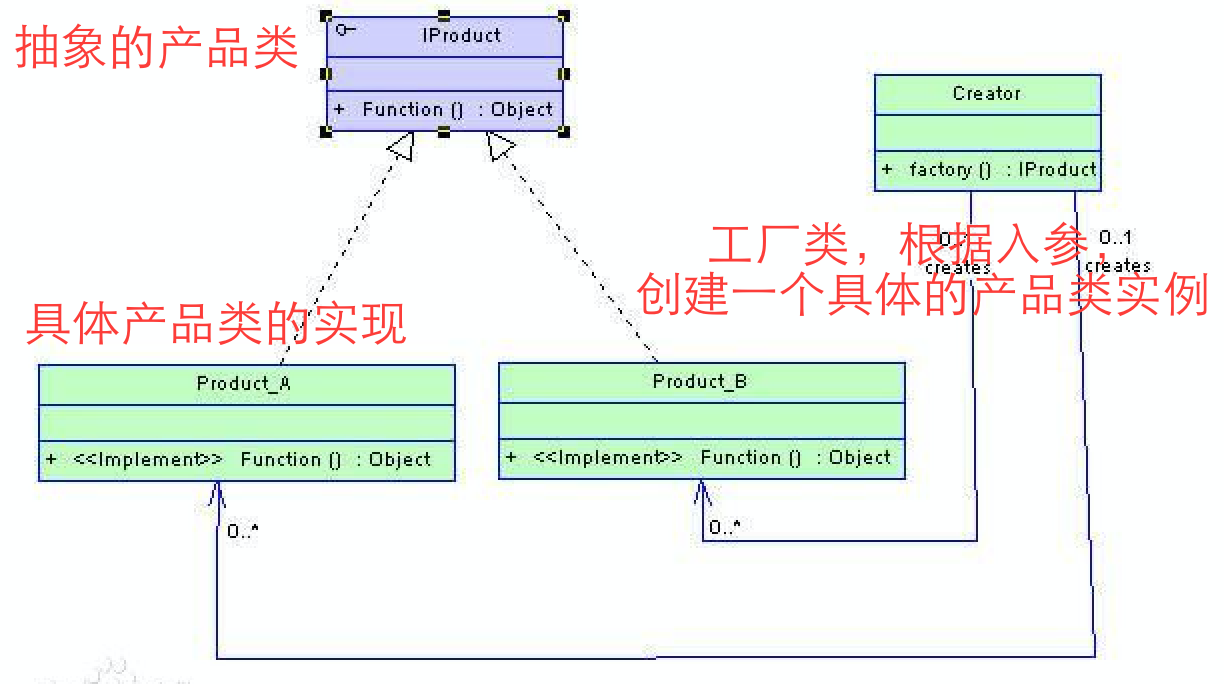
Java 设计模式

创建型模式

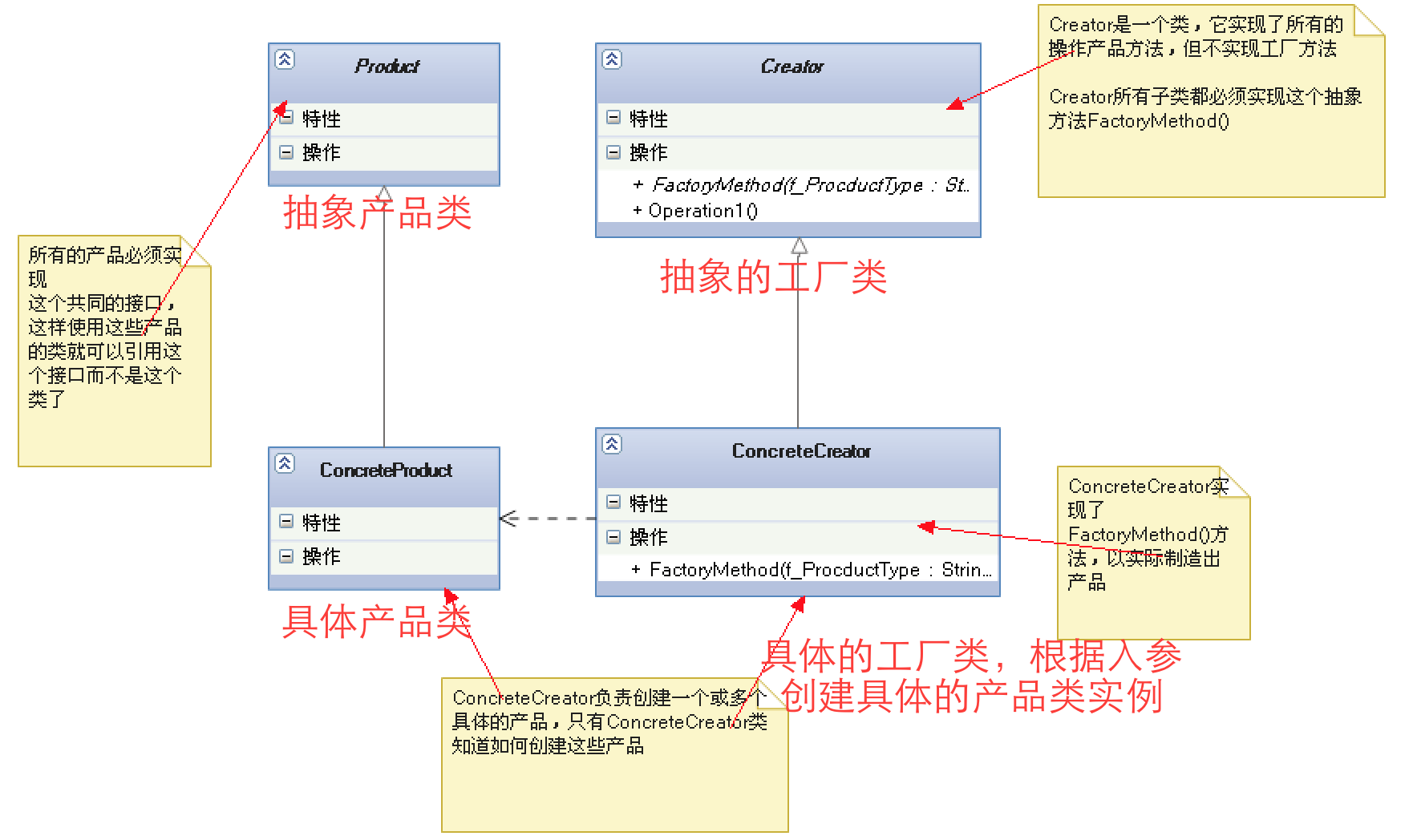
（1）简单工厂模式 该模式不是23种设计模式之一



简单工厂模式中工厂类创建具体产品的方法一般是静态的

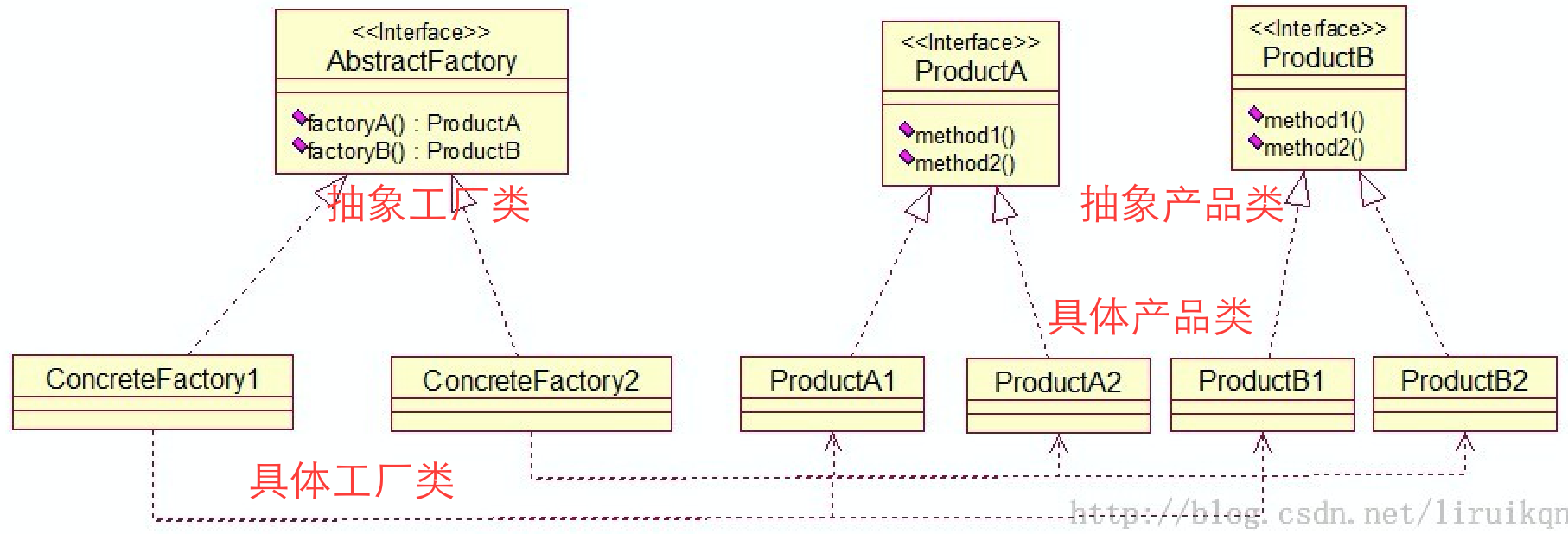
创建一个产品实例的方法：（1）实例化一个工厂类，返回一个具体的产品实例

（2）工厂方法模式 属于23种设计模式之一，属于创建型模式



创建一个产品实例：（1）先实例化一个具体的工厂类 ，返回一个具体的产品类实例

（3）抽象工厂模式，属于23中设计模式之一，属于创建型模式

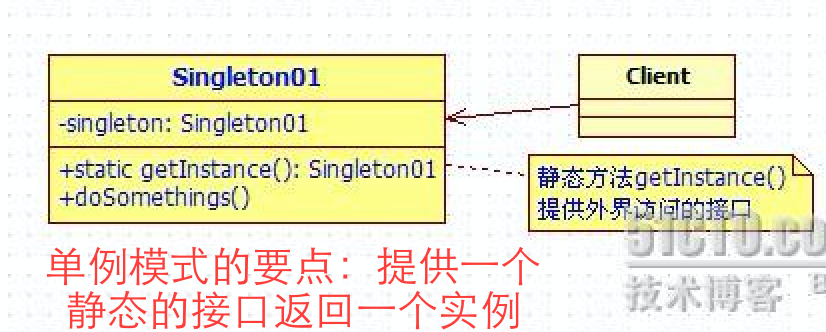


