本文UML类图参考《[Head First 设计模式](http://pan.baidu.com/s/1c03tZxu" \t "_blank)》（[源码](https://github.com/meatballhat/HeadFirstDesignPatterns)）与《[设计模式：可复用面向对象软件的基础](http://pan.baidu.com/s/1bnAN51d)》（[源码](https://github.com/TheoAndersen/GoF.Lexi)）两书中介绍的设计模式与UML图。

    整理常用设计模式的类图，一方面是为了练习UML，另一方面可以重新思考设计模式。当然，整理完成后可以作为一份手册供今后翻阅。

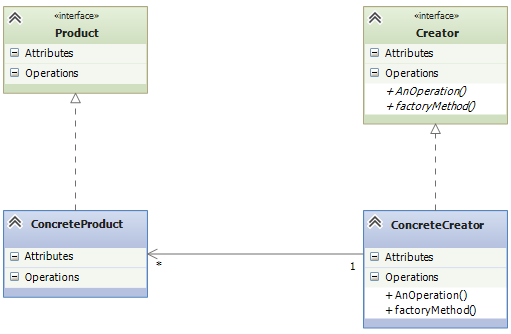
    绘图工具：[Visual Studio 2015](https://www.visualstudio.com/downloads/download-visual-studio-vs" \t "_blank)

