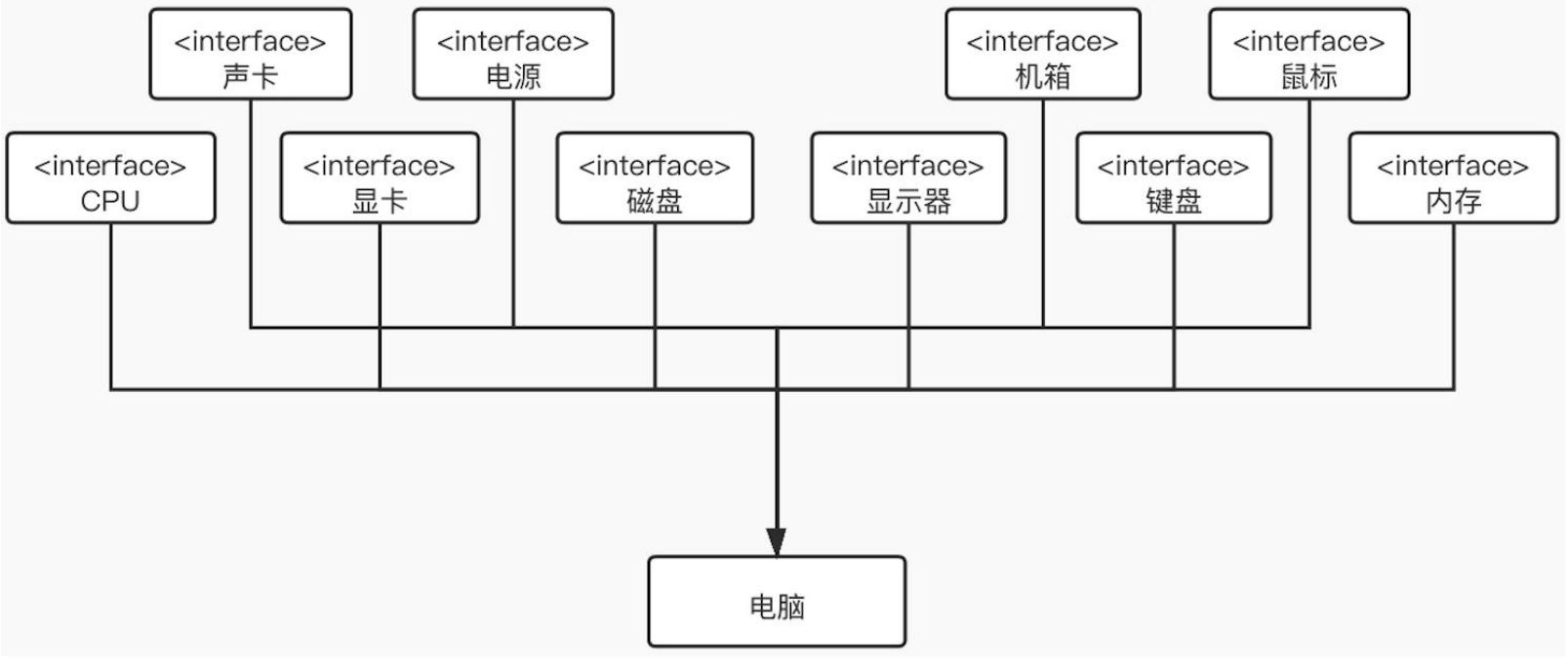
设计模式

六大设计原则

单一职责原则

SRP：Single Reposibility Principle
定义：一个类或者模块只负责完成一个职责。



里氏替换原则

LSP：Liskov Substitution Principle
定义：多用组合，少用继承。

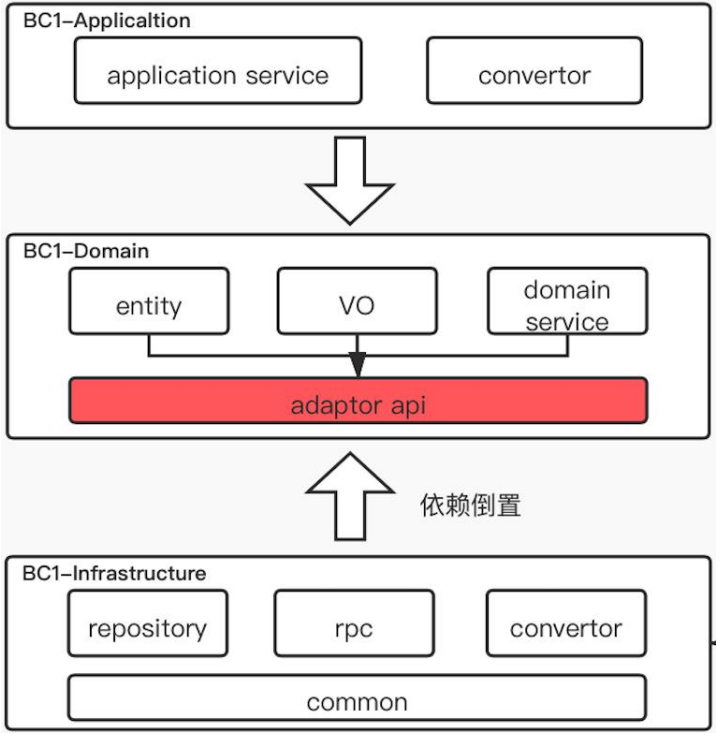
含义：
1> 里氏替换原则是针对继承而言的，如果继承是为了实现代码重用，也就是为了共享方法，那么共享的父类方法就应该保持不变，不能被子类重新定义。子类只能通过新添加方法来扩展功能，父类和子类都可以实例化，而子类继承的方法和父类是一样的，父类调用方法的地方，子类也可以调用同一个继承得来的，逻辑和父类一致的方法，这时用子类对象将父类对象替换掉时，当然逻辑一致，相安无事。

