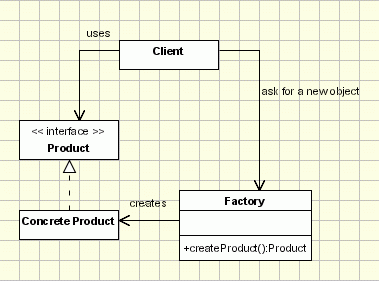
**设计模式 (Design Pattern)**

**简单工厂模式(Simple Factory):**

用一个专门的类或者方法来实例化对象。

工厂类中包含了必要的逻辑判断，去除了客户端与具体产品的依赖。

实例化参数可以是string等与所要实例化的类无关的参数（例如“策略模式+工厂模式”）；或者直接让客户端（调用者）来决定实例化那个类的对象。

****

**工厂方法模式(Factory Method):**

定义一个创建对象的接口，让子类决定实例化哪一个类。

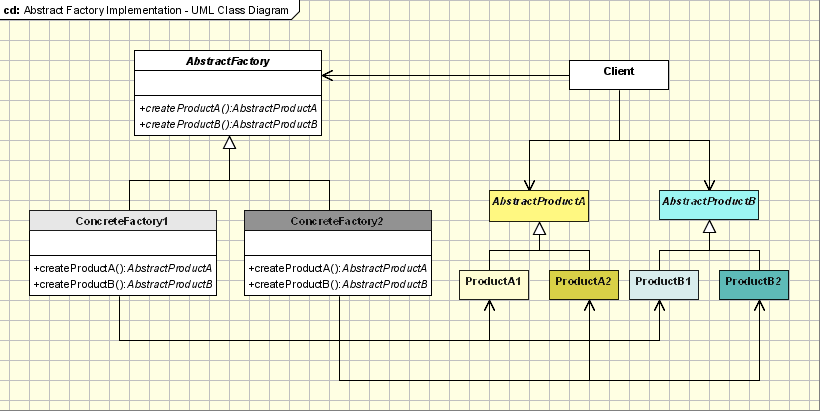
工厂方法使一个类的实例化延迟到其子类。

简单工厂模式 VS 工厂方法模式：简单工厂包含了必要的逻辑判断，去除了客户端与具体产品的依赖；但是与此同时，扩展性降低，比如再增加一个产品，就得修改之前的工厂代码，这违背了“开放-封闭原则”。