一、创建型

1. Factory Method（工厂方法）

*定义：定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个。工厂方法让类把实例化推迟到子类。*

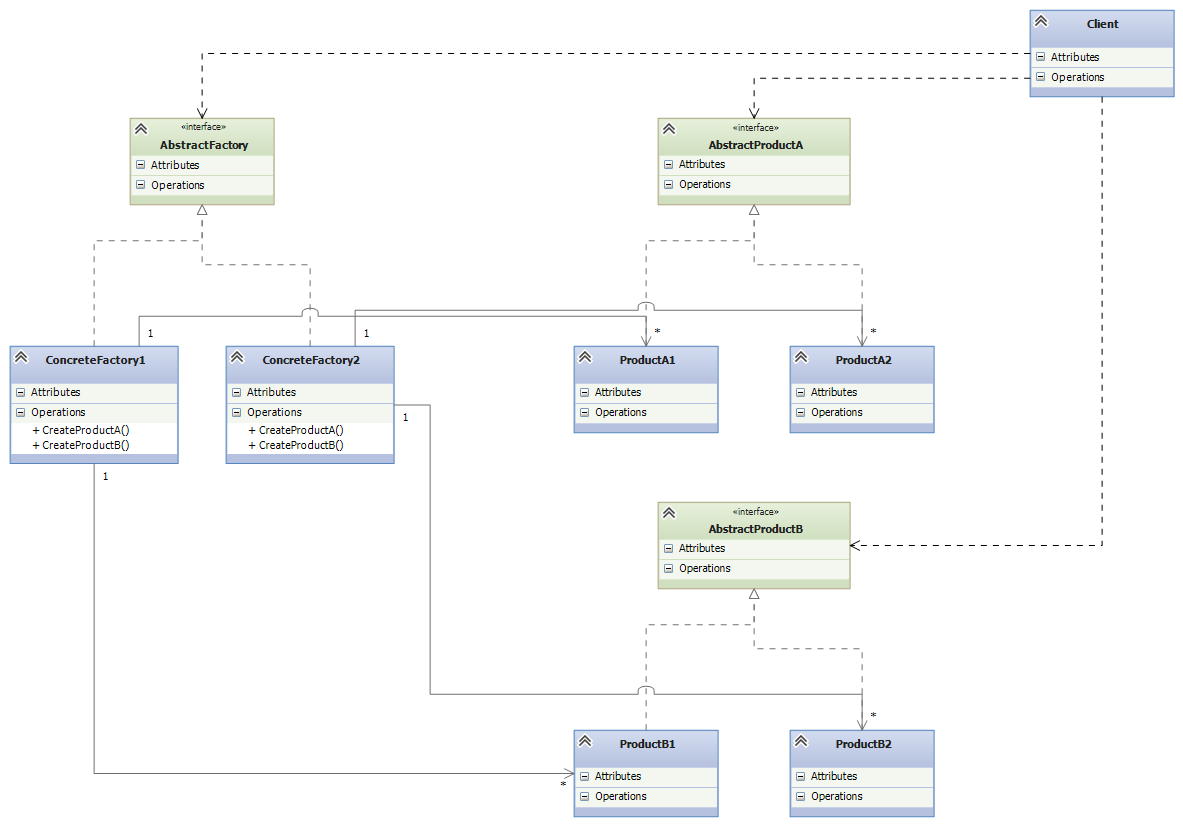
*类图：*



1. Abstract Factory（抽象工厂）

*定义：提供一个接口，用于创建相关或依赖对象的家族，而不需要明确指定具体类。*

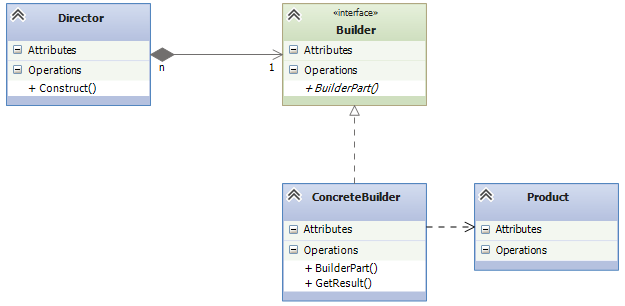
*类图：*

**

1. Builder（生成器模式）

*定义：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。*

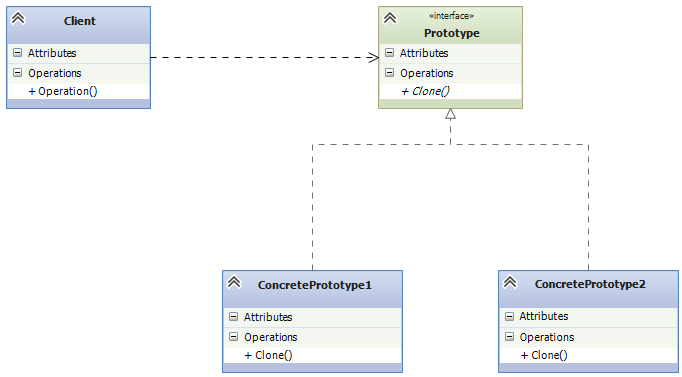
*类图：*

**

1. Prototype（原型模式）

*定义：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。*

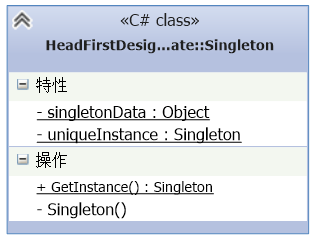
*类图：*

**

1. Singleton（单件模式）

*定义：确保一个类只有一个实例，并提供全局访问点。*