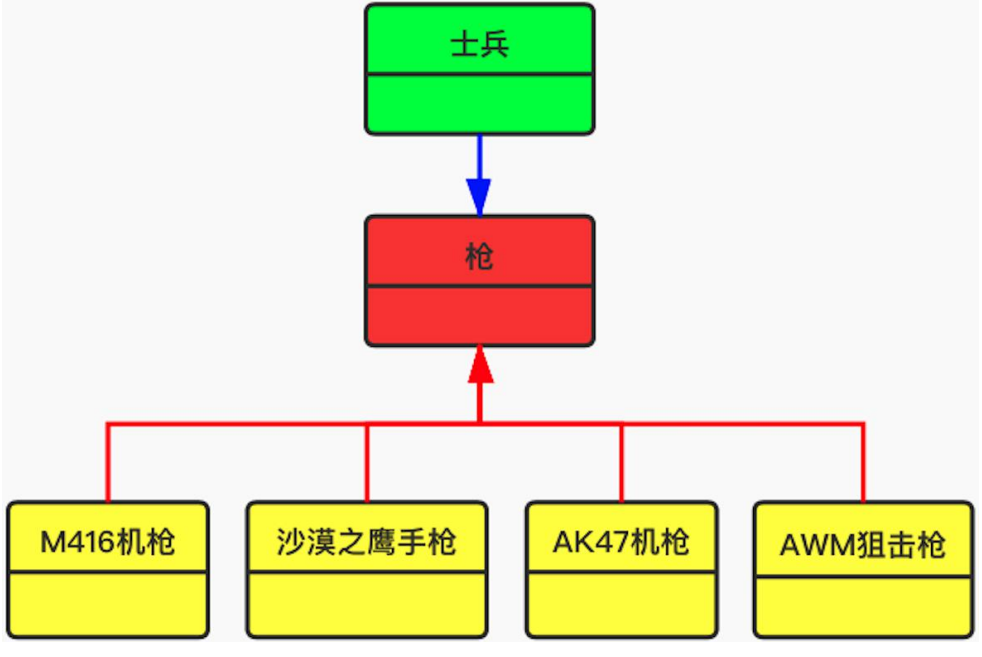
2> 如果继承的目的是为了多态，而多态的前提就是子类覆盖并重新定义父类的方法，为了符合LSP，我们应该将父类定义为抽象类，并定义抽象方法，让子类重新定义这些方法，当父类是抽象类时，父类就是不能实例化，所以也不存在可实例化的父类对象在程序里。也就不存在子类替换父类实例（根本不存在父类实例了）时逻辑不一致的可能。

