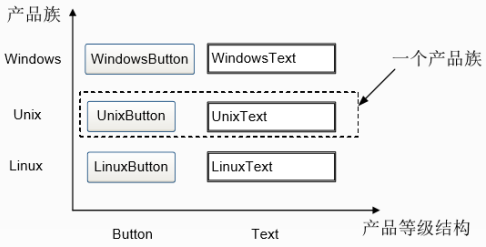
- Product：抽象产品
- ConcreteProduct：具体产品
- Factory：抽象工厂
- ConcreteFactory：具体工厂

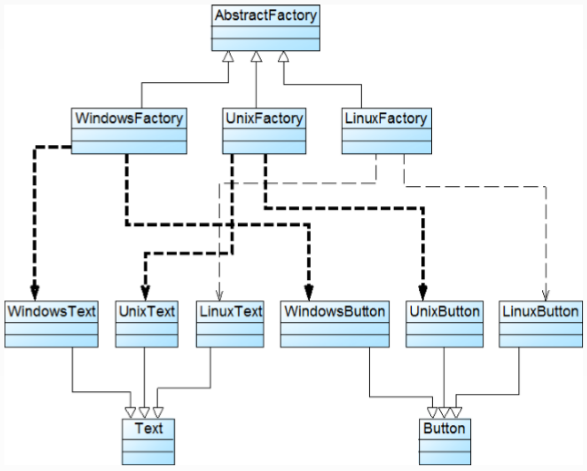
2.抽象工厂模式（Abstract Factory）

提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。属于对象创建型模式。

抽象工厂模式与工厂方法模式最大的区别：工厂方法模式针对的是一个产品等级结构，而抽象工厂模式则需要面对多个产品等级结构。



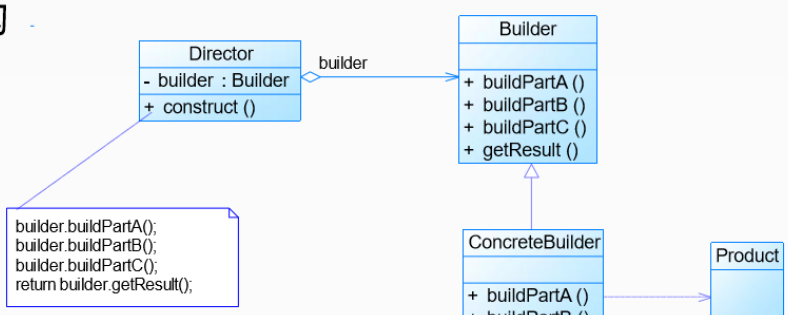




3.建造者模式（Builder Pattern、生成器模式）

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。建造者模式将部件和其组装过程分开，一步一步创建一个复杂的对象。用户只需要指定复杂对象的类型就可以得到该对象，而无须知道其内部的具体构造细节。

建造者模式结构

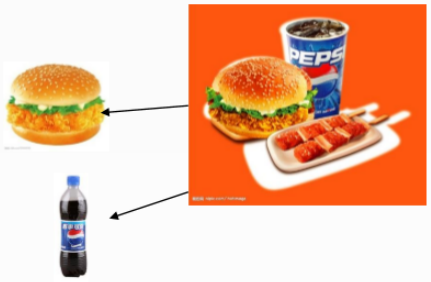


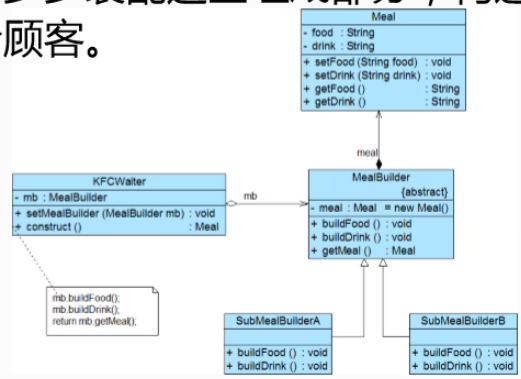
建造者模式包含如下角色：

- Builder：抽象建造者
- ConcreteBuilder：具体建造者
- Director：指挥者
- Product：产品角色

例：KFC套餐

建造者模式可以用于描述KFC如何创建套餐：套餐是一个复杂对象，它一般包含主食（汉堡、鸡肉卷等）和饮料（果汁、可乐等）等组成部分，不同的套餐有不同的组成部分，而KFC的服务员根据顾客的要求，一步步装配这些组成部分，构造一份完整的套餐，然后返回给顾客。