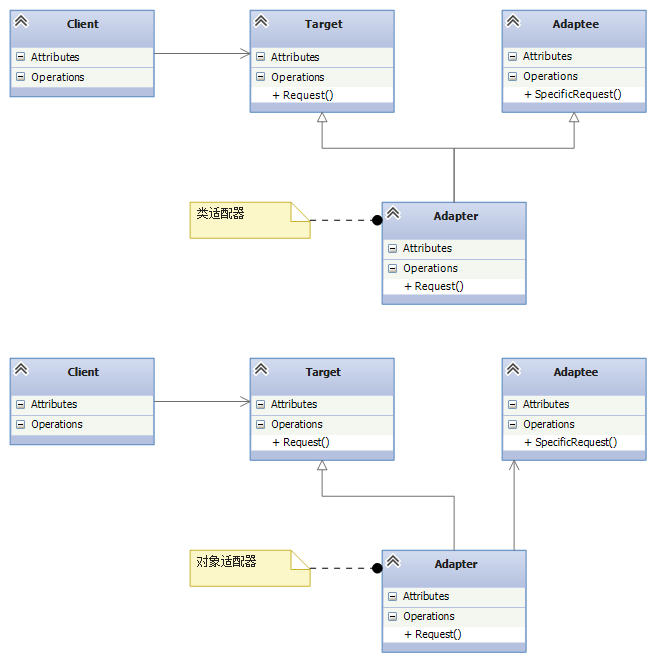
*类图：*

**

二、结构性

1. Adapter（适配器模式（类、对象））  
   *定义：将一个类的接口转换成客户期望的另一个接口。适配器让原本接口不兼容的类可以合作无间。*

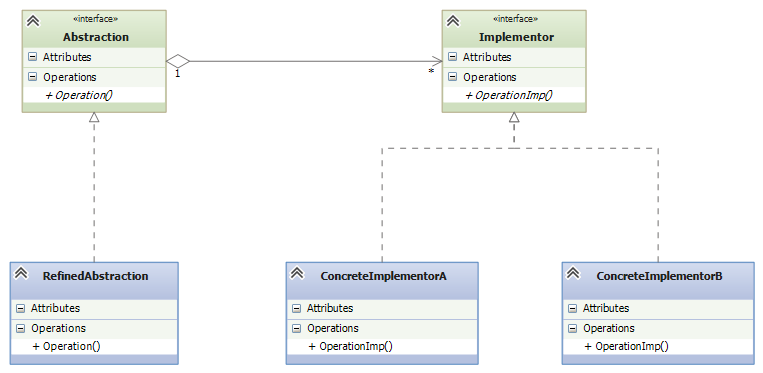
*类图：*

**

1. Bridge（桥连模式）

*定义：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。*

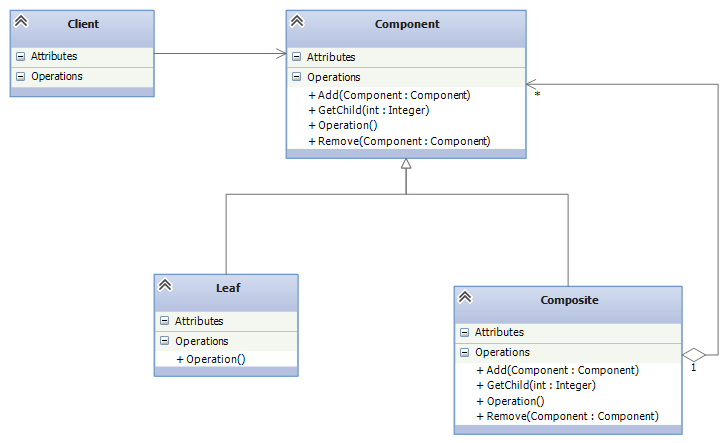
*类图：*

**

1. Composite（组合模式）

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

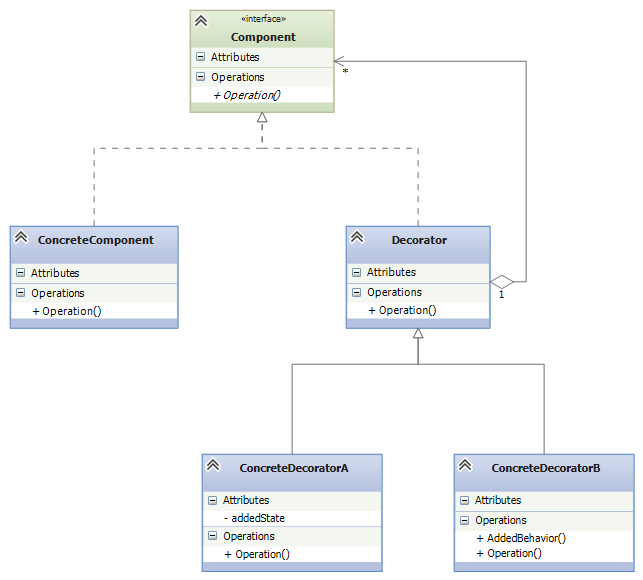
*类图：*

**

1. Decorator（装饰模式）

*定义：动态地给一个对象添加一些额外的职责。就增加功能来说，Decorator模式相比生成子类更加灵活。*

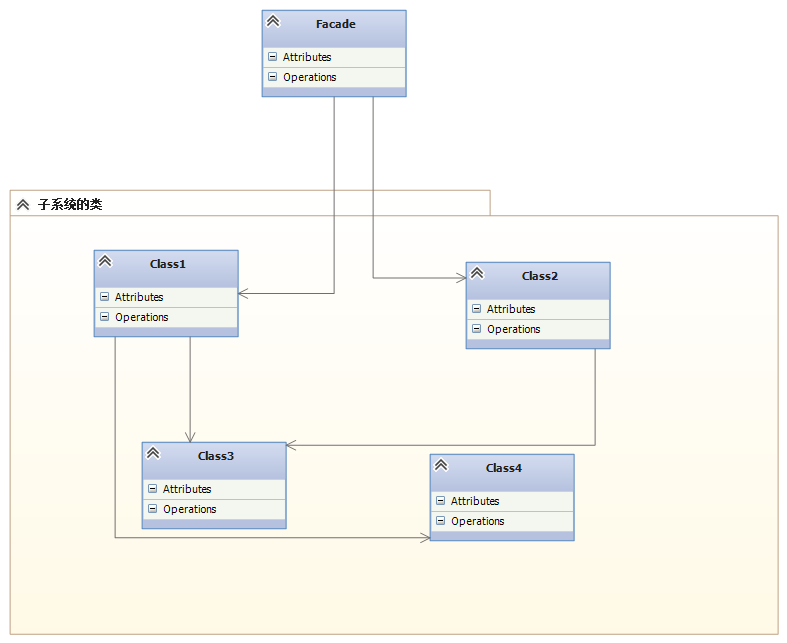
*类图：*

**

1. Facade（外观模式）