依赖倒置原则

DIP：Dependence Inversion Principle

定义：下层模块引入上层模块的依赖，改变原有自上而下的依赖方向。





接口隔离原则

定义：建立单一接口，不要建立臃肿庞大的接口。接口尽量细化，同时接口中的方法尽量少。

含义：

1> 接口要尽量小

不要违反单一职责原则。

要适度的小。要适度。

2> 接口要高内聚

提高接口、类、模块的处理能力，减少对外的交互。

3> 定制服务

通过对高质量接口的组装，实现服务的定制化。

迪米特法则/最少知识原则

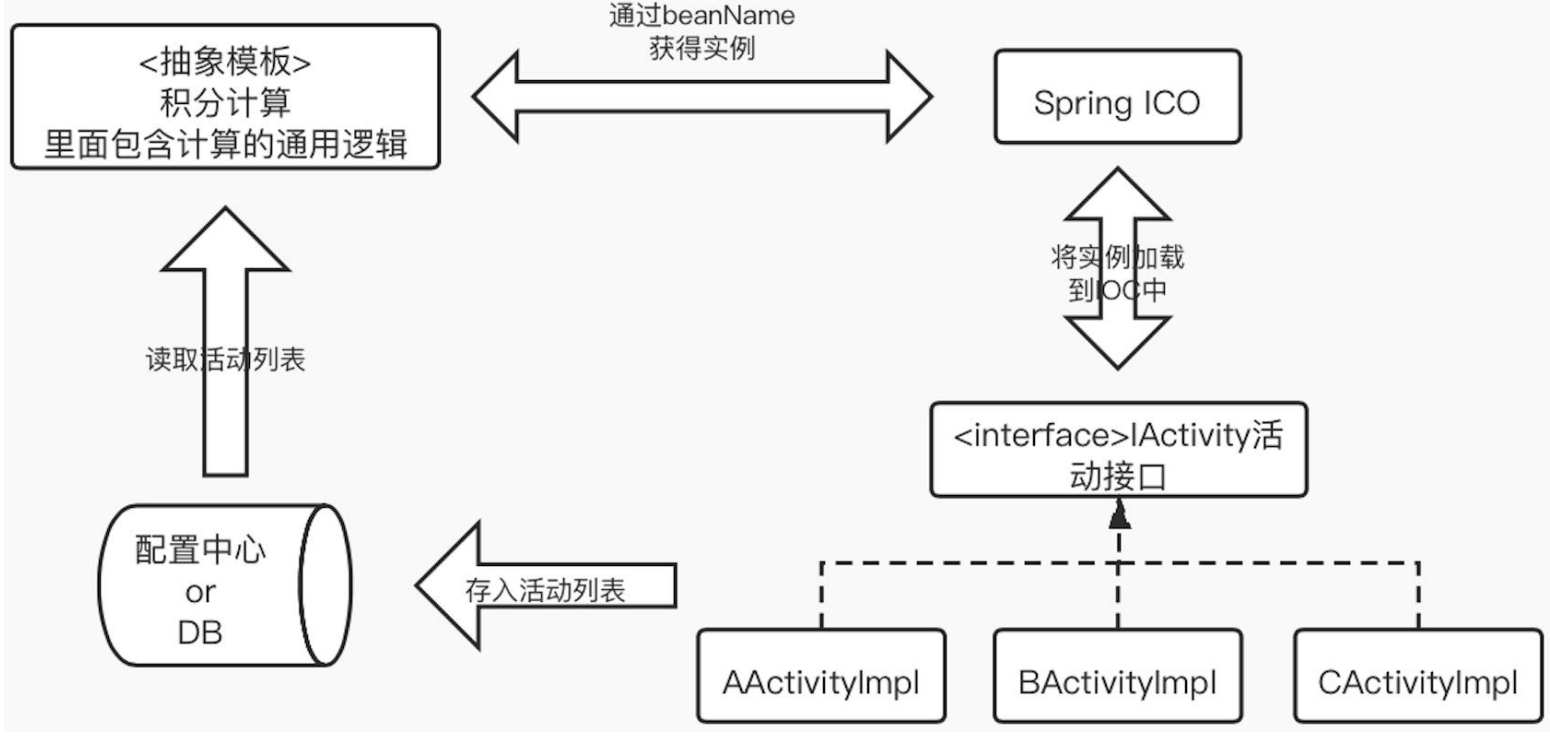
LoD：Law of Demeter
定义：只和你的密友谈话。

具体解释：一个类应该对自己需要耦合或调用的类知道得最少，你（被耦合或调用的类）的内部是如何复杂，那是你的事儿，和我没关系，我就知道你提供的这么多public方法，我就调用这么多，其他的我一概不关系。

拓展知识：DDD

开闭原则

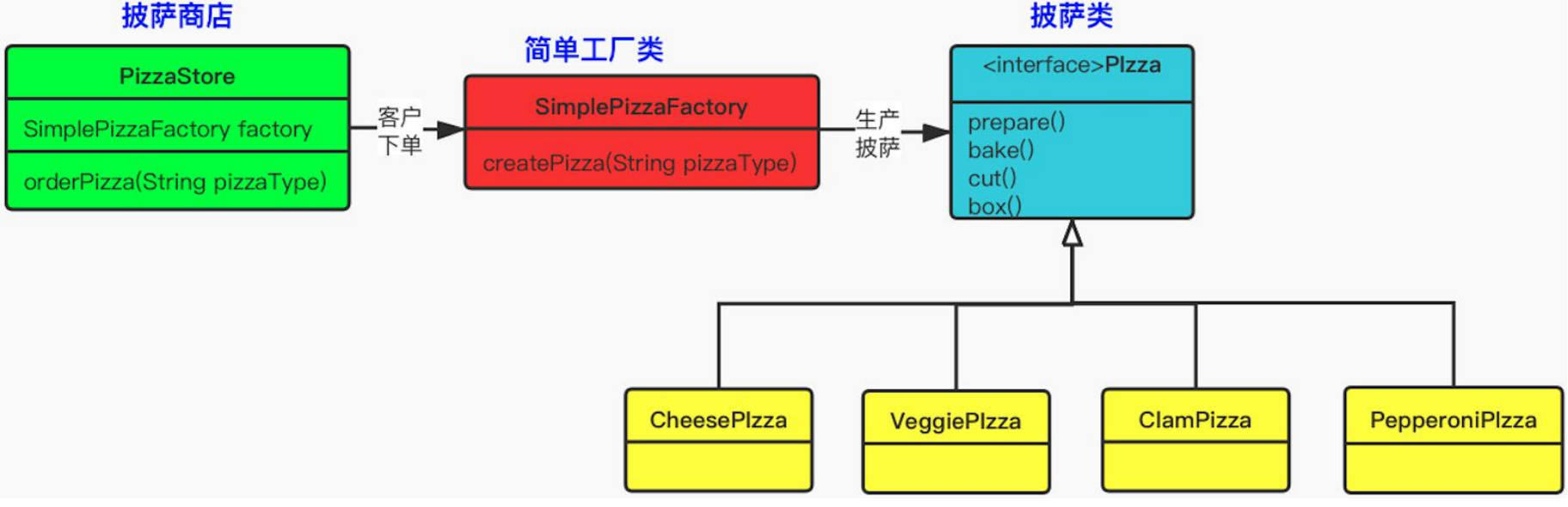
定义：类、方法、模块应该对扩展开放，对修改关闭。
含义：通俗讲：添加一个功能应该是在已有的代码基础上进行扩展，而不是修改已有的代码。