抽象工厂模式与工厂方法 模式区别：（1）工厂方法模式中具体工厂类只能创建一个产品类

                                                           （2）抽象工厂模式中具体工厂类可以创建多个产品类实例（有一个产品族概念）

创建多个产品实例：（1）先实例化一个具体的工厂类，返回多个具体的产品类实例

(4) 单例模式 ，属于23种设计模式之一，属于创建型模式



    单例要点：（1）定义一个private类型的构造方法  （2）定义一个private static类型的实例 （3）提供一个public static 的接口返回类的实例

   单例模式有几种模式：（1）懒汉式 ，没有同步锁，有线程安全问题

                                       （2）饿汉式，类加载时初始化，没有懒加载

                                       （3）内部类，创建一个内部类，在获取实例时初始化内部类，内部类是private static接口

                                       （4）双重校验锁

                                       （5）枚举型

                                                        public enum Singleton {

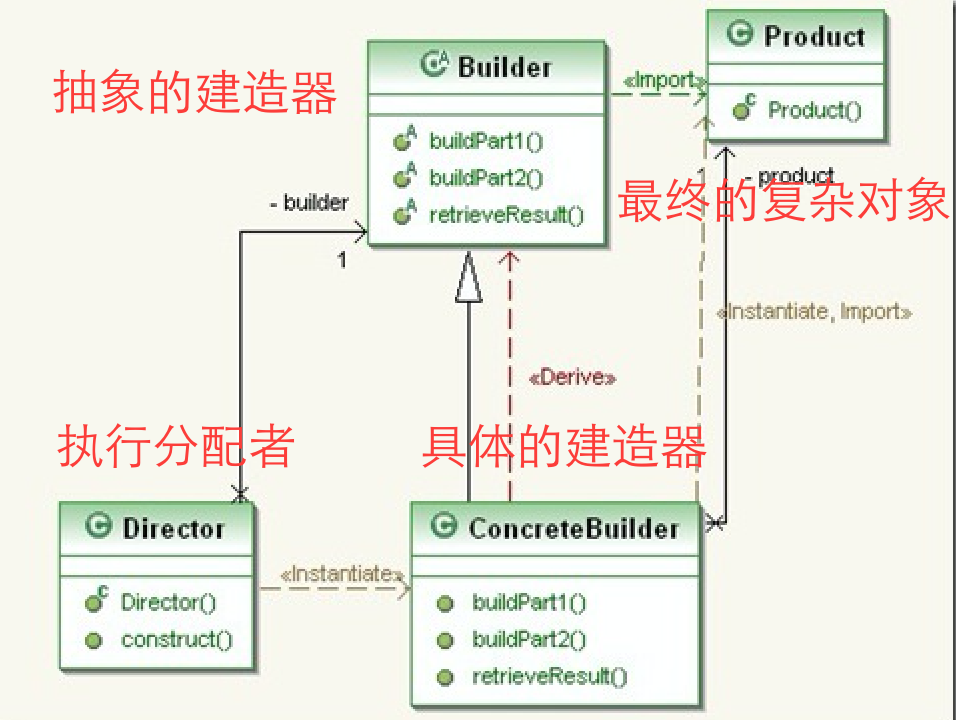
                                                                 INSTANCE;