*定义：为子系统中的一组接口提供一个一致的界面，Facade模式定义了一个高级接口，这个接口使得这一子系统更加容易使用。*

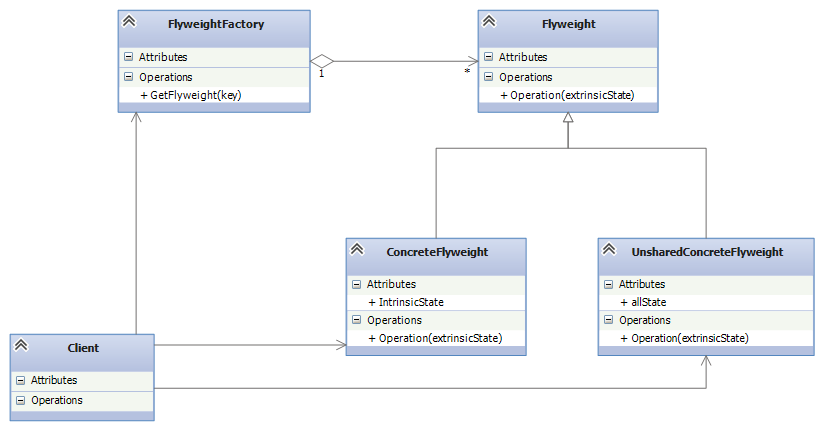
*类图：*

**

1. Flyweight（享元模式）

*定义：运用共享技术有效地支持大量细粒度的对象。*

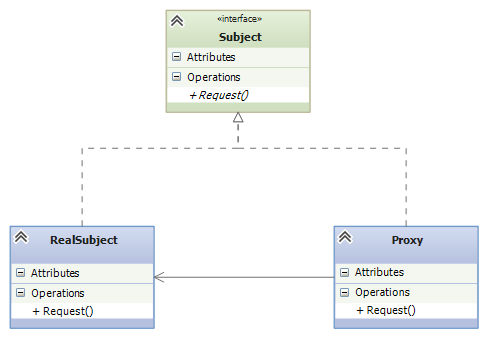
*类图：*

**

1. Proxy（代理模式）

*定义：为其他对象提供一种代理以控制对这个对象的访问。*

*类图：*

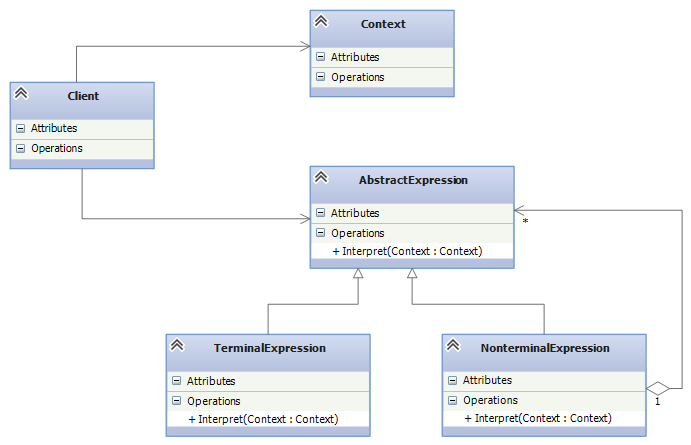
**

三、行为型

1. Interpreter（解释器模式）

*定义：给定一个语言，定义它的一种表示，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。*

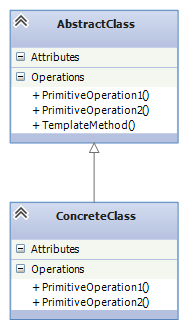
*类图：*

**

1. Template Method（模板方法）

*定义：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。TemplateMethod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。*