创建型设计模式

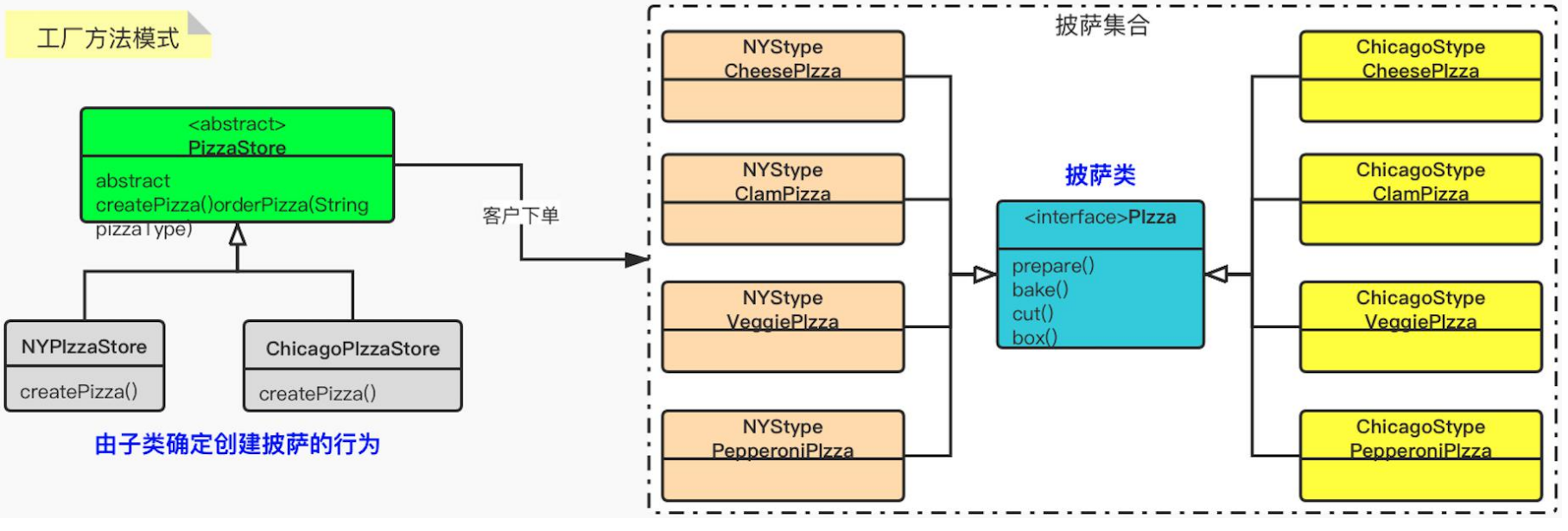
简单工厂

定义：简单工厂其实不是一个设计模式，反而比较像是一种编程习惯。



工厂方法模式

定义：定义了一个创建对象的接口（类或接口中的方法），但由子类决定要实例化的类是哪一个。工厂方法把实例化推迟到子类。

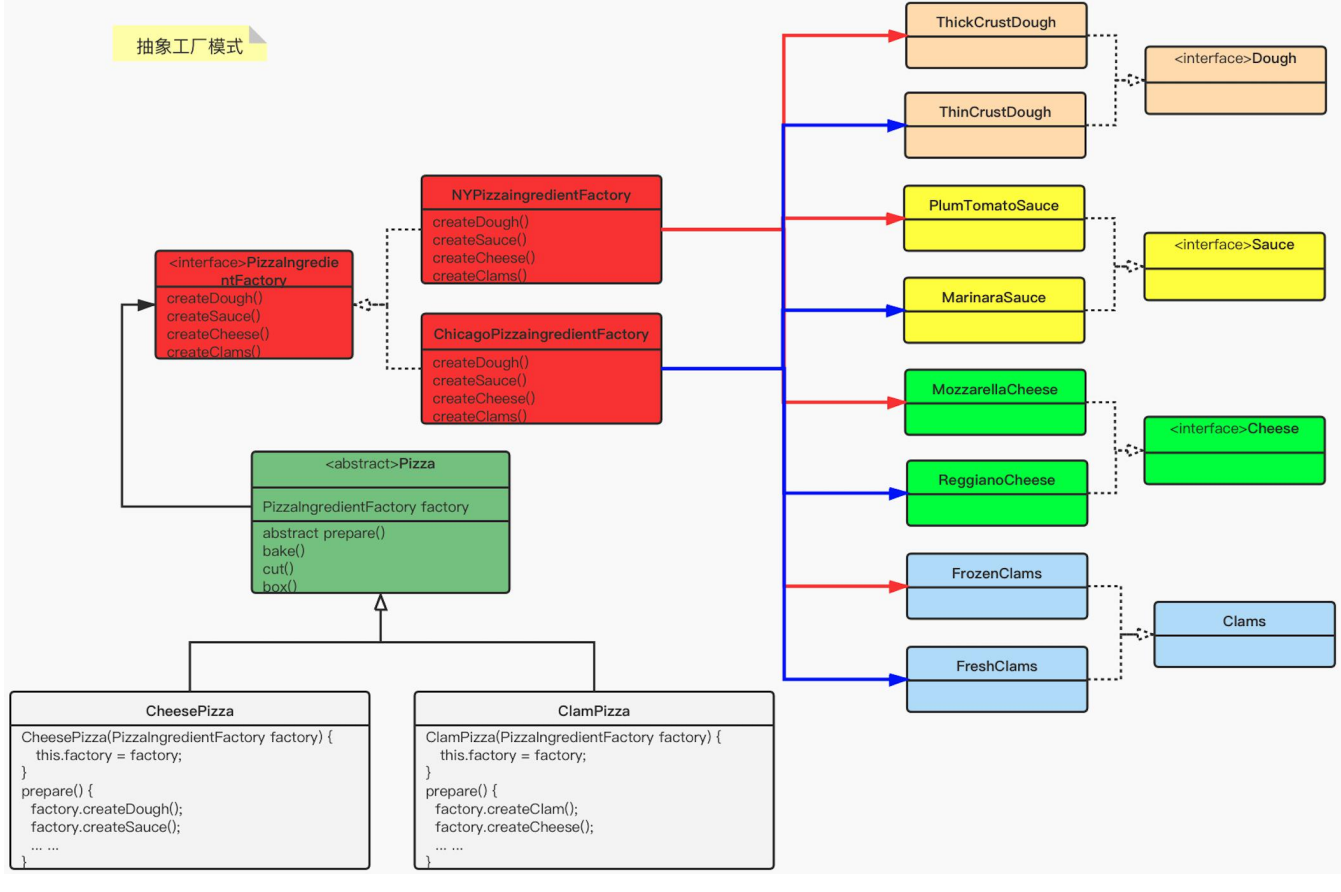


抽象工厂模式

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。

工厂方法：通过抽象方法提供对象生成入口。

抽象工厂：通过接口，来创建一组产品。