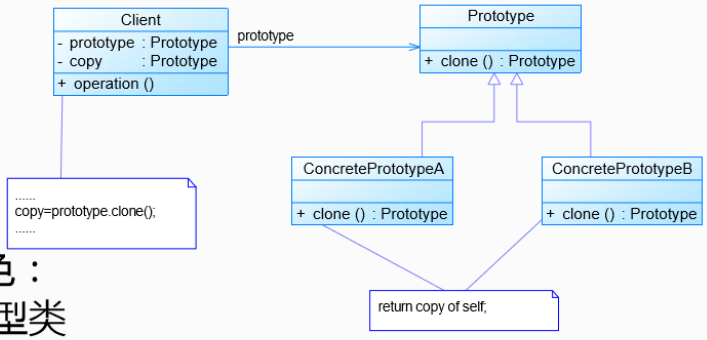


4.原型模式（Prototype）

用原型实例指定创建对象的种类，并且通过复制这些原型创建新的对象。

原型模式允许一个对象再创建另外一个可定制的对象，无须知道任何创建的细节。

模式结构

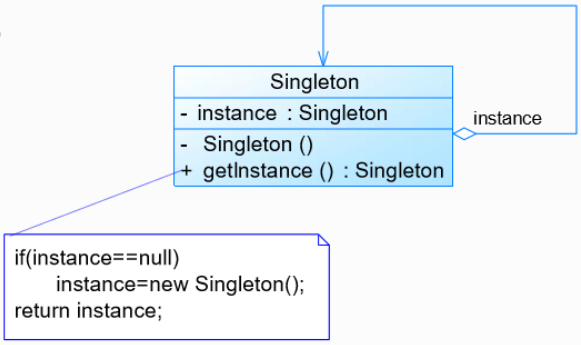


原型模式包含如下角色：

- Prototype：抽象原型类
- ConcretePrototype：具体原型类
- Client：客户类

5.单例模式（Singleton）

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，这个类称为单例类，它提供全局访问的方法。例如，一个系统中可以存在多个打印任务，但是只能有一个正在工作的任务。



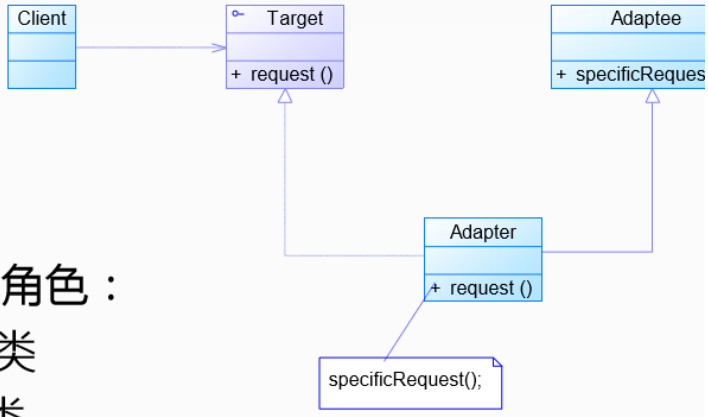
6.适配器模式（Adapter Class/Object）

将一个类的接口转换成客户希望的另外一个接口。使得原本不相容的接口可以协同工作。

适用性：

- 想使用一个已经存在的类，而它的接口不符合你的需求。
- 想创建一个可以复用的类，该类可以与其他不相关的类或不可预见的类（即那些接口可能不一定兼容的类）协同工作。

适配器模式结构



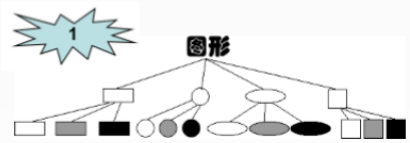
适配器模式包含如下角色：

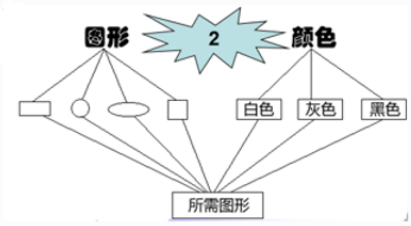
- Target：目标抽象类
- Adapter：适配器类
- Adaptee：适配者类
- Client：客户类

