设计模式总结

简单工厂违背开放-封闭原则——每次扩展都要修改工厂类（即：对对修改开放）

单一职责原则

开发封闭原则

依赖倒转原则

里氏代换原则

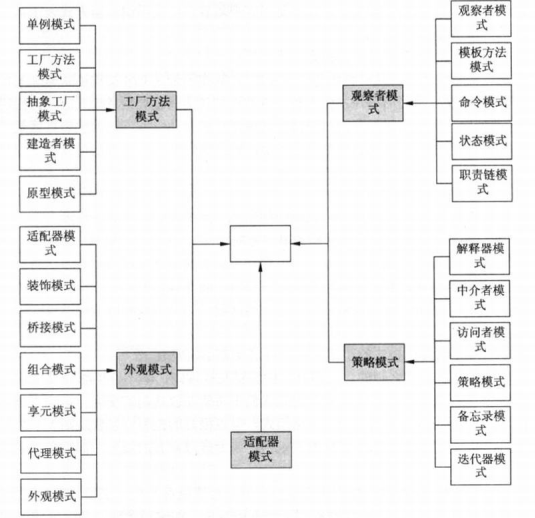
合成聚合复用原则

迪米特原则

总结：

|  |
| --- |
| 机器语言编程——程序员解决内存和外存容量  结构化编程——面向过程是开发【代码分割成多个模块，增强代码复用性，方便调试和修改——业务流程】  面向对象——关注的是对象【定义自己负责的事务，做要求它自己做的事情；自己对自己负责且应该清楚定义责任】  对象比流程更稳定和封闭  对象通过接口与外界联系，接口内部的事情与外界无关。  流程表面是只有一个入口和出口，但流程的每一步都可能改变某个数据的内部，改变某个设备的状态，对外界产生的影响——【划分功能模块，通过函数调用实现，需求更改，函数即要更改，函数之间的条用，关联会导致函数更改产生不可预测的bug】 |

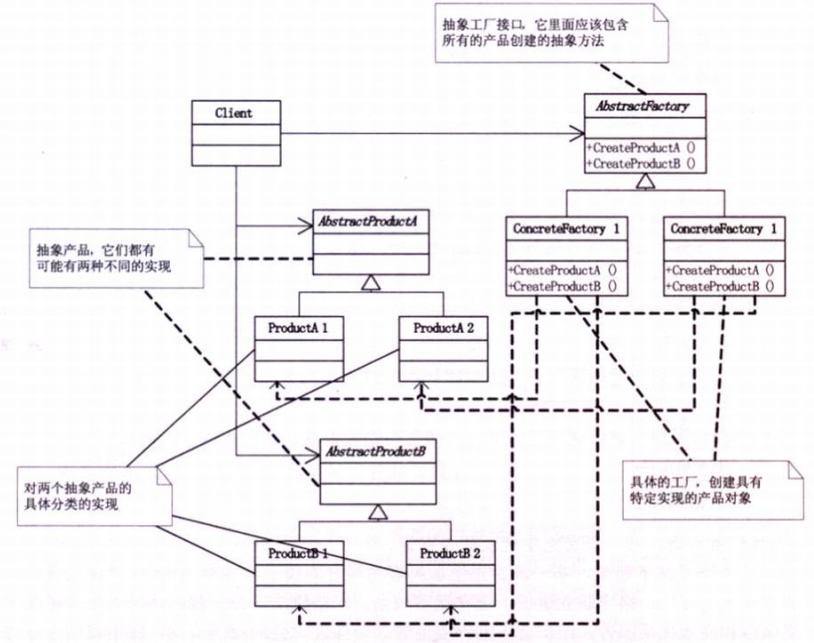
**设计模式特点分类：**



## 创建型模式

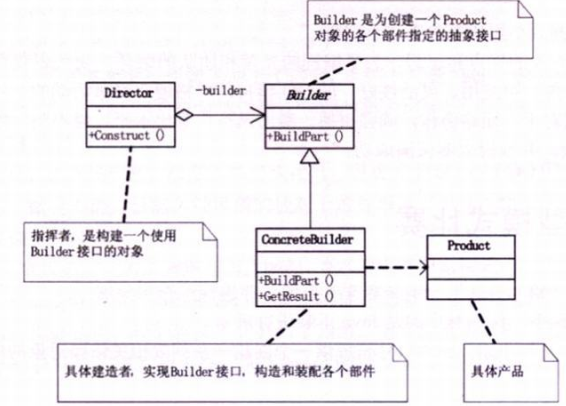
**抽象工厂**——提供一个创建一系列或相关依赖对象的接口，而无需指定它们具体的类。