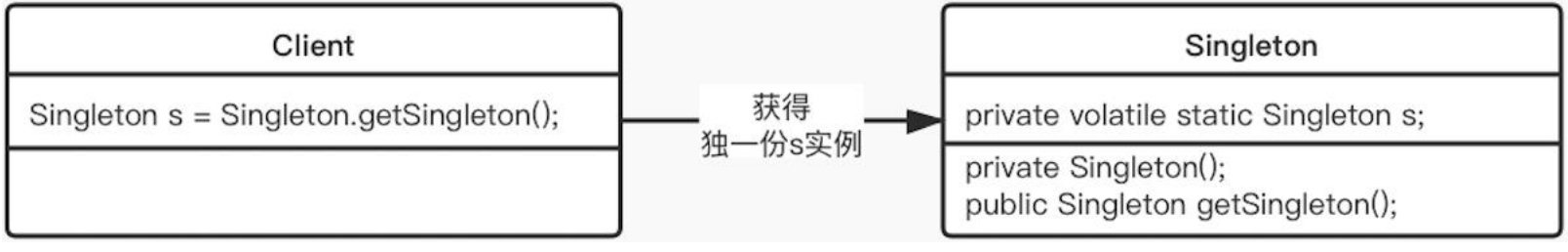


单例模式

定义：确保一个类只有一个实例，并提供一个全局访问点。

五种实现方式：

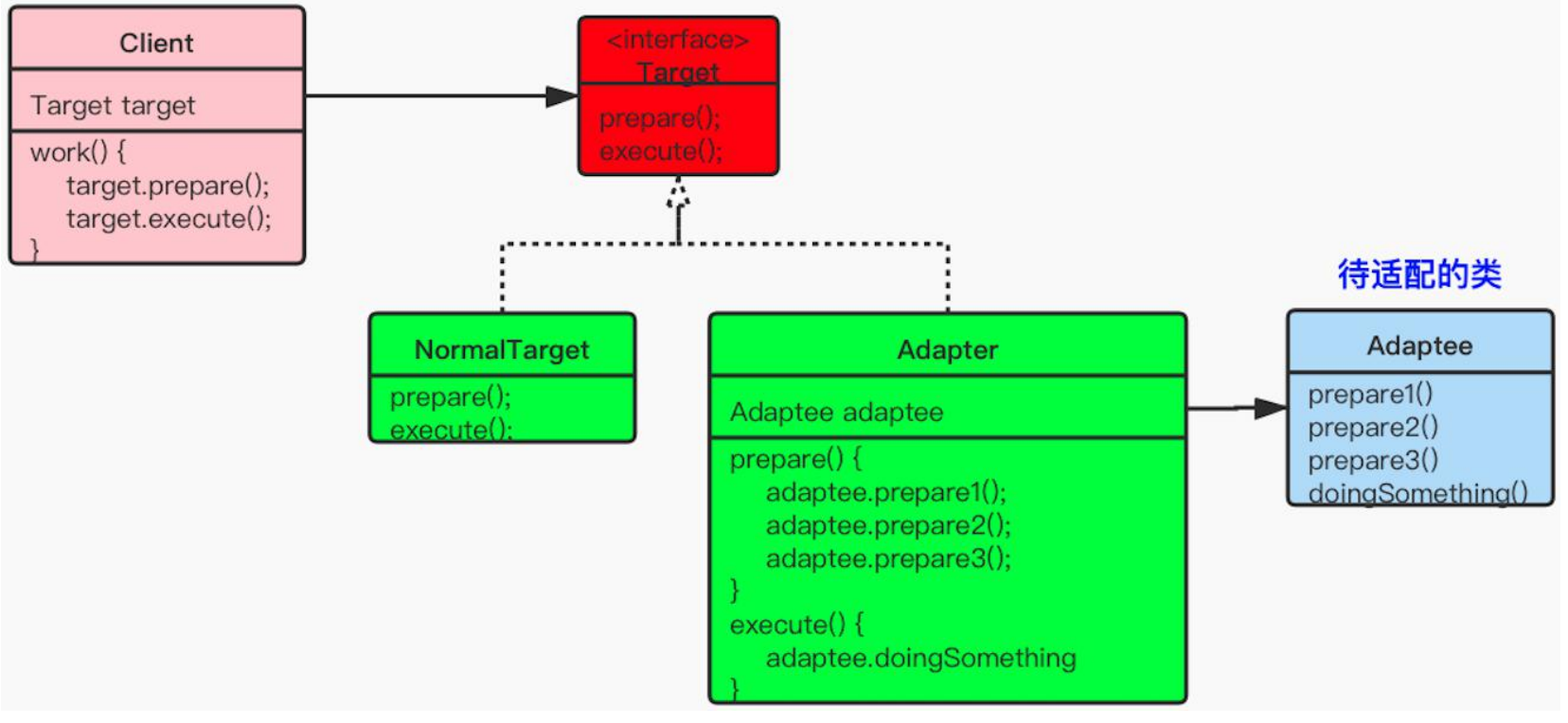
- 饿汉式
- 懒汉式
- 双重校验
- 静态内部类
- 枚举类



结构型设计模式

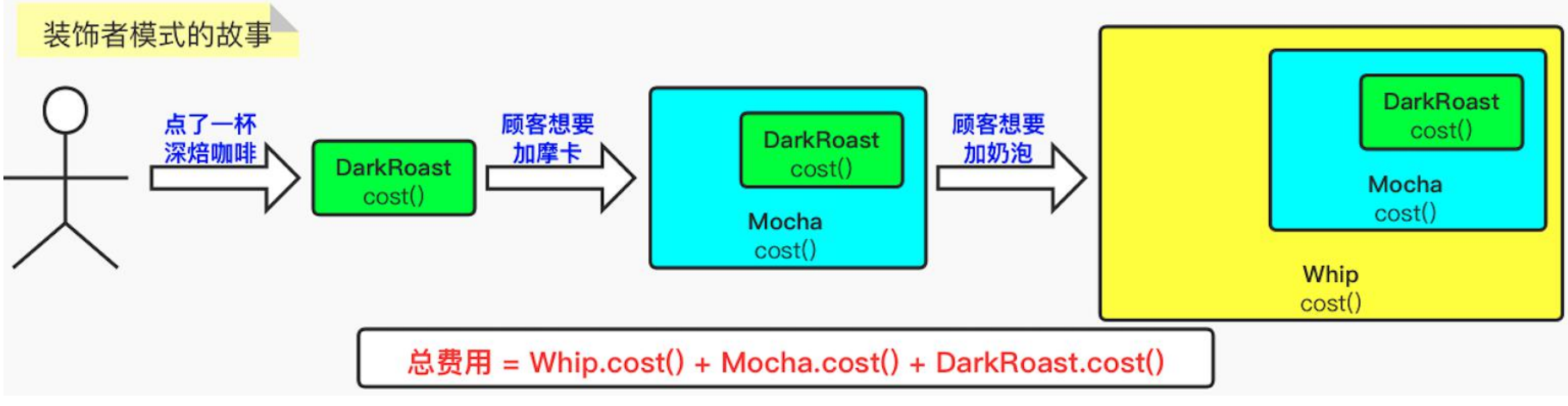
适配器模式

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。

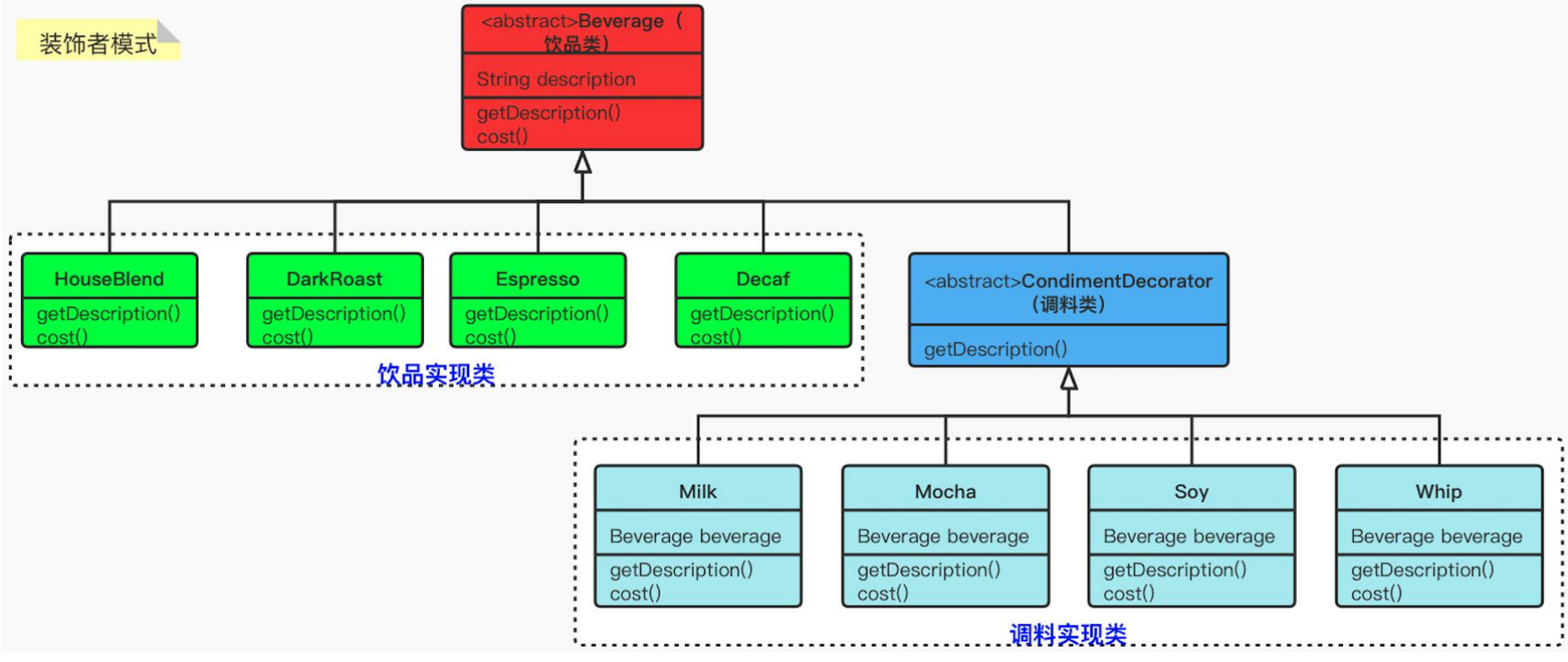


装饰者模式