                                                                 public void whateverMethod() {

                                                                 }

                                                        }

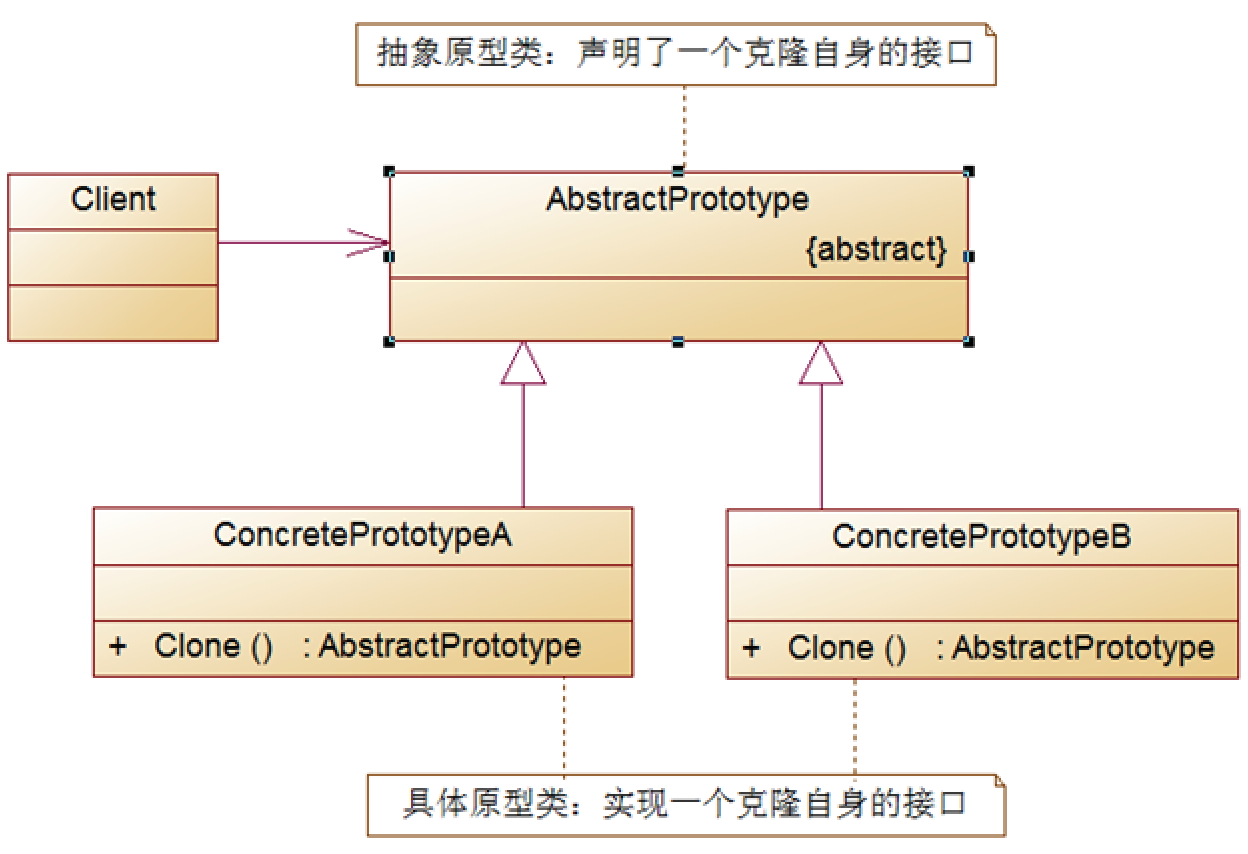
（5）builder 建造模式，属于23种设计模式之一，属于创建型模式



定义一个product接口，定义一个构建器接口、具体的构建器实例、通过导演者返回具体的product实例

构建器类要定义如何返回product类实例及该product类的属性

（6）protoType 模式，属于23中设计模式之一，属于创建型模式



 原型模式: 利用clone方法实现，有两种clone，深度复制和浅度复制

     深度复制：类里面对象的引用也被clone,两个对象完全独立，不会有影响

    浅度复制：类里面的对象引用不会被clone,导致引用的内存地址一样，一个地方更改，都会变动

行为型模式

（7）迭代器模式，属于行为模式

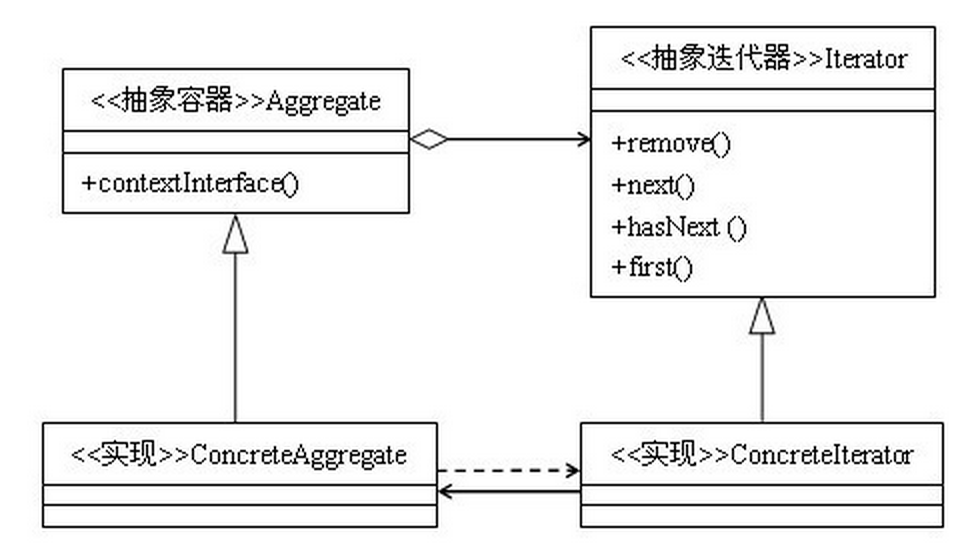
     迭代器模式由以下角色组成：

     迭代器角色（Iterator）: 负责定义访问和遍历元素的接口。

     具体迭代器角色（Concrete Iterator）：实现迭代器接口，并要记录遍历中的当前位置。

     容器角色(Container):  负责提供创建具体迭代器角色的接口。