7.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。

如果要绘制矩形、圆形、椭圆、正方形，至少需要4个形状类，但是如果绘制的图形需要具有不同的颜色，如红色、绿色、蓝色等。

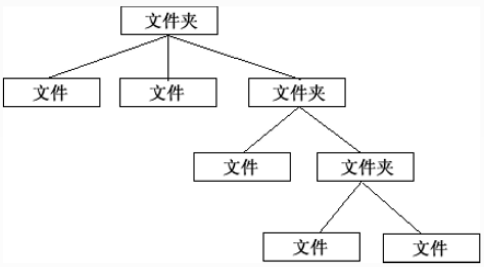




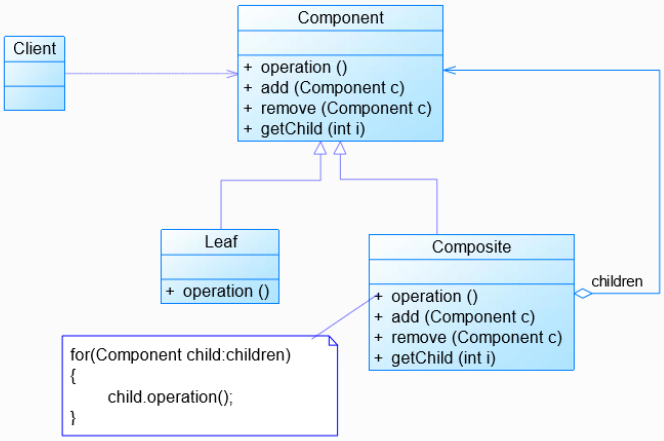
8.组合模式（Composite）

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

组合模式描述了如何将容器对象和叶子对象进行递归组合，使得用户在使用时无须对它们进行区分，可以一致地对待容器对象和叶子对象。



组合模式结构

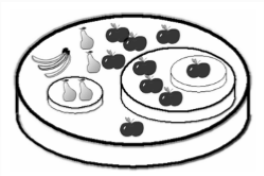


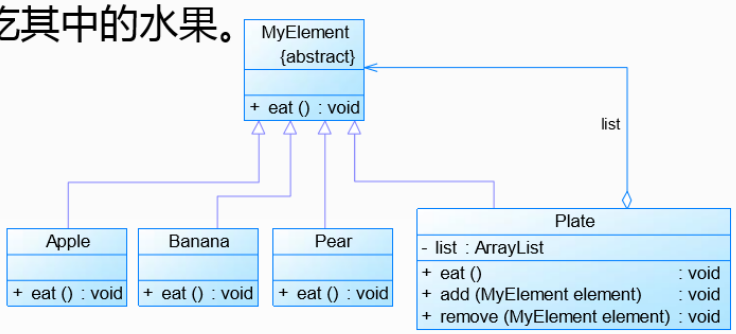
组合模式包含如下角色：

- Component: 抽象构件
- Leaf: 叶子构件
- Composite: 容器构件
- Client: 客户类

例：水果盘

在水果盘(Plate)中有一些水果，如苹果(Apple)、香蕉(Banana)、梨子(Pear)，当然大水果盘中还可以有小水果盘，现需要对盘中的水果进行遍历（吃），当然如果对一个水果盘执行“吃”方法，实际上就是吃其中的水果。