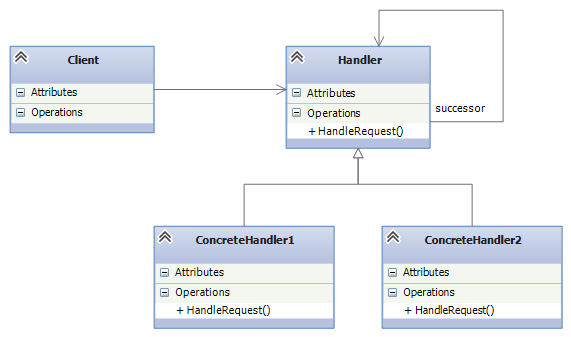
*类图：*

**

1. Chain of Responsibility（职责链模式）

*定义：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受者之间的耦合关系。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。*

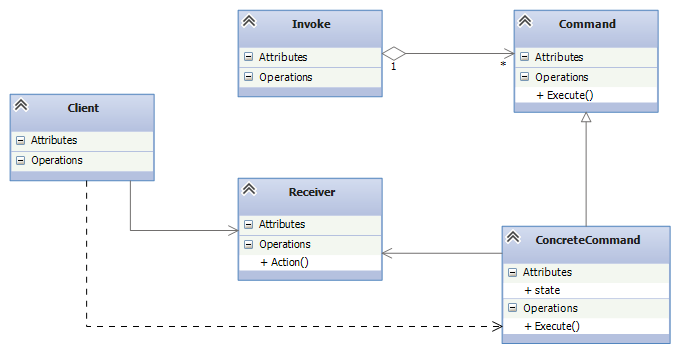
*类图：*

**

1. Command（命令模式）

*定义：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；对请求排队或记录请求日志，以及支持可撤销的操作。*

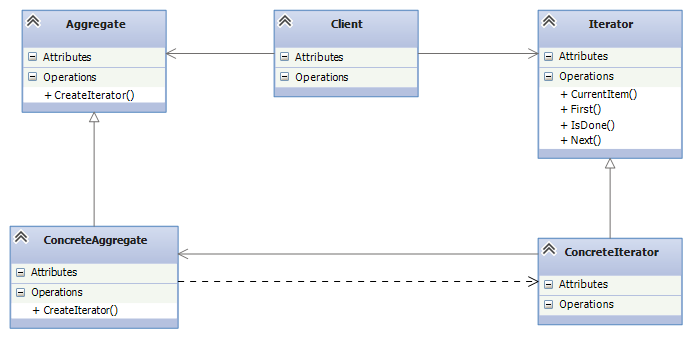
*类图：*

**

1. Iterator（迭代模式）

*定义：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。*

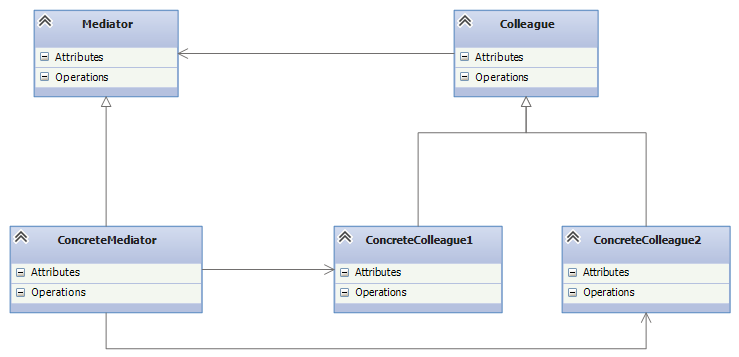
*类图：*

**

1. Mediator（中介模式）

*定义：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示地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。*

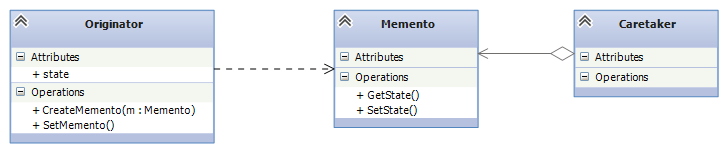
*类图：*

**

1. Memento（备忘录模式）

*定义：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。*

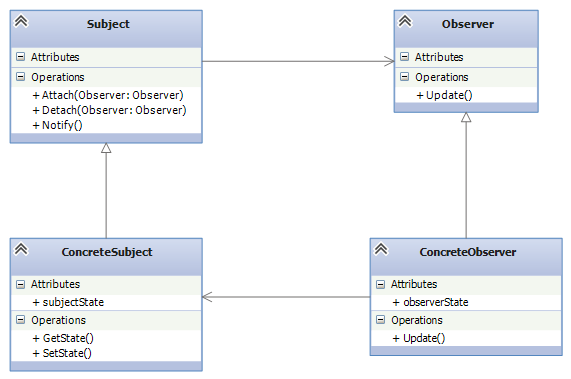
*类图：*

**

1. Observer（观察者模式）

*定义：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变化时，所有依赖于它的对象都得到通知并被自动更新。*