解决多个类型产品的创建 例如：同一对象与多个数据库ORM通过抽象工厂实现



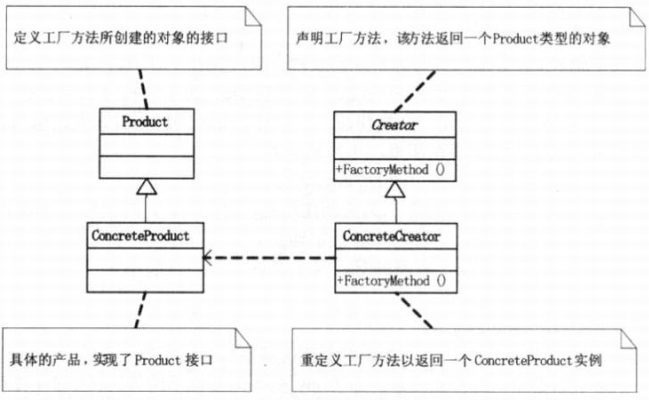
**建造者模式——**将一个复杂对象的构建与它的表示分离，使得同样的构建过程可创建不同的表示。

建造者把创建过程隐藏，一个请求，完整产品就创建，高内聚，低耦合



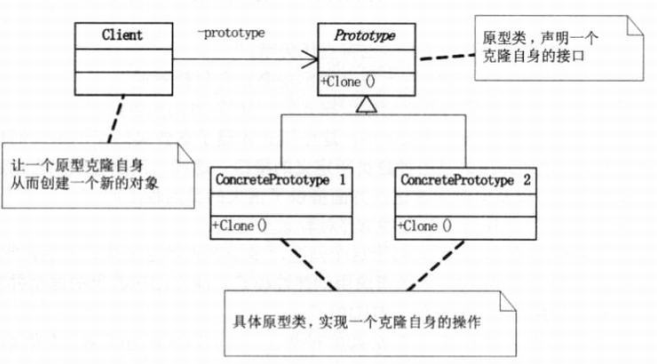
**工厂方法**——定义一个用于创建对象的接口，让子类决定实例化哪一个类，工厂模式使一个类的实例化延迟到其子类。

**职责分类**

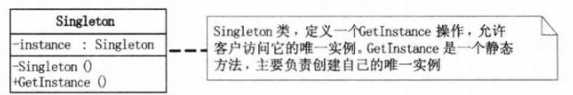


**原型模式**——用原型实例指定创建对象的种类，且通过拷贝这些原型创建新的对象。

Clone 解决数据结构的复制问题，copy不但复制结构，数据也复制



**单例模式**——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



抽象工厂——创建型模式隐藏了这些类的实例是如何被创建和放在一起，整个系统关于这些对象所知道的是由抽象类所定义的接口。这样，创建型模式在创建什么，谁创建它，它是怎么被创建的，以及何时创建这些方面提供了很大的灵活性

原型模式——当一个系统应该独立于它的产品创建，构成和表示时，应该考虑用创建型模式。建立相应数据的原型并克隆它们通常比每次用合适的状态手工实例化该类更方便。

内聚性——描述的是一个例程内部组成部分之间相互联系的紧密程度

耦合性——描述的是一个例程与其他例程之间联系的紧密程度

软件开发应该创建例程应该是：内部完成【高内聚】，但与其他例程之间的联系则是小巧，直接，可见，灵活——松耦合

建造者如何实践松耦合？

建造者是将一个复杂对象的构建与它的表示分离；易改变一个产品的内部表示，且使构造代码和表示代码分开。客户无需关心产品的创建过程，只需知道需要即可， 创建者用同样的构建过程创建不同的产品给客户。