工厂方法克服了简单工厂违背开放-封闭原则的缺点，又保持了封装对象创建过程的优点。



**抽象工厂模式(Abstract Factory):**

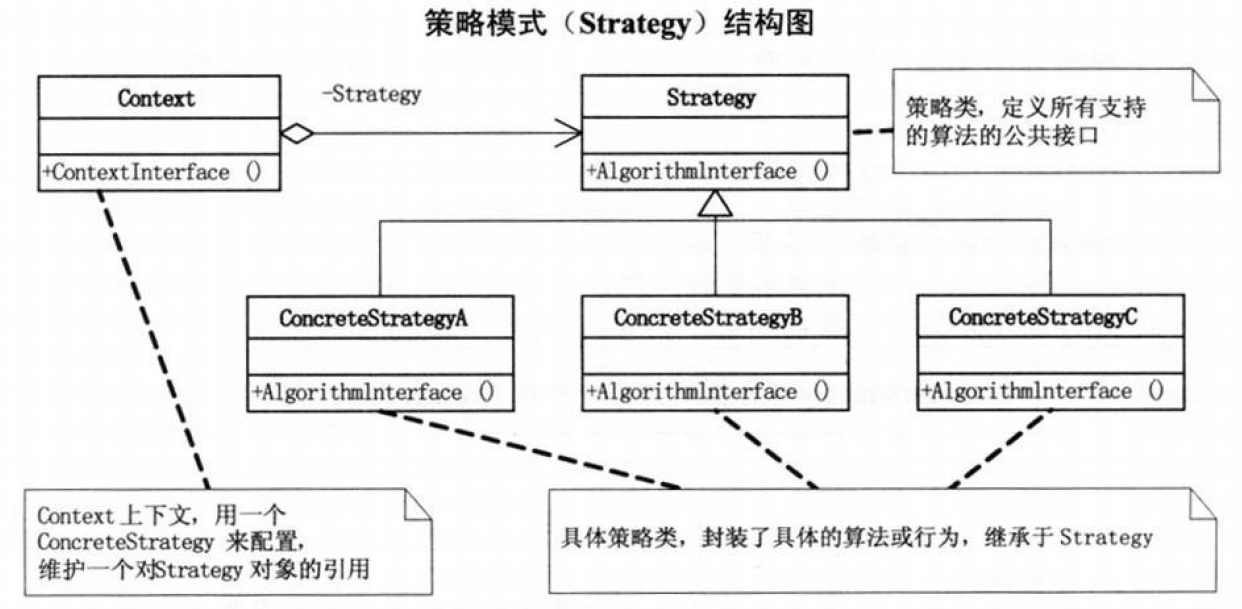


**策略模式(Strategy):**

定义算法家族，算法之间可以相互替换。

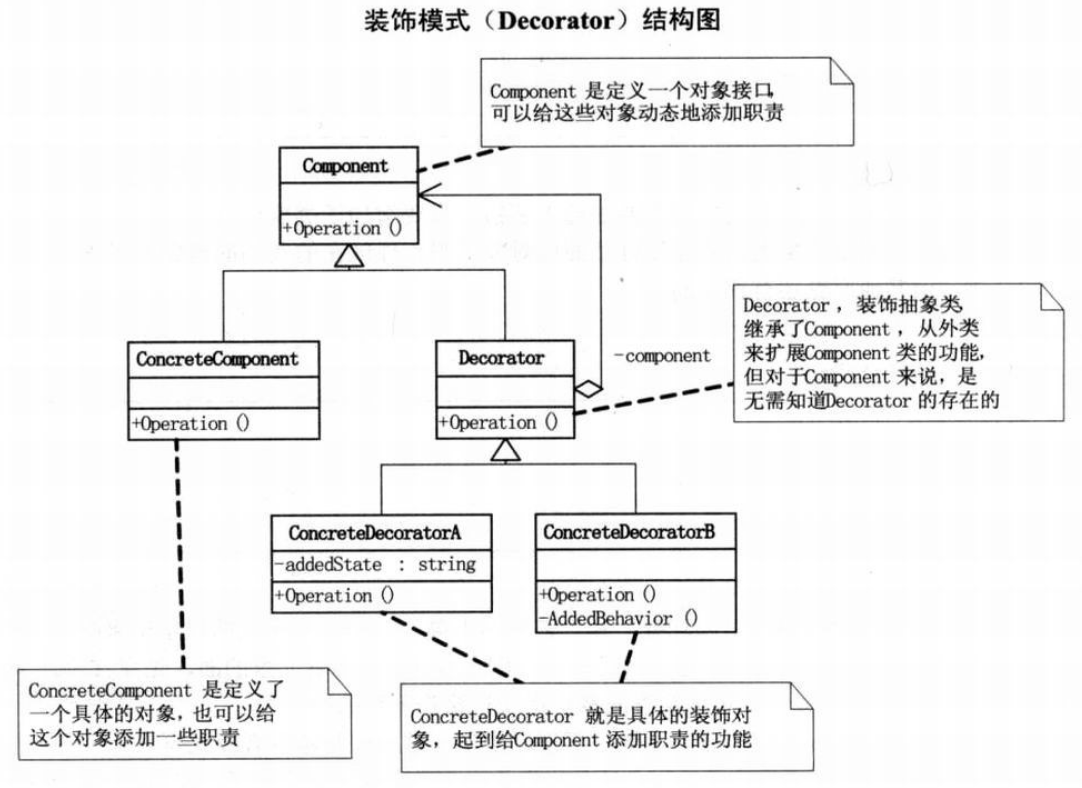
这些算法完成的都是相同的工作（所有method接口/声明式都是一样的），只是实现不同，它可以以相同的方式调用所有的方法，减少了算法类之间的耦合。