     具体容器角色（Concrete Container）：实现创建具体迭代器角色的接口， 这个具体迭代器角色与该容器的结构相关。



**迭代器模式的适用场景**

       迭代器模式是与集合共生共死的，一般来说，我们只要实现一个集合，就需要同时提供这个集合的迭代器，就像java中的Collection，List、Set、Map等，这些集合都有自己的迭代器。假如我们要实现一个这样的新的容器，当然也需要引入迭代器模式，给我们的容器实现一个迭代器。

       但是，由于容器与迭代器的关系太密切了，所以大多数语言在实现容器的时候都给提供了迭代器，并且这些语言提供的容器和迭代器在绝大多数情况下就可以满足我们的需要，所以现在需要我们自己去实践迭代器模式的场景还是比较少见的，我们只需要使用语言中已有的容器和迭代器就可以了

  (8) Observer 观察者模式

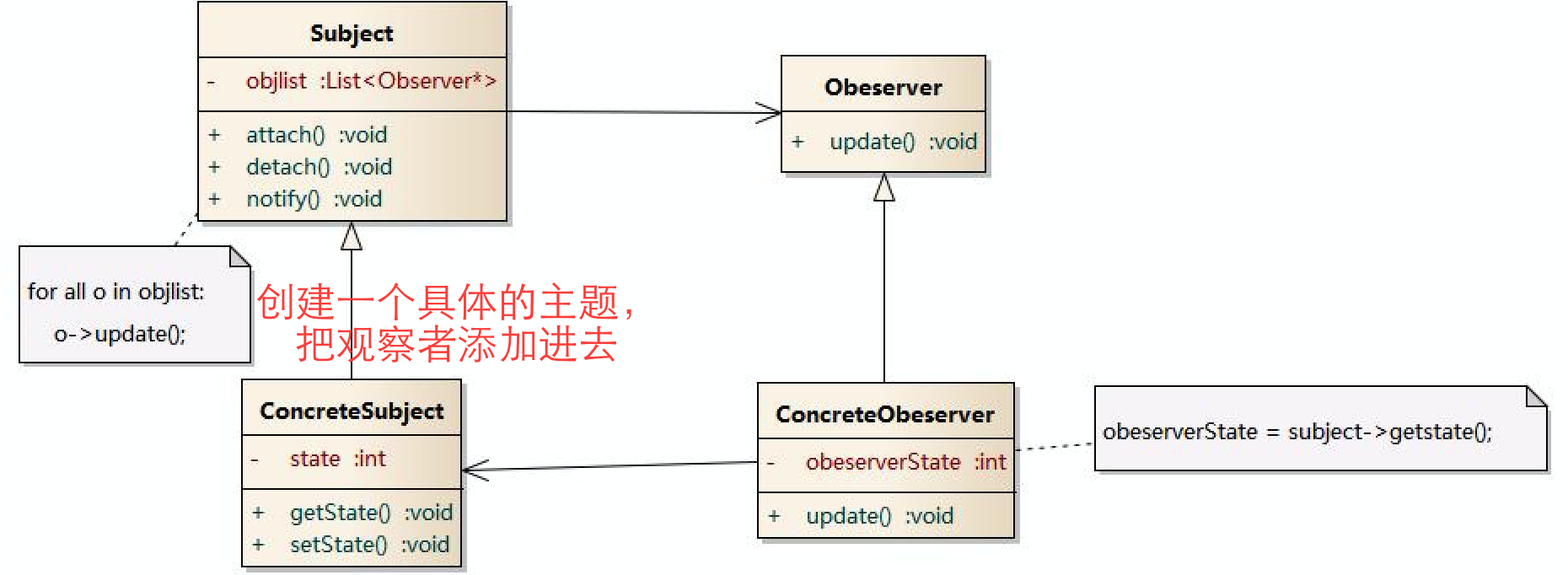
     观察者模式的角色：

     Subject: 主题接口

     ConcreteSubject: 主题实现

     Observer: 观察者

     ConcreteObserver: 具体观察者



* 观察者模式定义对象间的一种一对多依赖关系，使得每当一个对象状态发生改变时，其相关依赖对象皆得到通知并被自动更新。观察者模式又叫做发布-订阅模式、模型-视图模式、源-监听器模式或从属者模式。观察者模式是一种对象行为型模式。
* 观察者模式包含四个角色：目标又称为主题，它是指被观察的对象；具体目标是目标类的子类，通常它包含有经常发生改变的数据，当它的状态发生改变时，向它的各个观察者发出通知；观察者将对观察目标的改变做出反应；在具体观察者中维护一个指向具体目标对象的引用，它存储具体观察者的有关状态，这些状态需要和具体目标的状态保持一致。