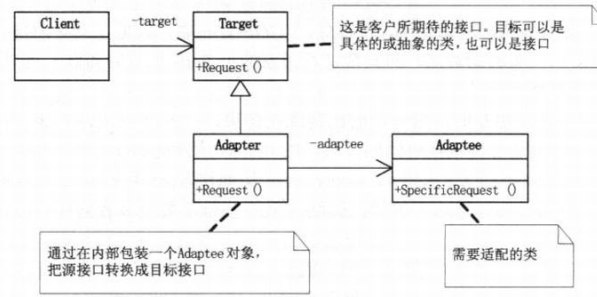
单例模式——对一些类，一个实例很重要。一个全局变量可使得一个对象被访问，但它不能防止客户实例化多个对象。单例优势是让类自身负责保存它的唯一实例。这个类可保证没有其他实例可被创建，且提供访问该实例的方法。这使得对唯一的实例可严格地控制客户怎样以及何时访问它。

工厂方法——创建型模式抽象了实例化的构成。它们帮助一个系统独立于如何创建，组合和表示它的那些对象。创建型模式都会将关于该系统使用哪些具体的类的信息封装起来。允许客户用结构和功能差别很大的“产品”对象配置一个系统。配置可以是静态的即在编译时指定，也可是动态的即运行时再指定。

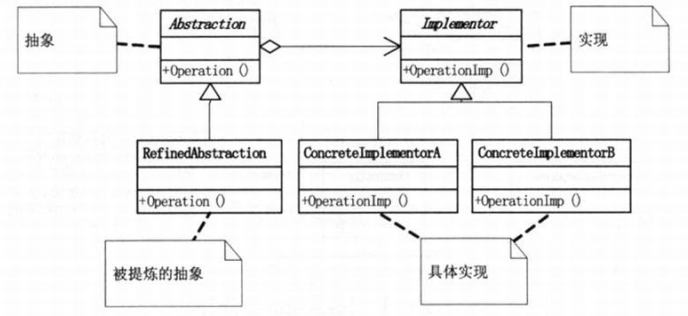
工厂方法优势——

## 结构型模式

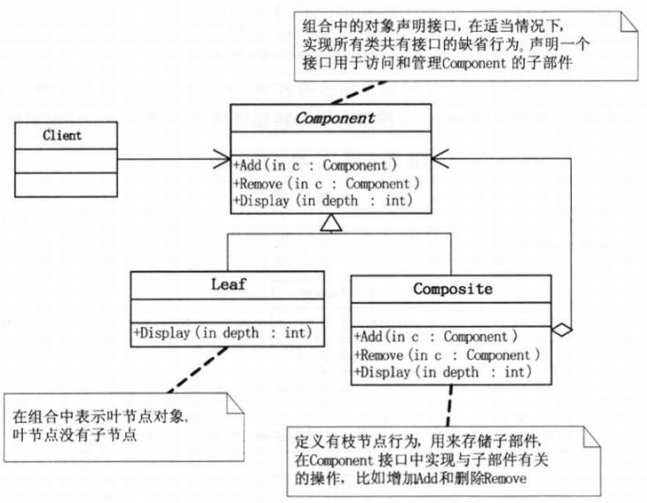
**适配器模式**——将一个类的接口转换成客户希望的另一个接口。适配器模式使用原本由于接口不兼容而不能一起工作的哪些类可以一起工作。