（Context类的构造函数参数是子策略；Context类中可以是一个vector<Strategy\*>）

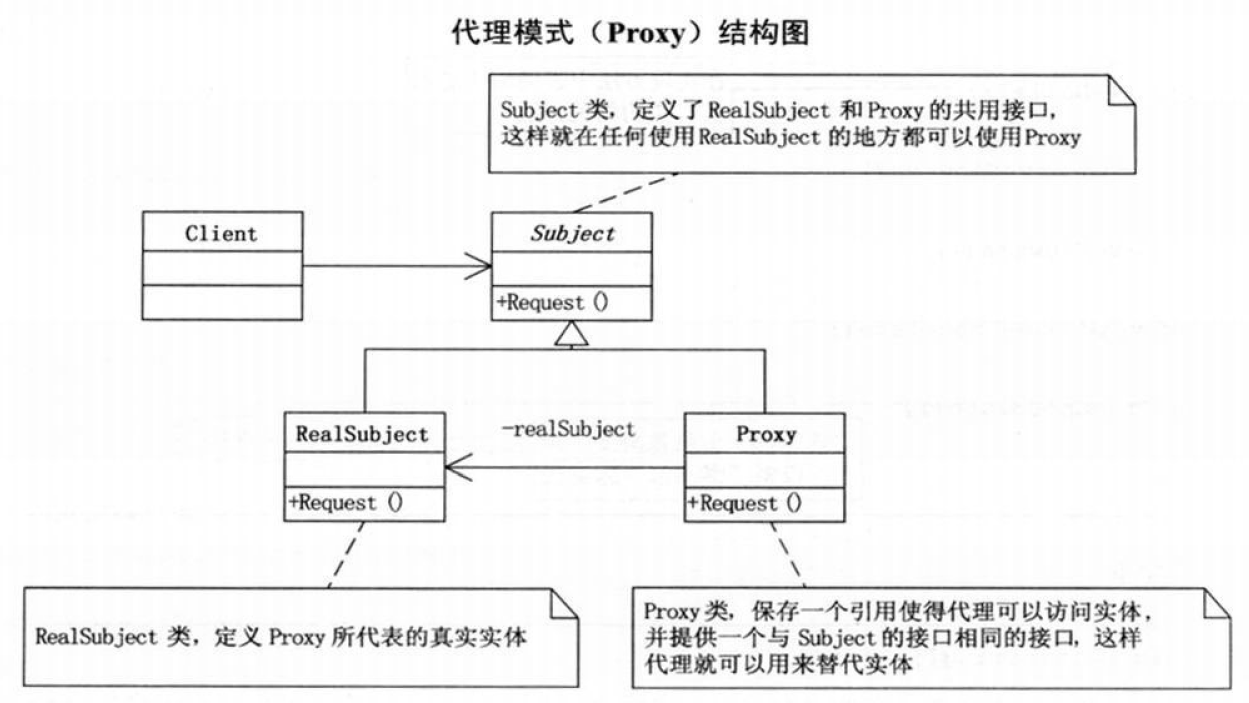


**装饰模式(Decorator):**

把类的核心职责和装饰功能分开。而且可以去除相关类中的重复逻辑。

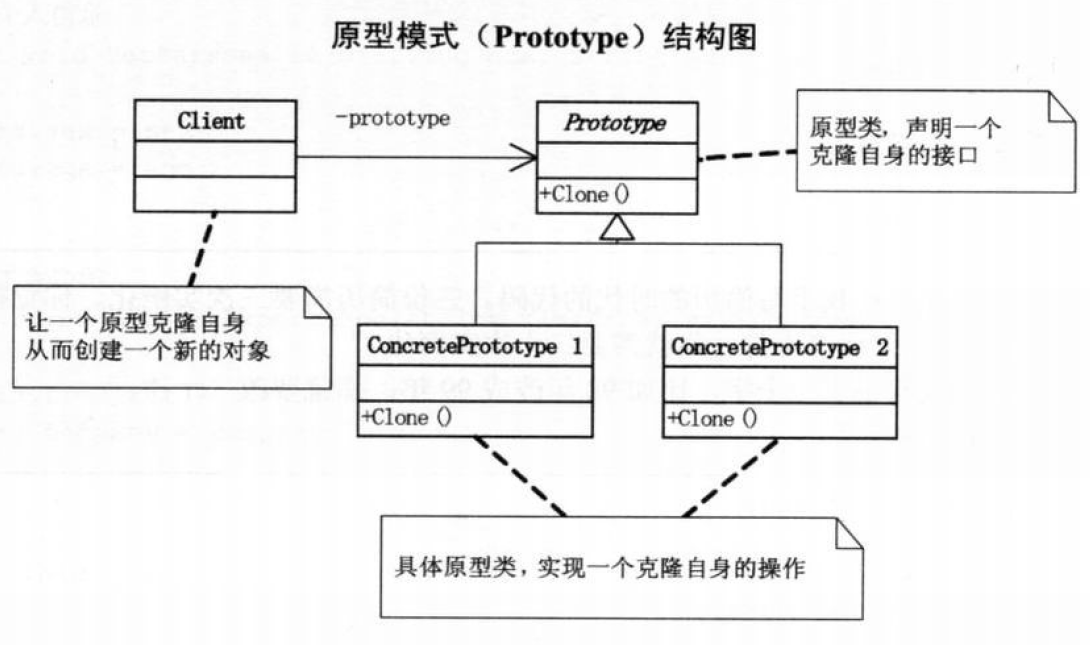


**代理模式(Proxy):**



**原型模式(Prototype)：**

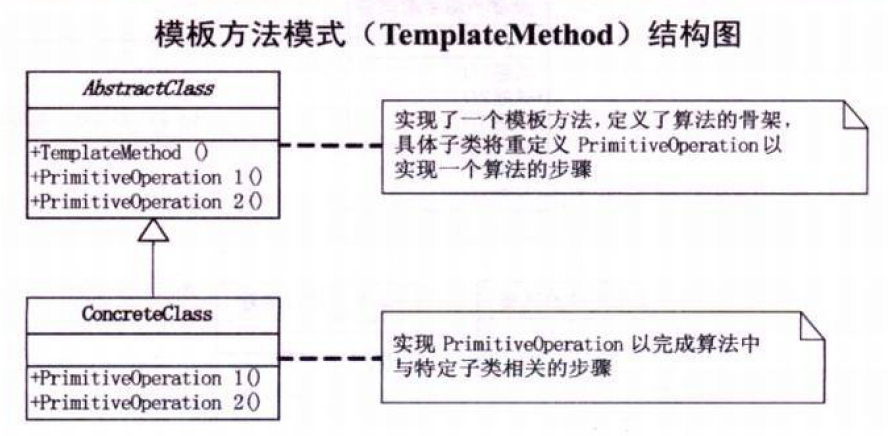
