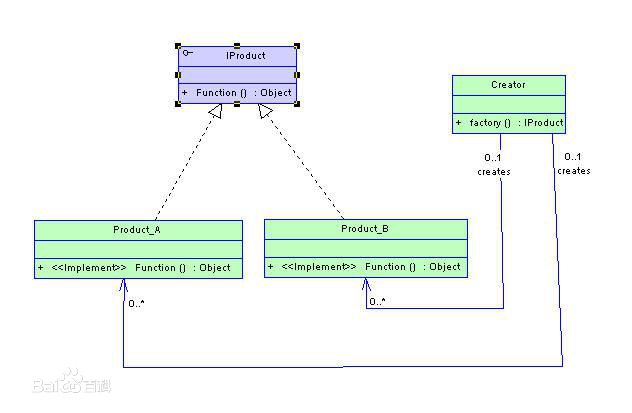
**简单工厂模式(Simple Factory):**

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。



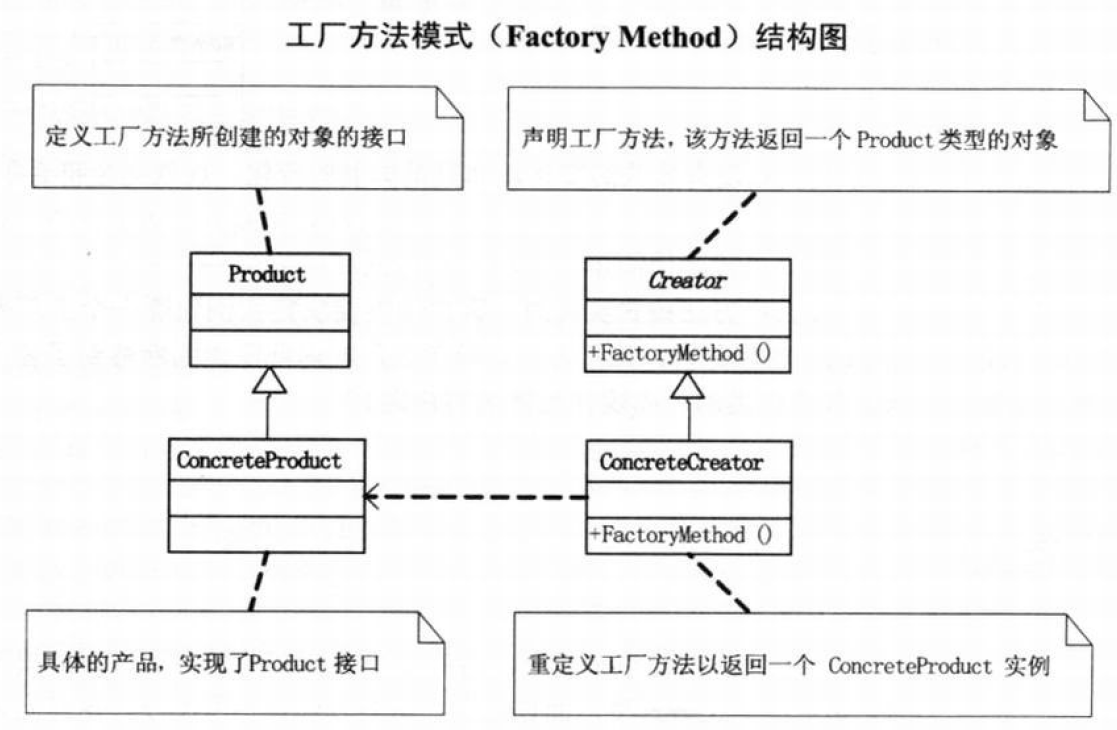
**工厂方法模式(Factory Method):**

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。