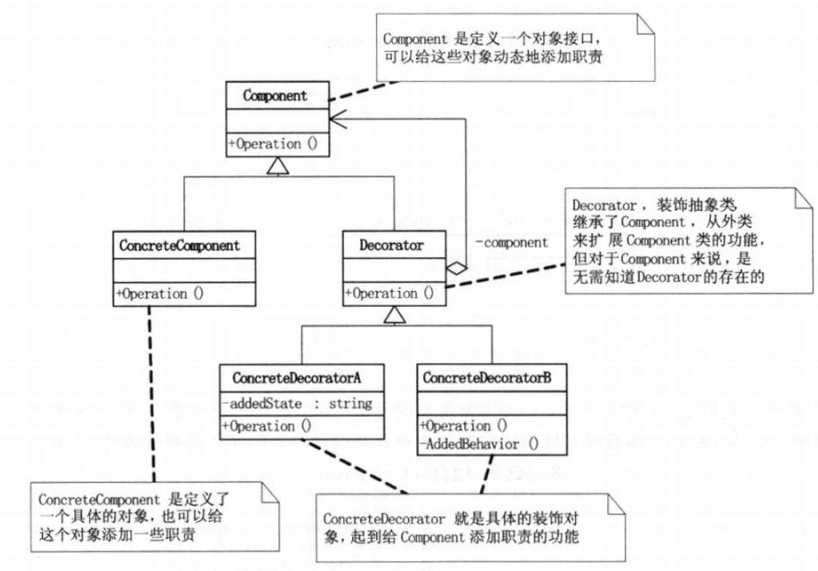
**桥接模式**——将抽象部门与它的实现部门分离，使它们都可以独立地变化。



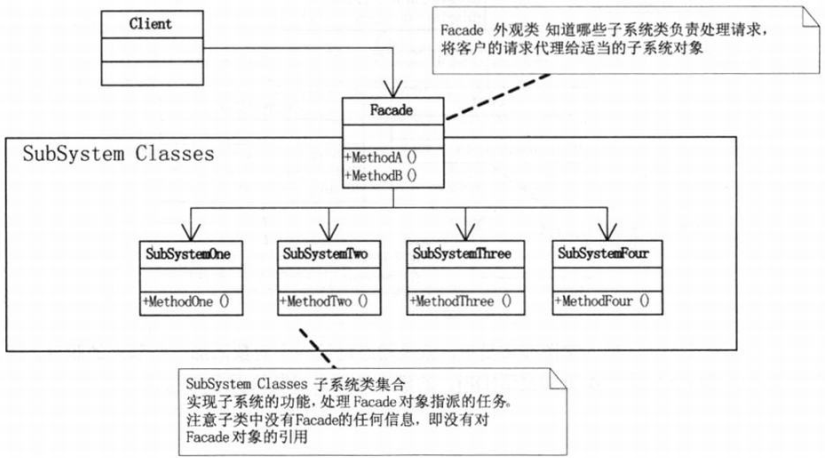
**组合模式**——将对象合成树形结构以表示“部分-整体”的层次结构，组合模式使得用户对单个对象和组合对象的使用具有一致性。



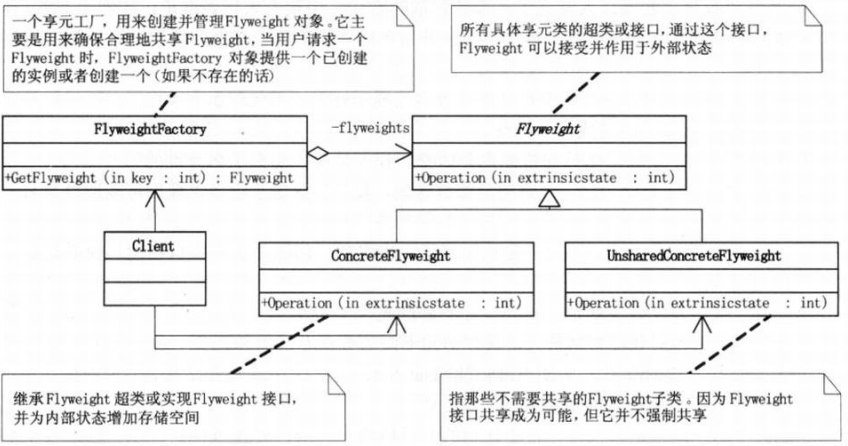
**装饰模式**——动态地给一个对象添加一些额外的职责。



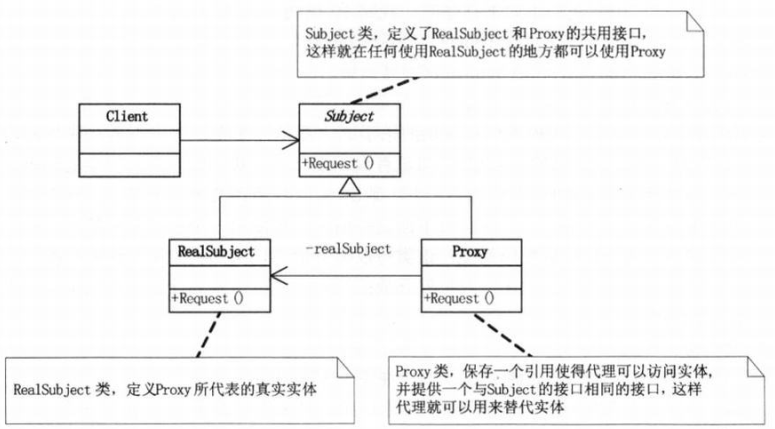
**外观模式**——为子系统中的一组接口提供一个一致的界面，外观模式定义一个高层接口，这个接口使得这一子系统更加容易使用。