定义：动态地将责任附加到对象上。若要扩展功能，装饰者提供了比继承更有弹性的替代方案。

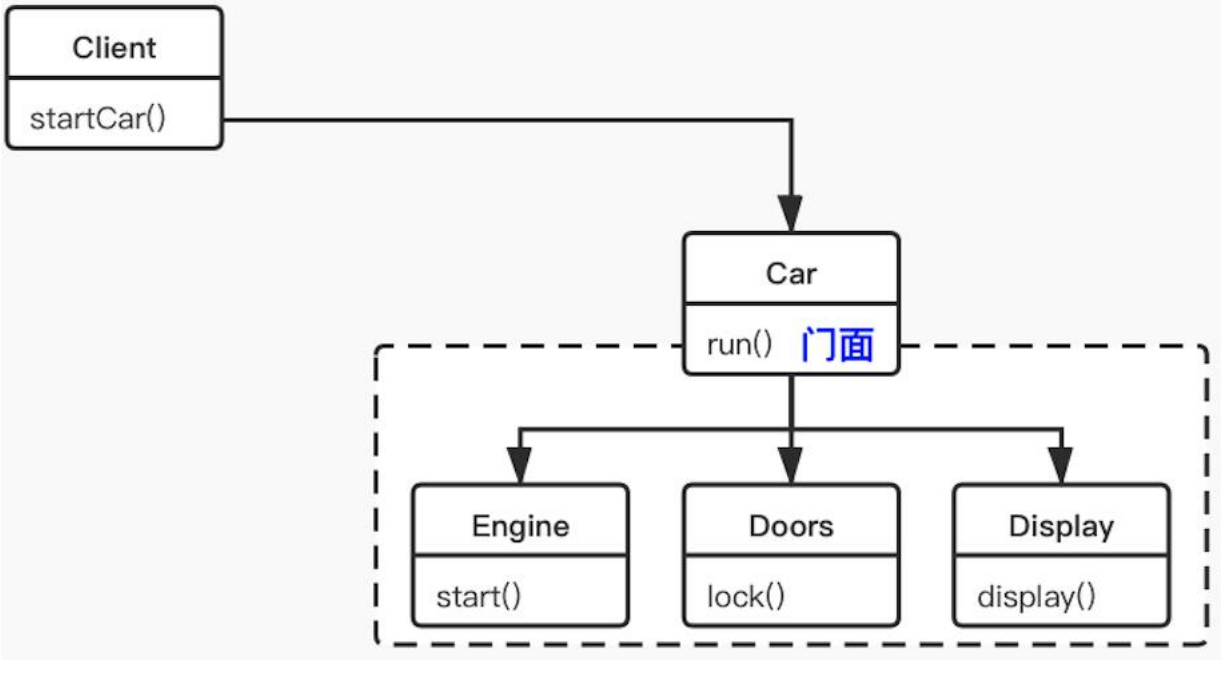


装饰者模式


装饰者模式
<abstract>Beverage（饮品类）
String description
getDescription()
cost()
HouseBlend
getDescription()
cost()
DarkRoast
getDescription()
cost()
Espresso
getDescription()
cost()
Decaf
getDescription()
cost()
饮品实现类
<abstract>CondimentDecorator（调料类）