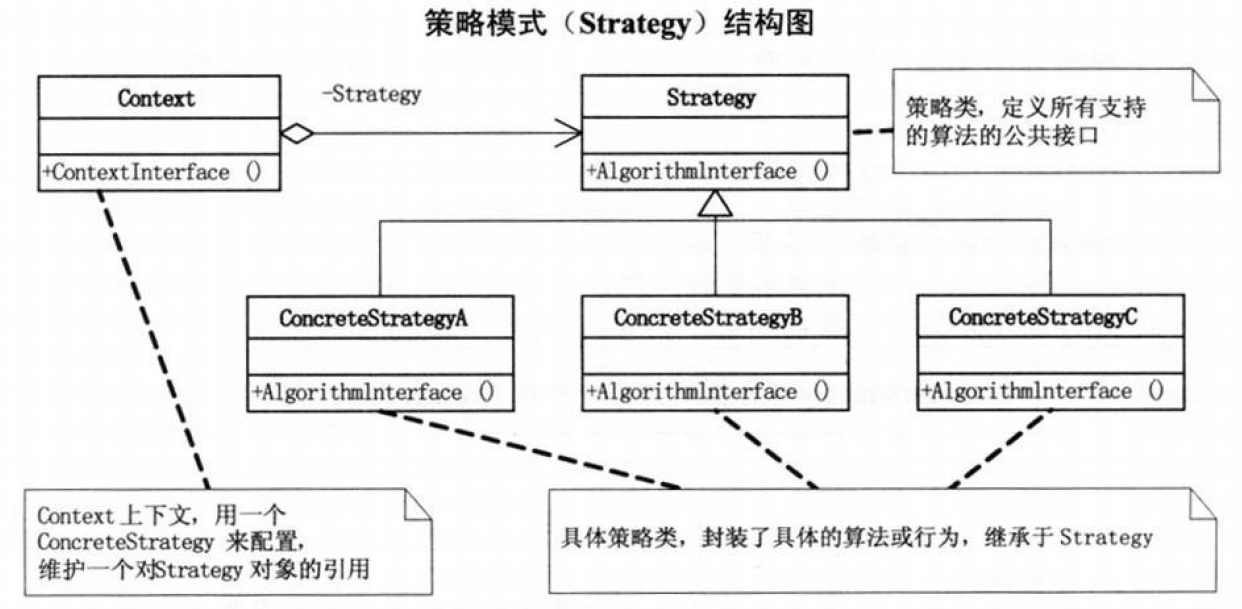


**策略模式(Strategy):**

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\*>）



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

**开放-封闭原则：**

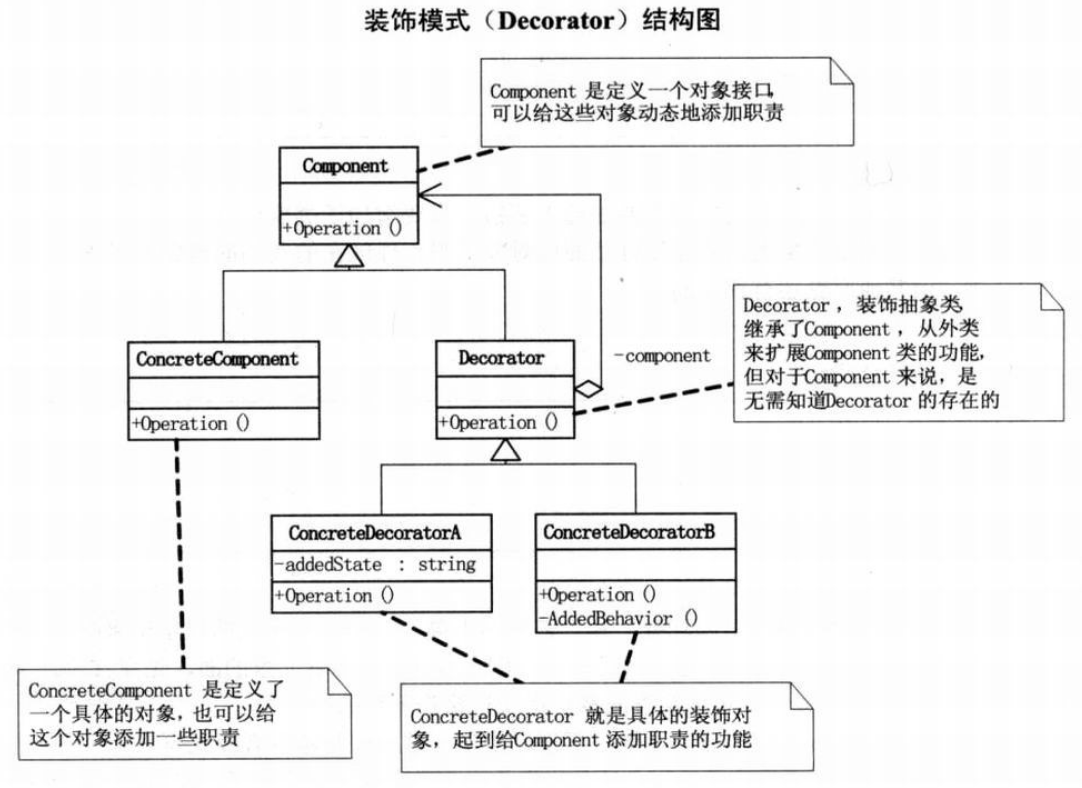