**享元模式**——为运用共享技术有效地支持大量细粒度的对象



**代理模式**——为其他对象提供一种代理以控制对这个对象的访问。



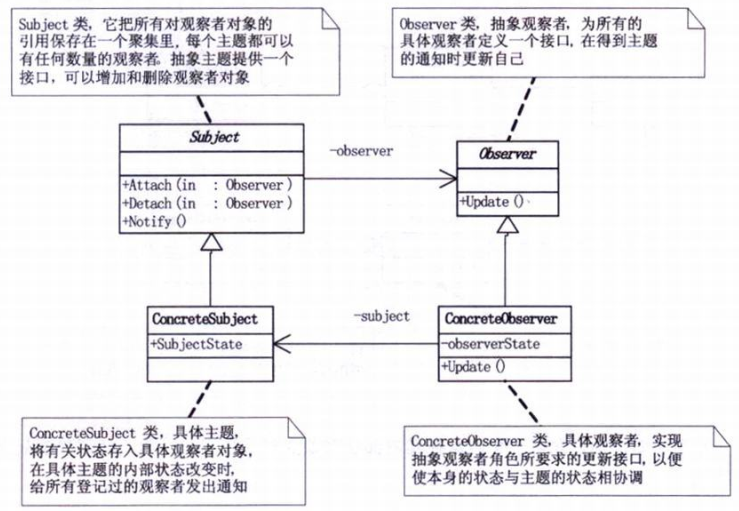
代理-外观区别——代理对象代表一个单一对象而外观对象代表一个子系统；代理的客户对象无法直接访问目标对象，由代理提供对单独的目标对象的访问控制；而外观的客户对象可直接访问子系统中的各个对象，但通常由外观对象提供对子系统各元件功能的简化的共同层次的调用接口。

代理-适配器区别——都属于一种衔接性质的功能。代理是一种原有对象的代表，其他需要与这个对象打交道的操作都是和这个代表交涉。而适配器则不需要虚构出一个代表者，只需为应付特定使用目的，将原来的类进行一些组合。

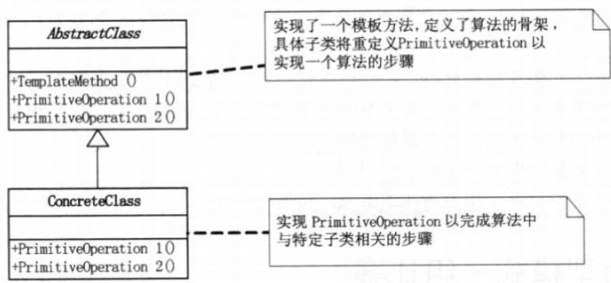
适配器优势——解决两个已有接口之间不匹配的问题，不需考虑接口实现且不需考虑它们各自可能会如何演化。适配器方式不需对两个独立设计的类中任一个进行重新设计，即可使得它们协同工作。

## 行为型模式

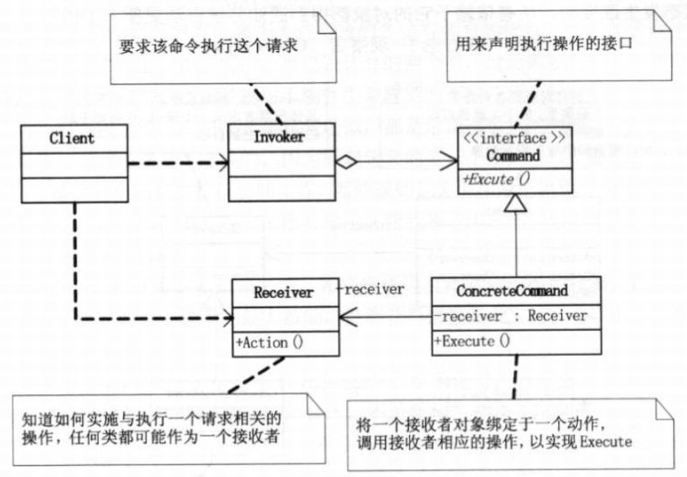
**观察者模式**——定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。