（9）模板方法模式

**抽象模板(Abstract Template)角色**有如下责任：

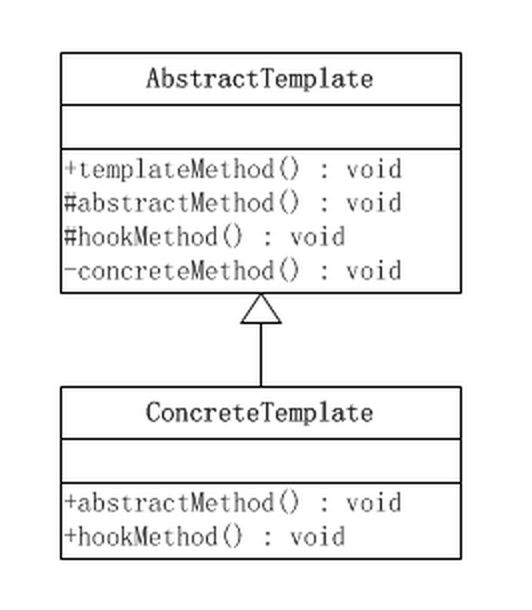
    定义了一个或多个抽象操作，以便让子类实现。这些抽象操作叫做基本操作，它们是一个顶级逻辑的组成步骤。

    定义并实现了一个模板方法。这个模板方法一般是一个具体方法，它给出了一个顶级逻辑的骨架，而逻辑的组成步骤在相应的抽象操作中，推迟到子类实现。顶级逻辑也有可能调用一些具体方法。

**具体模板(Concrete Template)角色**又如下责任：

     实现父类所定义的一个或多个抽象方法，它们是一个顶级逻辑的组成步骤。

     每一个抽象模板角色都可以有任意多个具体模板角色与之对应，而每一个具体模板角色都可以给出这些抽象方法（也就是顶级逻辑的组成步骤）的不同实现，从而使得顶级逻辑的实现各不相同。



  思想：设计一个抽象的类，实现核心接口，核心接口包括多个抽象的方法，抽象的方法组成核心接口，可以利用钩子方法，定义哪些抽象方法被使用，并且核心接口一般使用final修饰，继承时，不会被修改。