用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象。



**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。

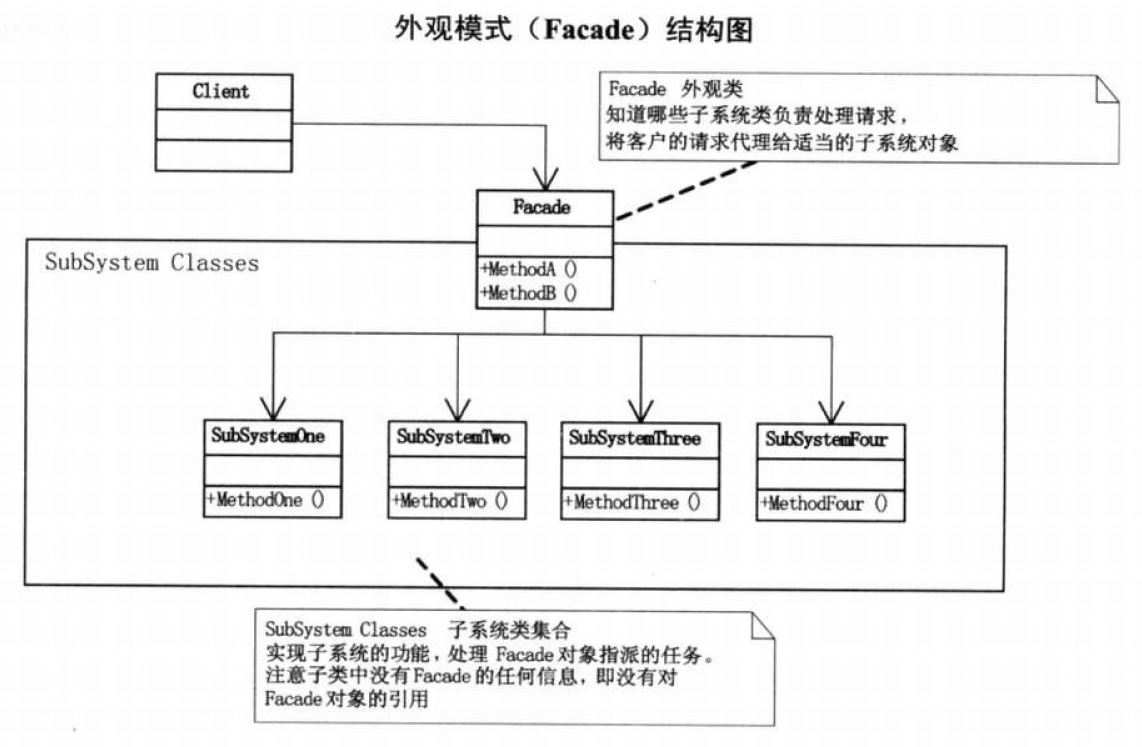
模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。