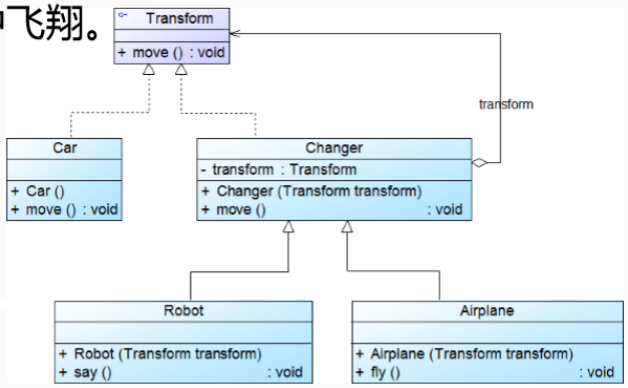
9.装饰模式（Decorator）

动态地给一个对象添加一些额外的职责。提供了用子类扩展功能的一个灵活的替代，但比生成子类更为灵活。

装饰模式可以在不需要创造更多子类的情况下，将对象的功能加以扩展。

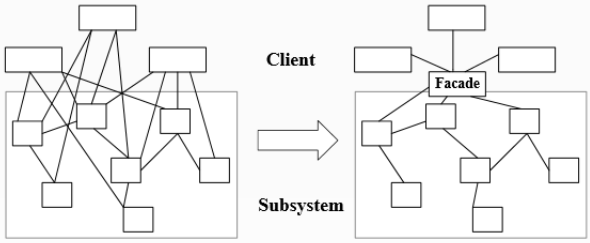
例：变形金刚

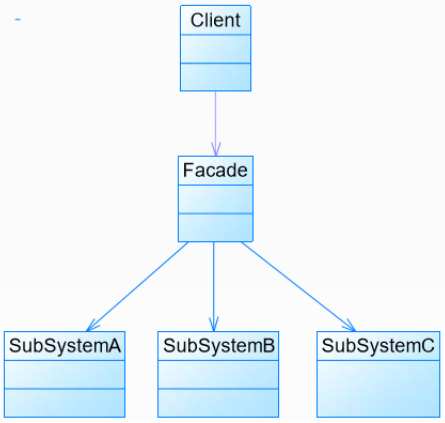
变形金刚在变形之前是一辆汽车，它可以在陆地上移动。当它变成机器人之后除了能够在陆地上移动之外，还可以说话；还可以变成飞机，在天空中飞翔。



10.外观模式（Facade）

定义了一个高层接口，为子系统中的一组接口提供一个一致的界面，从而简化子系统的使用。





11.享元模式（Flyweight）

提供支持大量细粒度对象共享的有效方法。

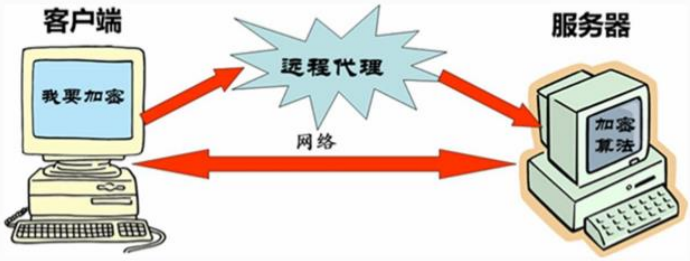
面向对象技术可以很好地解决灵活性或可扩展性问题，但在很多情况下需要在系统中增加类和对象的个数。当对象数量太多时，将导致运行代价过高，带来性能下降等问题。