getDescription()
Milk
Beverage beverage
getDescription()
cost()
Mocha
Beverage beverage
getDescription()
cost()
Soy
Beverage beverage
getDescription()
cost()
Whip
Beverage beverage
getDescription()
cost()
调料实现类

门面模式

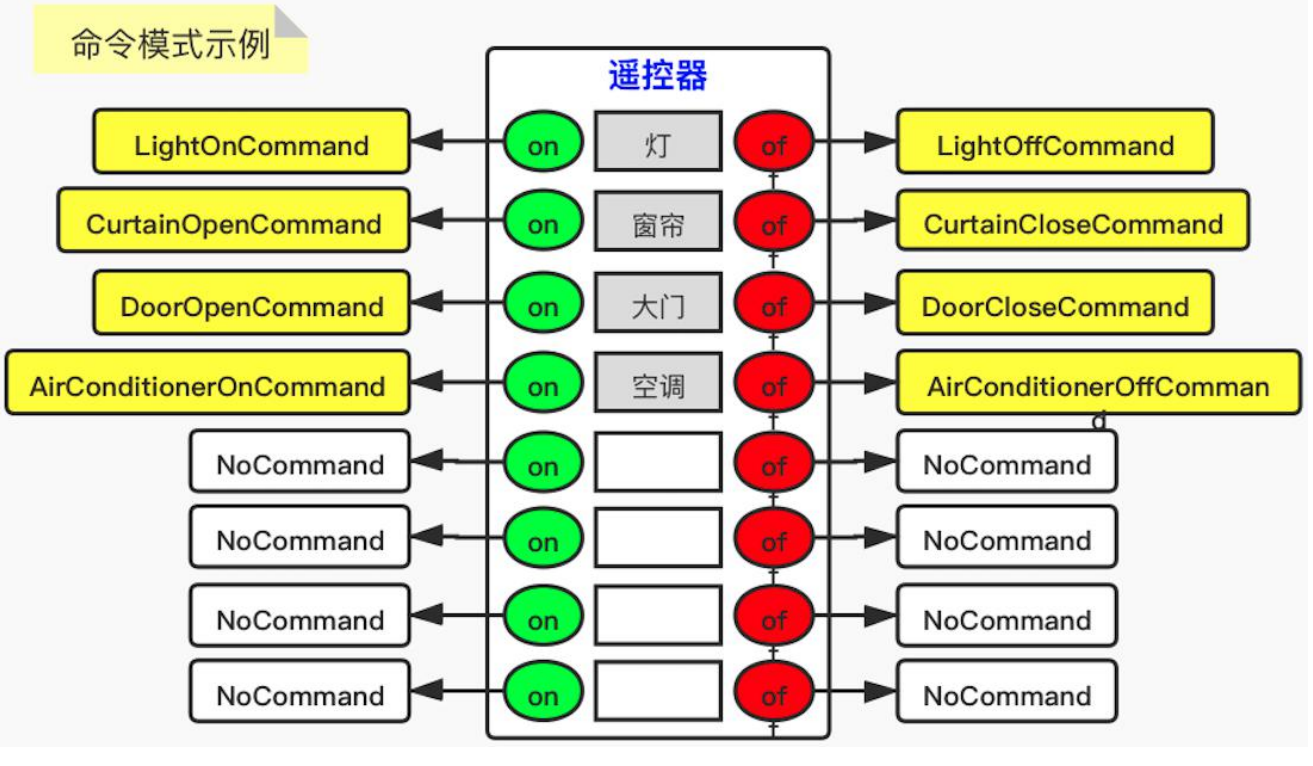
定义：提供了一个统一的接口，用来访问子系统中的一群接口。外观定义了一个高层接口，让子系统更容易使用。