**外观模式(Façade)：**

为子系统中的一组接口提供一个一致的界面。

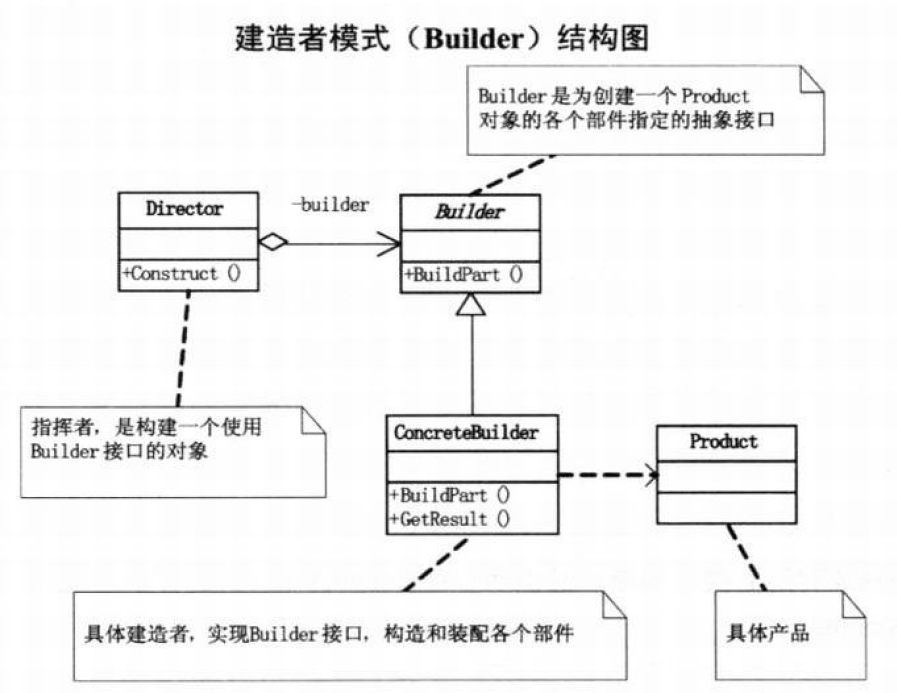
此模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。



**建造者模式(Builder):**

将一个复杂对象的构建与它的表示分离开。

使得同样的构建过程可以构建不同的表示。



**观察者模式(Observer)：**

**也叫发布-订阅模式(Publish-Subsribe).**

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一个主题对象。这个主题对象在状态发生变化时，会通知所有观察者对象，使他们能够自动更新自己。