**依赖倒置原则：**

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。

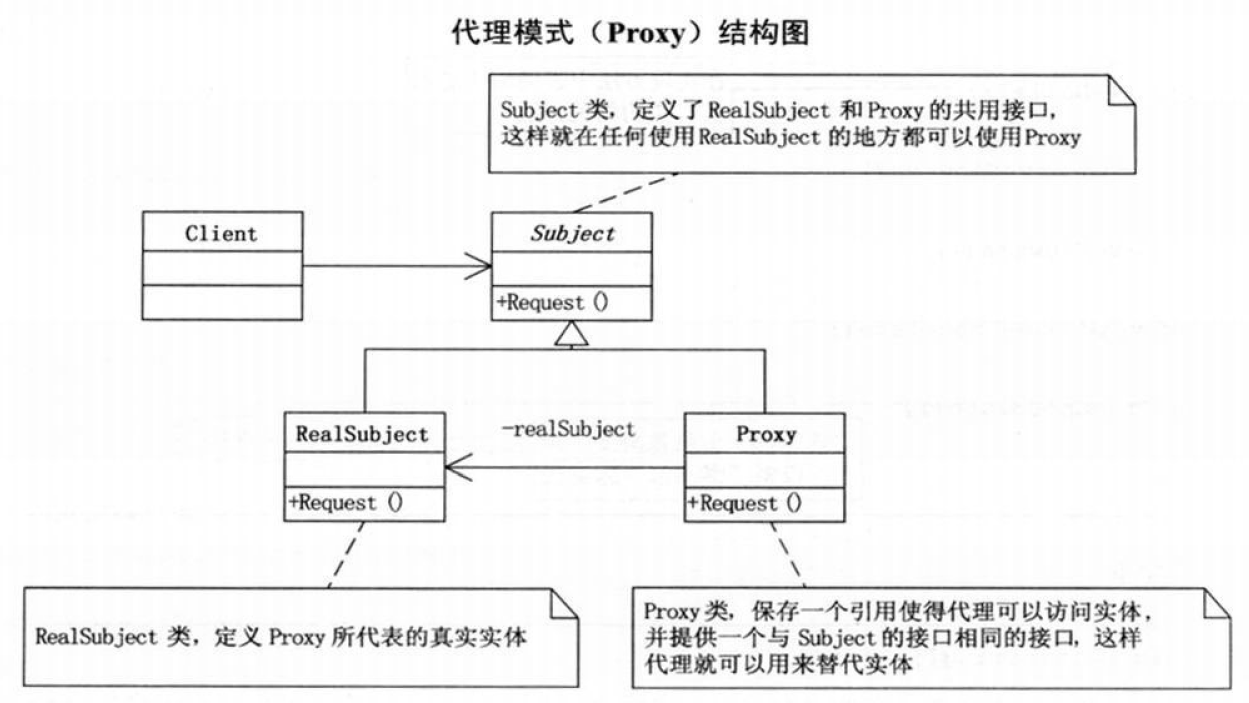
主张使用“抽象(Abstraction)”和“多态(Polymorphism)”将设计中的静态结构改为动态结构，维持设计的封闭性。“抽象”是语言提供的功能。“多态”由继承语义实现。

**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。