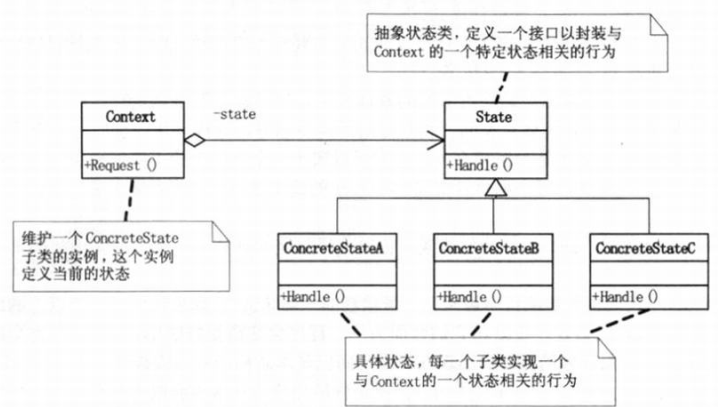
**模板方法模式**——定义一个操作的算法骨架，而将一些步骤延迟到子类中，模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。



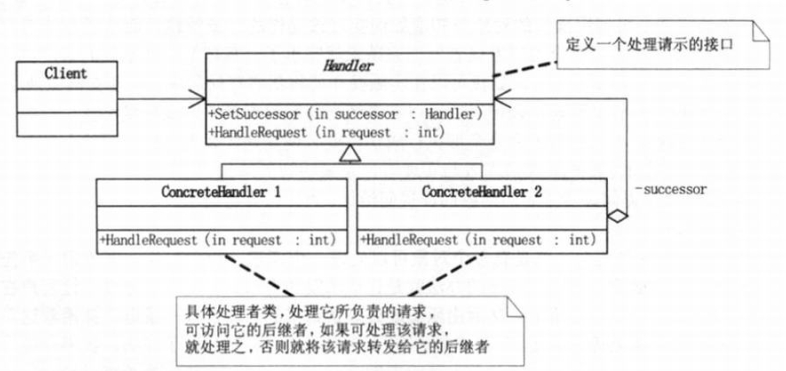
**命令模式**——将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化；可以对请求排队或记录请求日志，以及支持可撤销的操作。



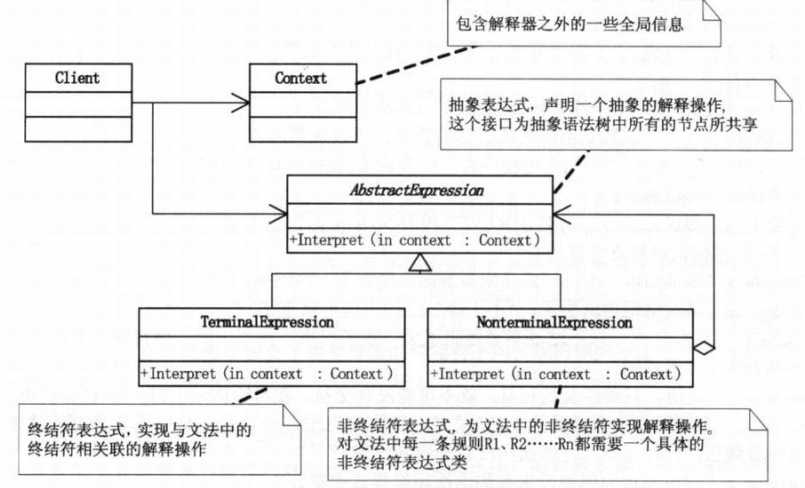
**状态模式**——允许一个对象在其内部转态改变时改变它的行为，让对象看起来似乎修改了它的类。