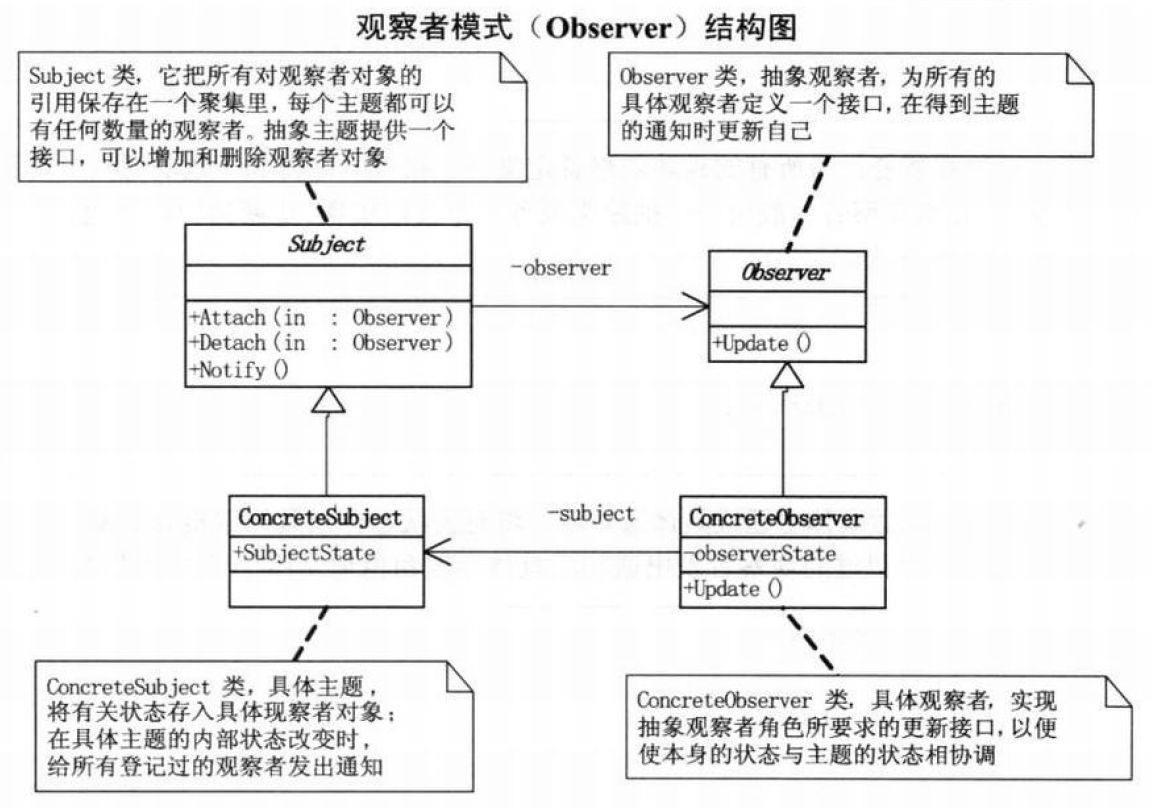
**优点：**观察者模式主要优点是解耦合，让耦合的双方都依赖于抽象，而不是依赖于具体。

当一个对象变化，希望引起其他多个对象作出改变时；

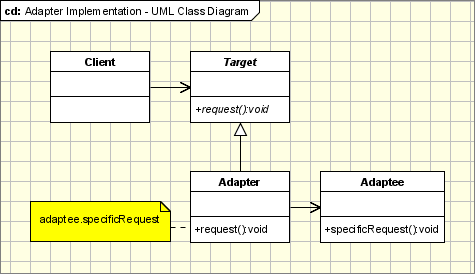
当一个对象不知道具体有多少个对象需要作出改变时；

这是应该使用观察者模式。

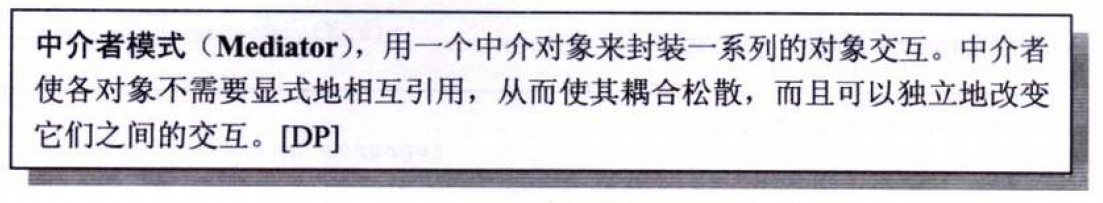
**不足：**发布者和订阅者之间还是相互依赖，只不过是依赖“抽象”。如果发布者和订阅者相互不知道，而要由客户端来决定通知谁，例如不同观察者应对变化的方法名都不相同（不再是Update），那么应该使用事件委托。