如在一个文档中多次出现相同的图片，则只需要创建一个图片对象，通过在应用程序中设置该图片出现的位置，可以实现该图片在不同地方多次重复显示。

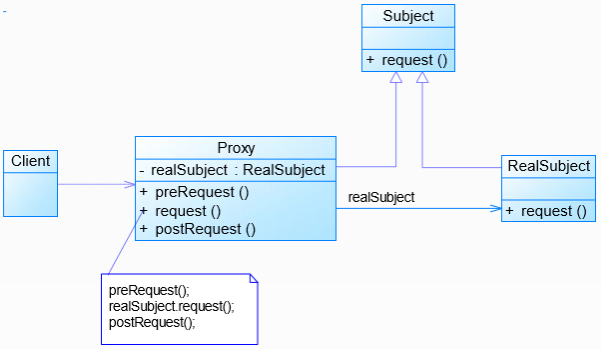
享元模式通过共享技术实现相同或相似对象的重用。

12.代理模式（Proxy）

给某一个对象提供一个代理，并由代理对象控制对原对象的引用。



模式结构



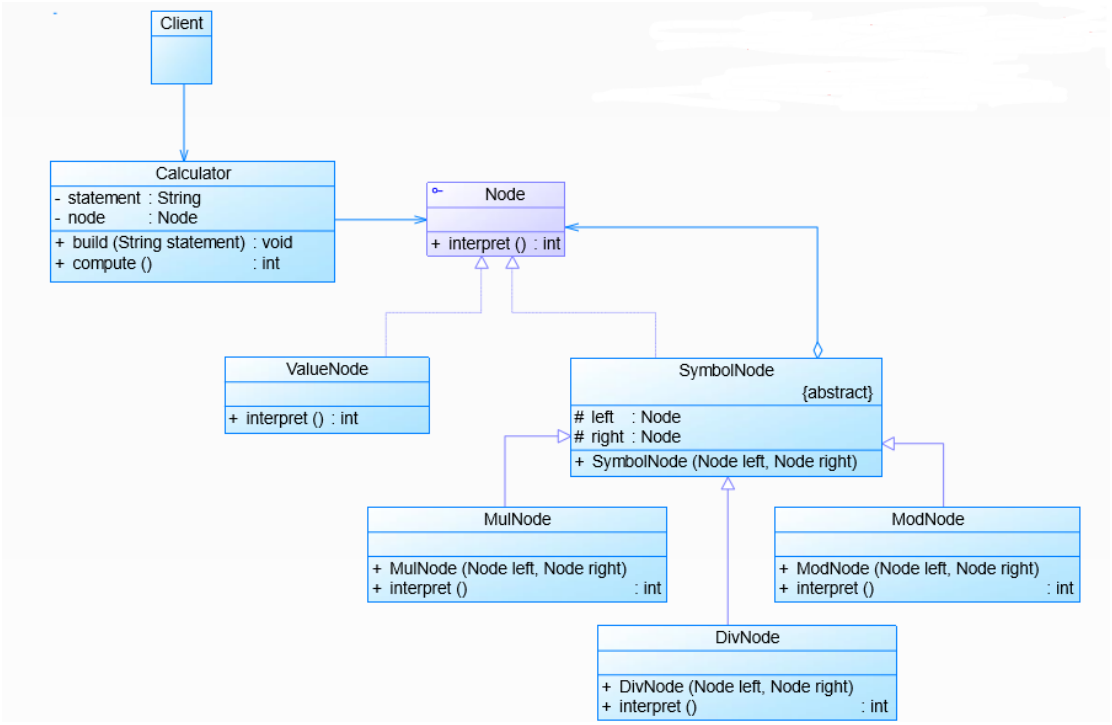
代理模式包含如下角色：

- Subject: 抽象主题角色
- Proxy: 代理主题角色
- RealSubject: 真实主题角色

13.解释器（Interpreter）

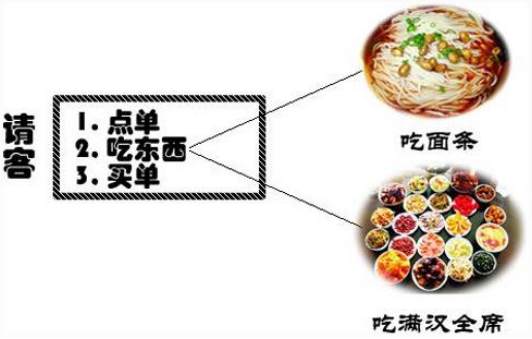
定义语言的文法，并且建立一个解释器来解释该语言中的句子，这里的“语言”意思是使用规定格式和语法的代码。

例：构造一个语言解释器，使得系统可以执行整数间的乘、除和求模运算。如用户输入表达式“3 * 4 / 2 % 4”，输出结果为2。

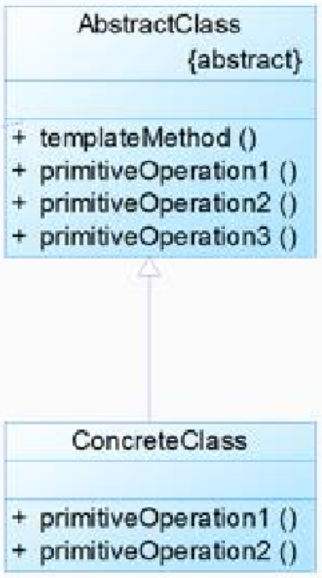


14.模板方法（Template Method）

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。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。