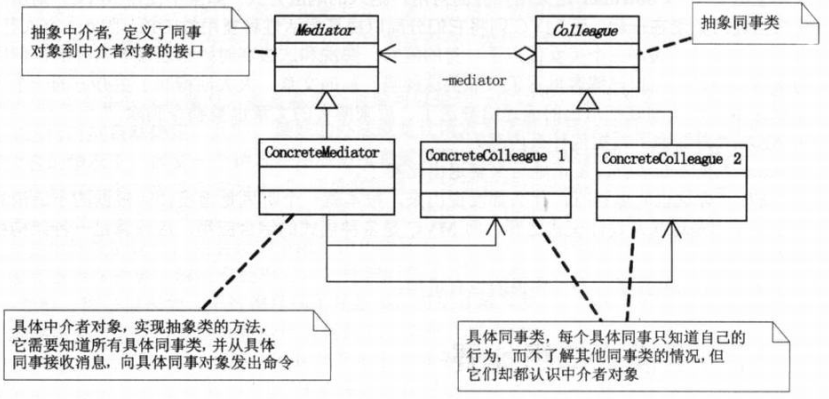
**职责链模式**——使多个对象都有机会处理请求，从而避免请求的发送者和接收者之间的耦合关系。将这些对象连城一条链，并沿着这条链传递该请求，直到有一个对象处理它为止。



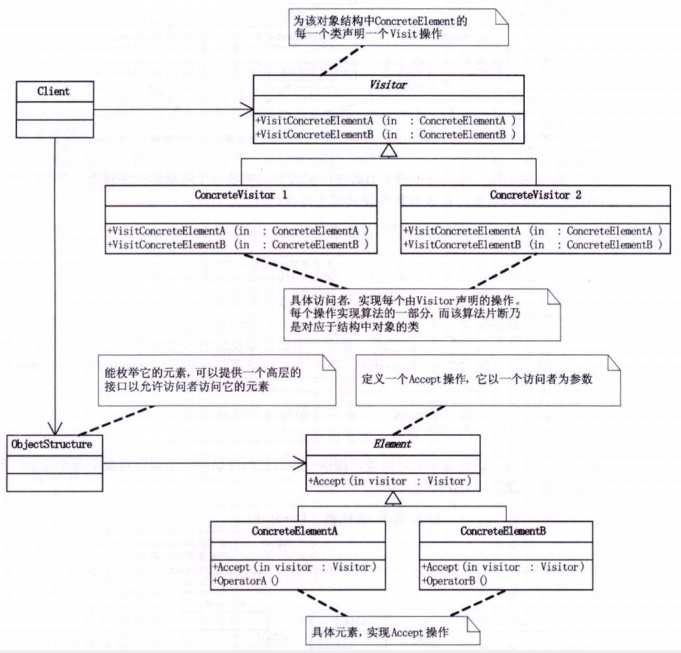
**解释器模式**——给定一个语言，定义它的文法的一种表示，并定义一个解释器，这个解释器使用该表示来解释语言中的句子。



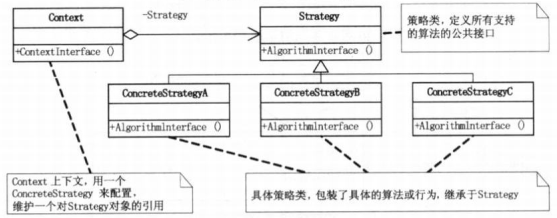
**中介者模式**——用一个中介对象来封装一系列的对象交互。中介者使各对象不需显示的相互引用，从而使其耦合松散，且可独立地改变他们之间的交互。



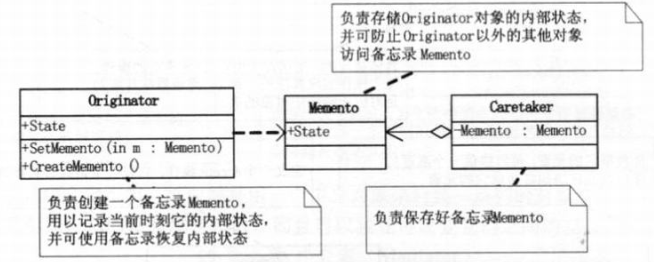
**访问者模式**——表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变元素的类的前提下定义作用于这些元素的新操作。



**策略模式**——定义一系列的算法，把他们一个个封装起来，且使它们可互相替换。本模式使得算法可独立于使用它的客户而变化。



**备忘录模式**——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，



**迭代器模式**——提供一种方法顺序访问一个聚合对象中的各个元素，又不需暴露该对象的内部表示。