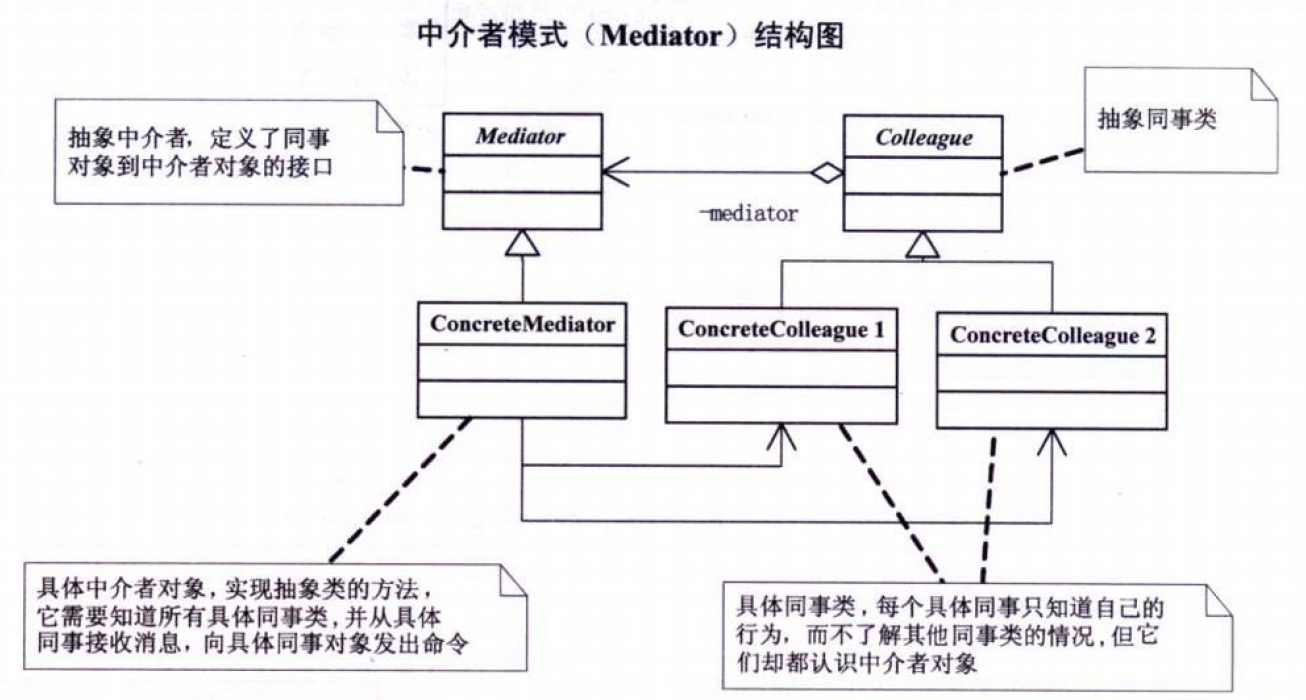


**适配器模式(Adapter)：**

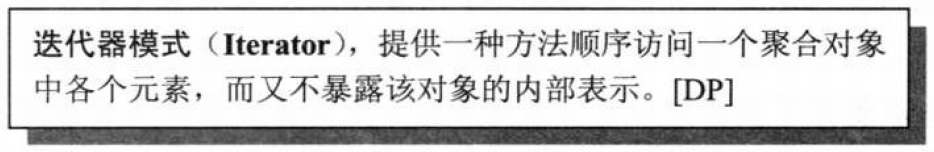


**中介者模式：**





**迭代器模式(Iterator):**

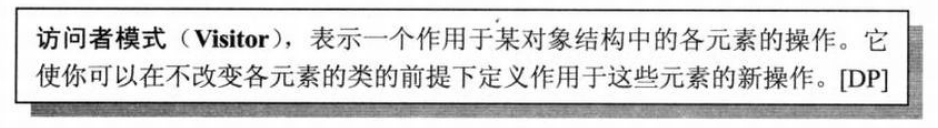


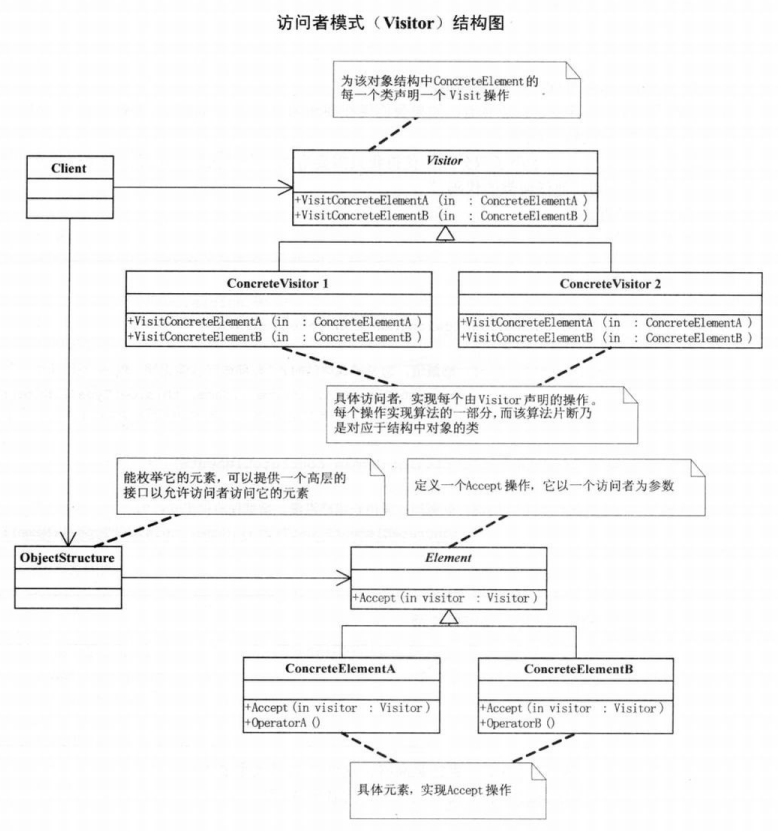
**访问者模式(Visitor)：**

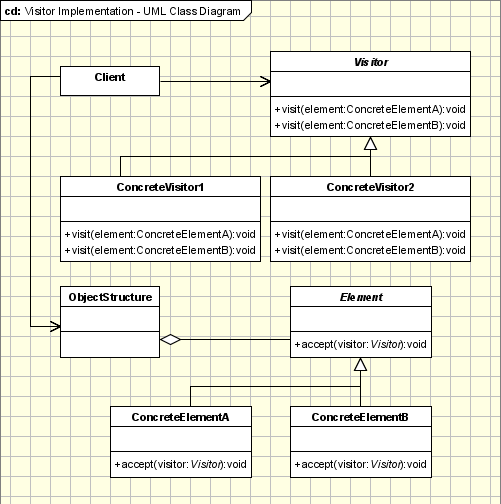
访问者模式适用于数据结构相对稳定的系统；

如果有相对稳定的数据结构(Element)，又有易于变化的算法(Visitor)，那么应该使用访问者模式；

实际上，很少有稳定不变的数据结构，所以访问者模式很少采用。







**准则：**

**单一职责原则(Dependence Inversion Principle)：**

**开放-封闭原则：**

**依赖倒置原则：**

里氏替换原则：任何基类可以出现的地方，子类一定可以出现。

主张使用“抽象(Abstraction)”和“多态(Polymorphism)”将设计中的静态结构改为动态结构，维持设计的封闭性。“抽象”是语言提供的功能。“多态”由继承语义实现。

**迪米特法则：**如果两个类不必彼此直接通信，那么这两个类就不应当发生直接相互的作用。如果一个类需要调用某一个方法，可以通过第三者转发这个调用。