（10）Command 命令模式

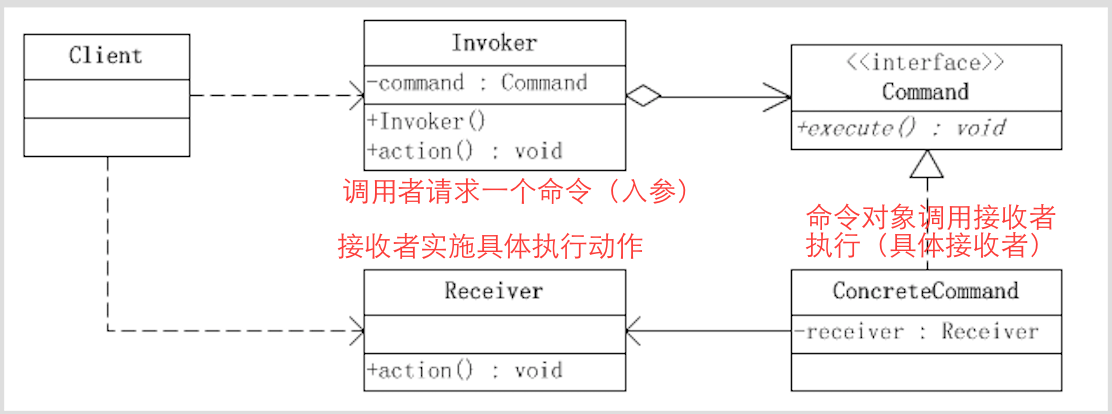
          命令模式涉及到五个角色，它们分别是：

**客户端(Client)角色：**创建一个具体命令(ConcreteCommand)对象并确定其接收者。

**命令(Command)角色：**声明了一个给所有具体命令类的抽象接口。

**具体命令(ConcreteCommand)角色：**定义一个接收者和行为之间的弱耦合；实现execute()方法，负责调用接收者的相应操作。execute()方法通常叫做执行方法。

**请求者(Invoker)角色：**负责调用命令对象执行请求，相关的方法叫做行动方法。

**接收者(Receiver)角色：**负责具体实施和执行一个请求。任何一个类都可以成为接收者，实施和执行请求的方法叫做行动方法。   

   客户端：（1）实例化一个Invoker对象，实例化一个Command对象、实例化一个Receiver对象

                 （2）invoker对象调用action方法，invoker.action—>command.execute—>receiver.action

                    receiver是最终执行命令的主体。

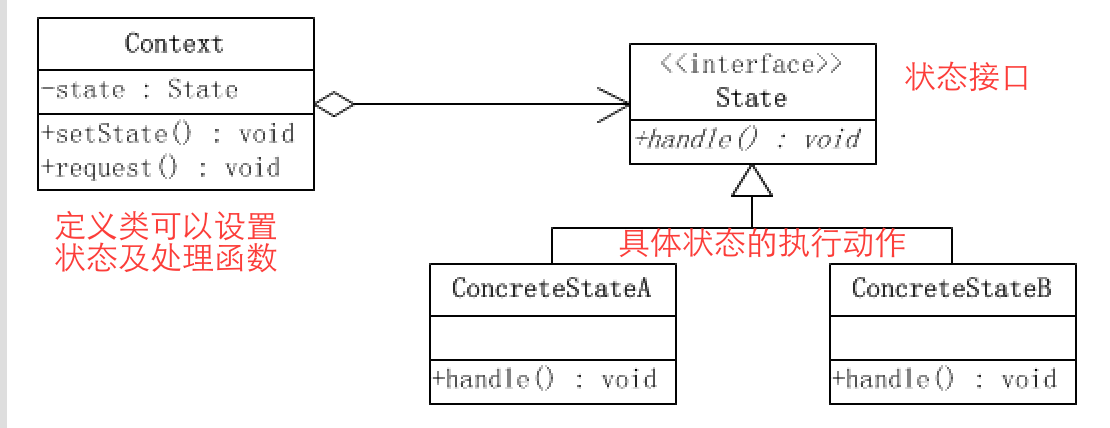
     （11）State，状态模式

     状态模式所涉及到的角色有：

          环境(Context)角色，也成上下文：定义客户端所感兴趣的接口，并且保留一个具体状态类的实例。这个具体状态类的实例给出此环境对象的现有状态。

          抽象状态(State)角色：定义一个接口，用以封装环境（Context）对象的一个特定的状态所对应的行为。

          具体状态(ConcreteState)角色：每一个具体状态类都实现了环境（Context）的一个状态所对应的行为。