模板方法模式结构

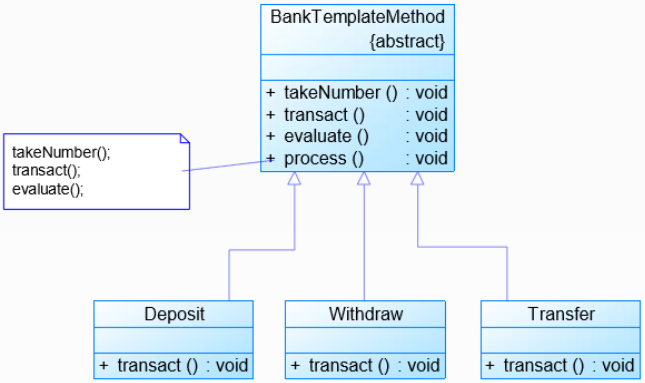


模板方法模式包含如下角色：

- AbstractClass: 抽象类
- ConcreteClass: 具体子类

例：银行业务办理流程

在银行办理业务时，一般包含几个基本步骤，首先取号排队，然后办理具体业务，最后对银行工作人员进行评分。无论具体业务是取款、存款还是转账，基本流程都一样。

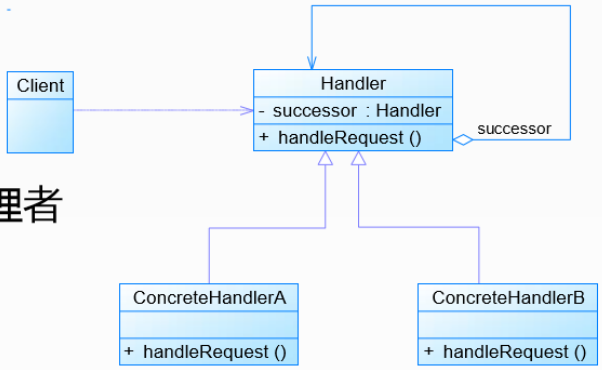


15.责任链（Chain of Responsibility）

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，并沿着这条链传递该请求，直到有一个对象处理它为止。

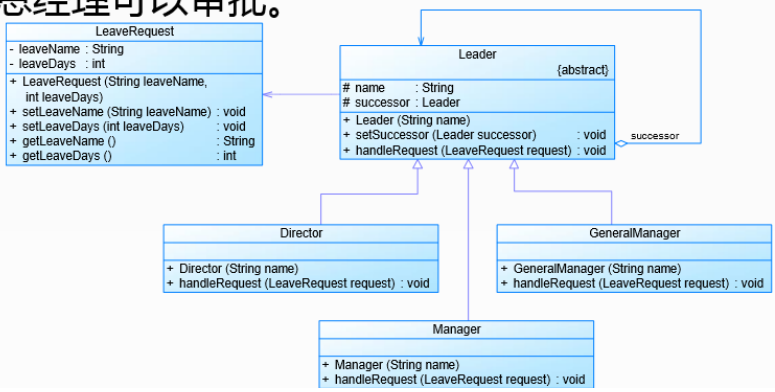
职责链模式包含如下角色：

- Handler: 抽象处理者
- ConcreteHandler: 具体处理者
- Client: 客户类



实例：审批假条

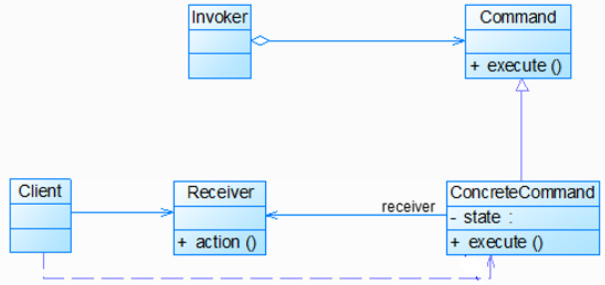
某OA系统需要提供一个假条审批的模块，如果员工请假天数小于3天，主任可以审批该假条；如果员工请假天数大于等于3天，小于10天，经理可以审批；如果员工请假天数大于等于10天，小于30天，总经理可以审批。



16.命令模式（Command）

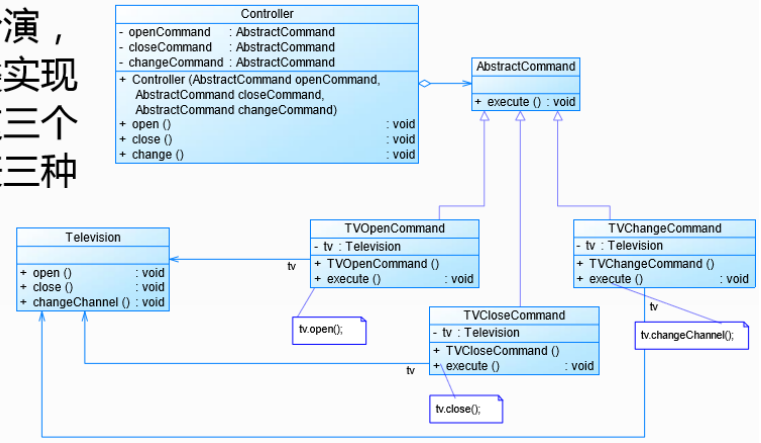
将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化。