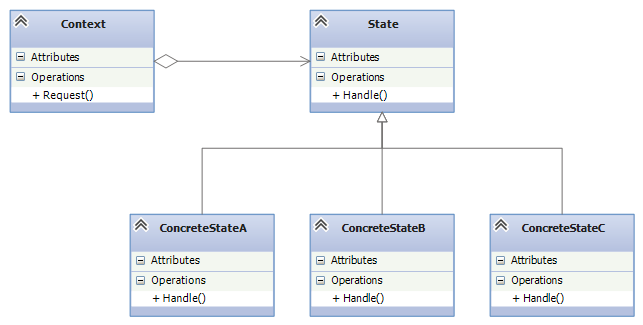
*类图：*

**

1. State（状态模式）

*定义：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。*

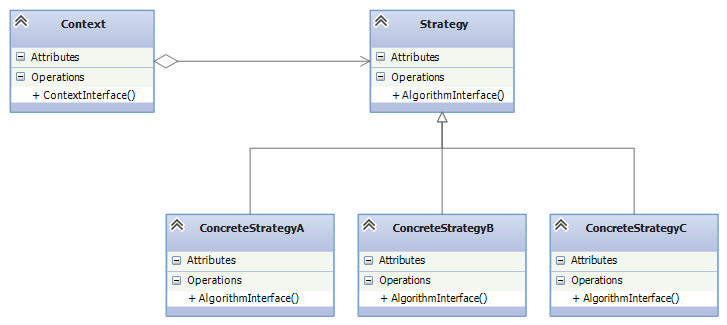
*类图：*

**

1. Strategy（策略模式）

*定义：定义一系列的算法，把它们一个个封装起来，并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。*

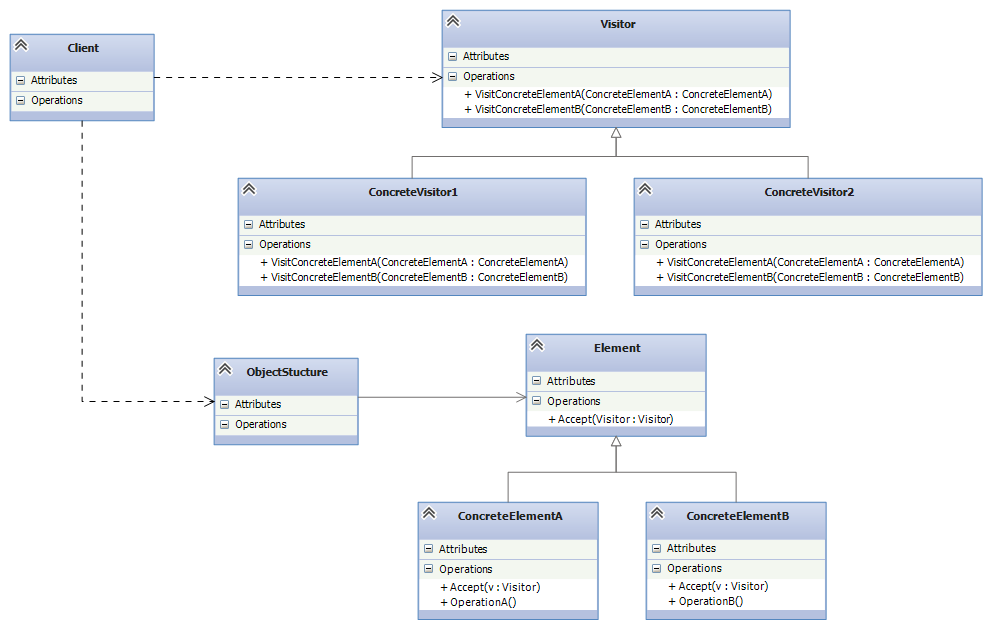
*类图：*

**

1. Visitor（访问者模式）

*定义：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的前提下定义作用于这些元素的新操作。*

*类图：*

**