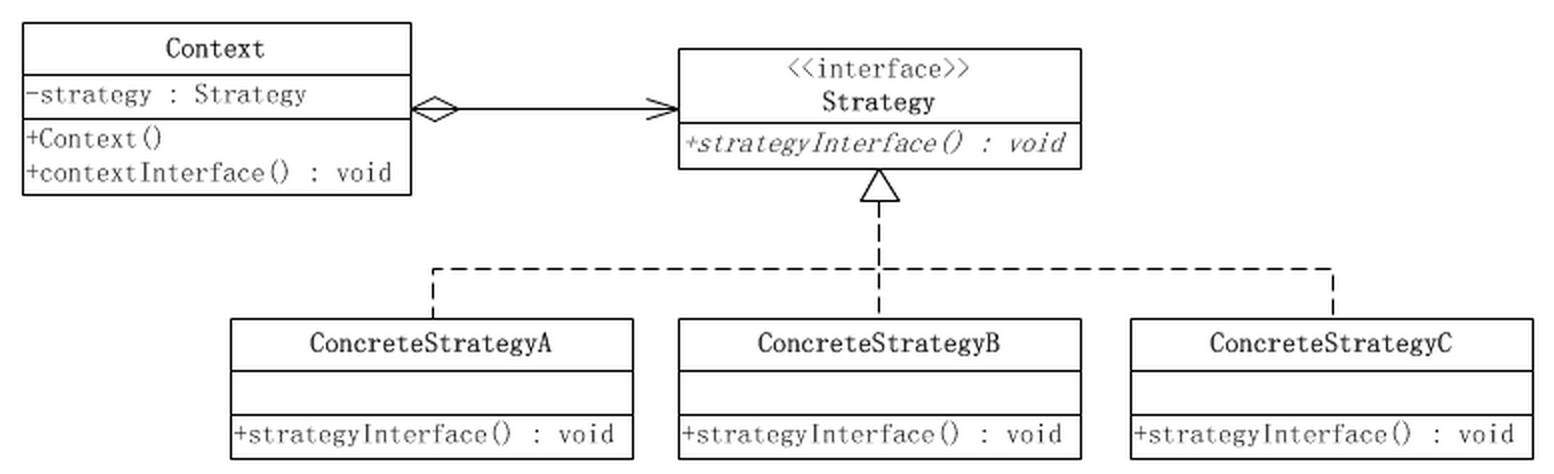
     （12）Strategy 策略模式

          这个模式涉及到三个角色：

**环境(Context)角色：**持有一个Strategy的引用。

**抽象策略(Strategy)角色：**这是一个抽象角色，通常由一个接口或抽象类实现。此角色给出所有的具体策略类所需的接口。

**具体策略(ConcreteStrategy)角色：**包装了相关的算法或行为。



           策略模式类图和状态模式很像，策略模式是算法的不同实现，状态模式是状态的不同，有不同的处理方式。

     （13）访问者模式，通过双重分派

          访问者模式涉及到的角色如下：

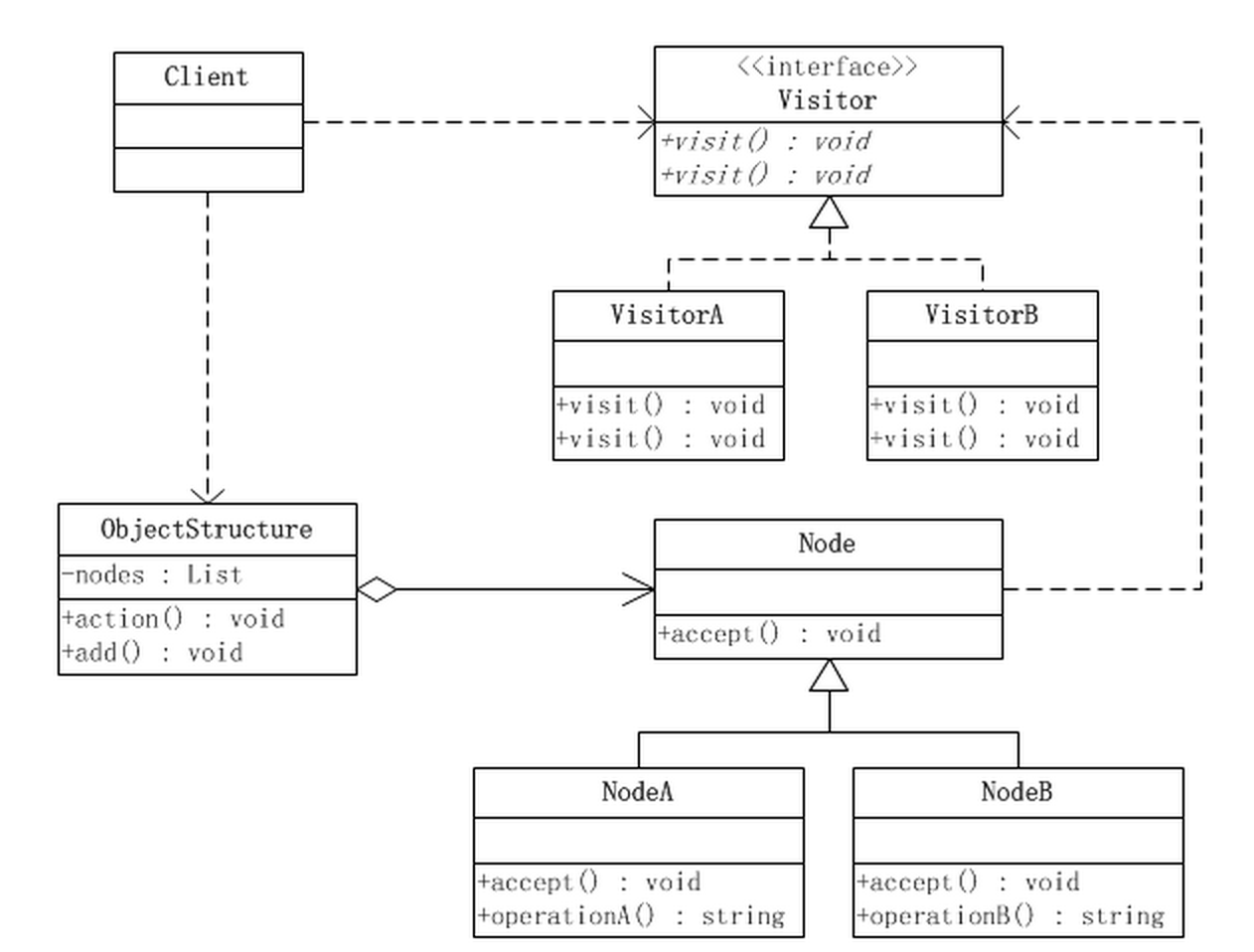
**抽象访问者(Visitor)角色：**声明了一个或者多个方法操作，形成所有的具体访问者角色必须实现的接口。

**具体访问者(ConcreteVisitor)角色：**实现抽象访问者所声明的接口，也就是抽象访问者所声明的各个访问操作。

**抽象节点(Node)角色：**声明一个接受操作，接受一个访问者对象作为一个参数。

**具体节点(ConcreteNode)角色：**实现了抽象节点所规定的接受操作。

**结构对象(ObjectStructure)角色：**有如下的责任，可以遍历结构中的所有元素；如果需要，提供一个高层次的接口让访问者对象可以访问每一个元素；如果需要，可以设计成一个复合对象或者一个聚集，如List或Set。



          概念：双重分派

          创建一个ObjectStructure对象，

          创建NodeA,NodeB,添加到ObjectStructure中

          创建Visitor对象

          遍历ObjectStructure中的Node对象，将Visitor传入

     (14)