对命令进行封装，将发出命令的责任和执行命令的责任分割开。请求的一方不必知道接收请求的一方的接口，也不必知道请求是怎么被接收，以及操作是否被执行、何时被执行，以及是怎么被执行的。



例：电视机遥控器

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。



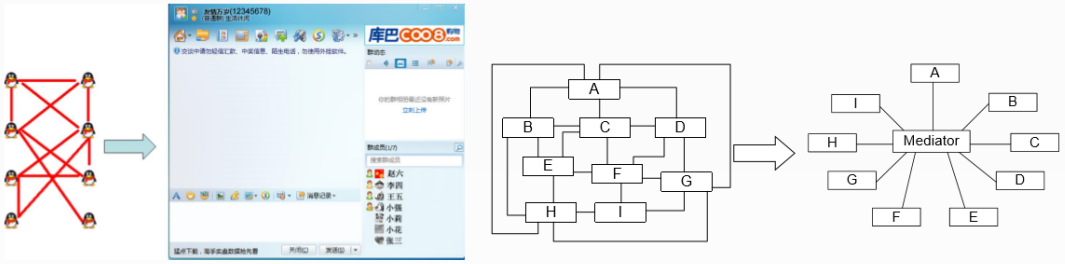
17.迭代器模式（Iterator）

提供一种方法顺序访问一个聚合对象中各个元素，而又不需暴露该对象的内部表示。

怎样遍历一个聚合对象，又不需要了解聚合对象的内部结构，还能够提供多种不同的遍历方式，这就是迭代器模式所要解决的问题。

18.中介者模式（Mediator）

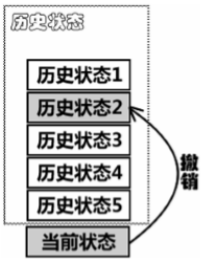
用一个中介对象来封装一系列的对象交互。中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。



19.备忘录模式（Memento）

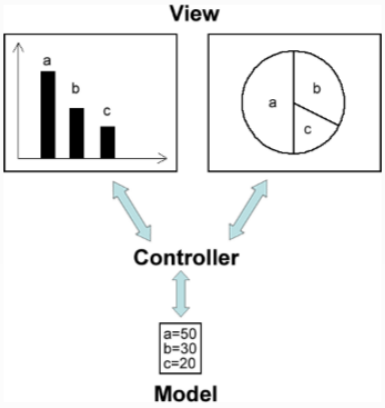
在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。这样以后就可将该对象恢复到原先保存的状态。

为了使软件更加人性化，对于误操作，需要提供一种类似“后悔药”的机制，让软件可以回到误操作前的状态，因此需要保存用户每一次操作后系统的状态，一旦出现误操作，可以把存储的历史状态取出即可回到之前的状态。



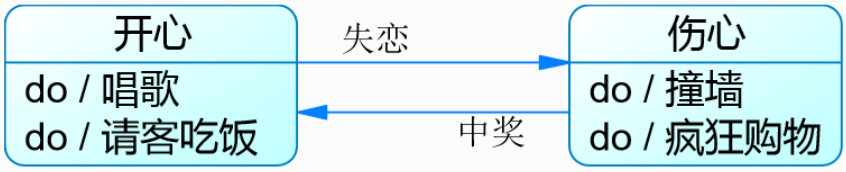
20.观察者模式（Observer）

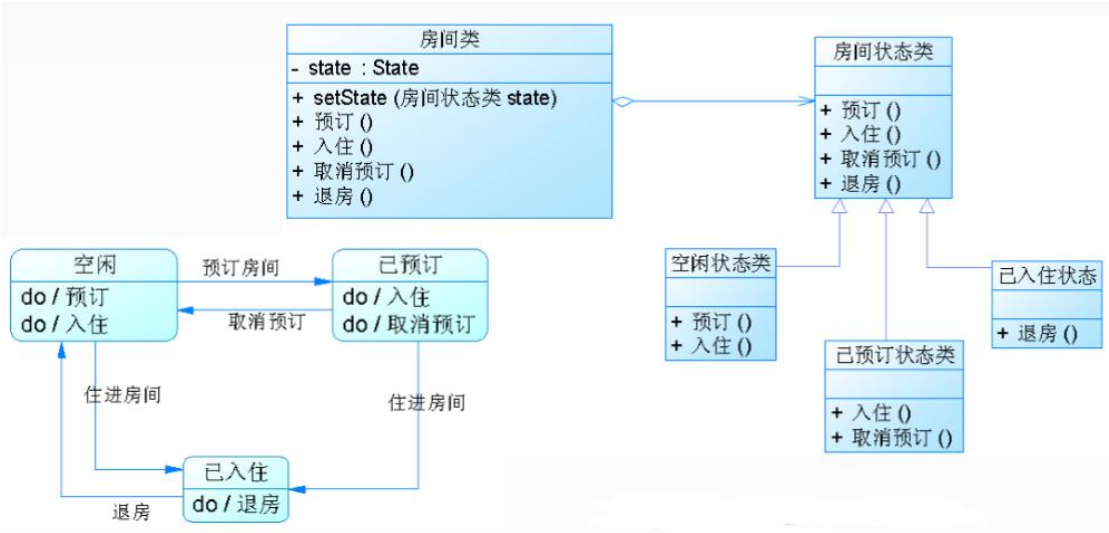
定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。



21. State模式（状态）

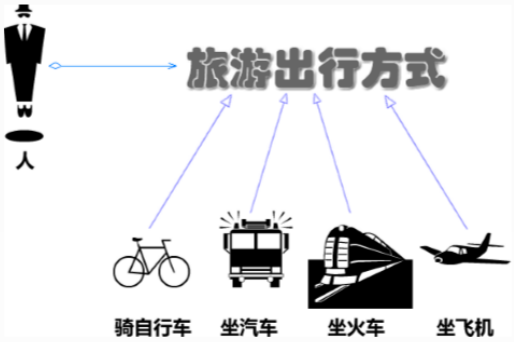
允许一个对象在其内部状态改变时改变它的行为。对象看起来似乎修改了它的类。





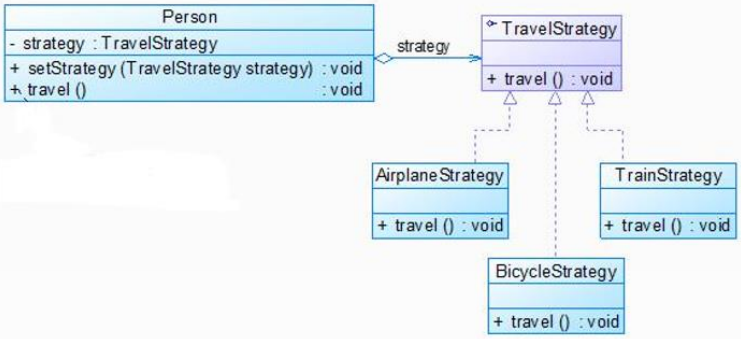
22.策略模式（Strategy）

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。本模式使得算法可独立于使用它的客户而变化。



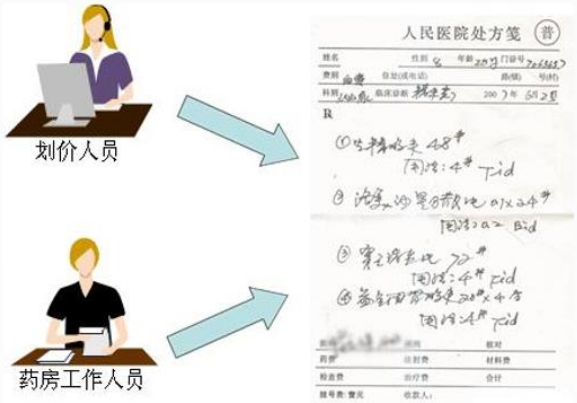
例：旅游出行策略

旅游出行方式可以有多种，如乘坐飞机旅游，乘火车旅游，骑自行车游也是一种出行方式。不同的旅游出行方式有不同的实现过程，客户根据自己的需要选择一种合适的旅游方式。



23.访问者模式（Visitor）

表示一个作用于某对象结构中的各元素的操作，它使我们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。



访问者模式结构

访问者模式包含如下角色

- Vistor: 抽象访问者
- ConcreteVisitor: 具体访问者
- Element: 抽象元素
- ConcreteElement: 具体元素
- ObjectStructure: 对象结构