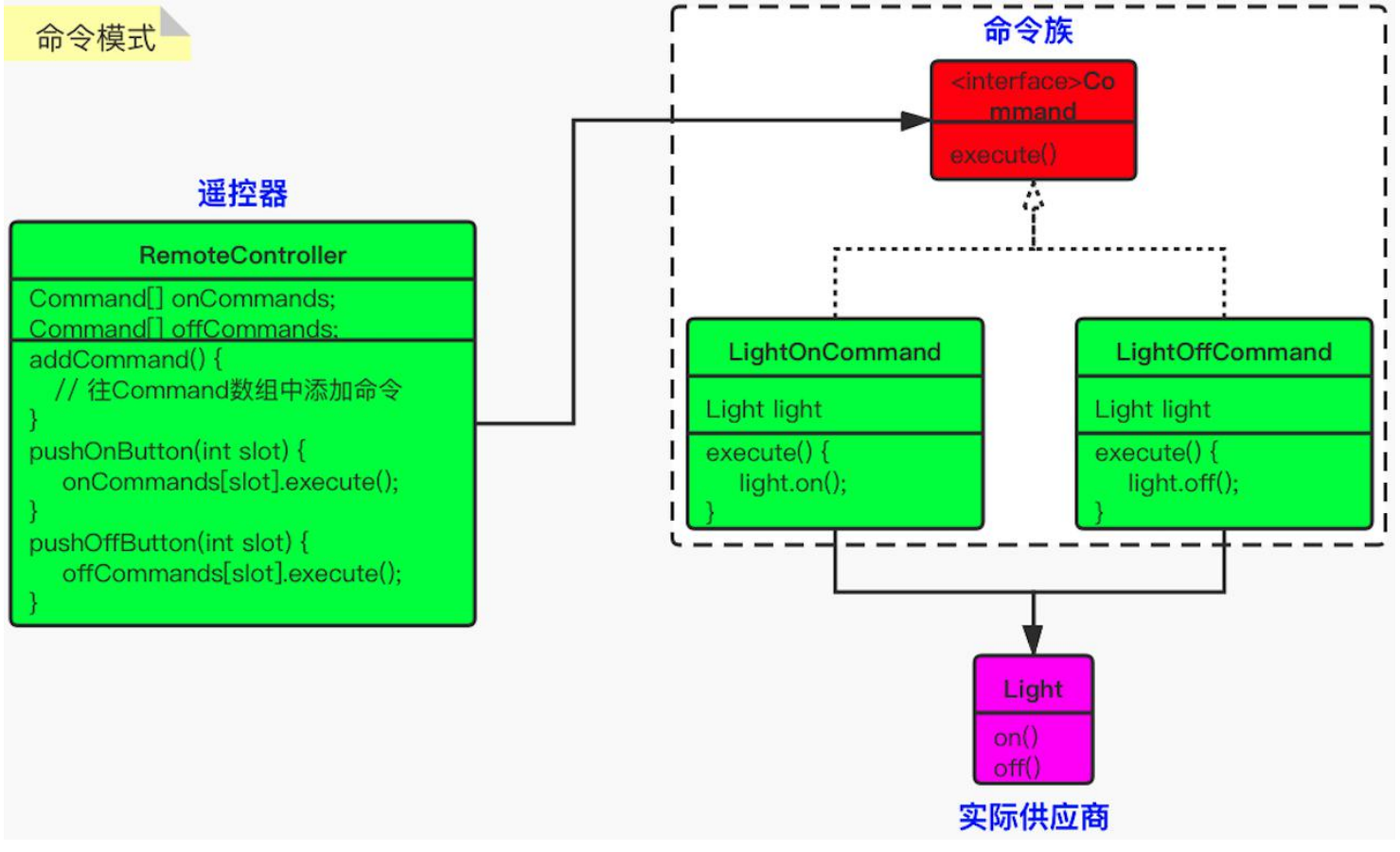
行为型设计模式

命令模式

定义：将”请求“封装成对象，以便使用不同的请求、队列或日志来参数化其他对象。命令模式也支持可撤销的操作。

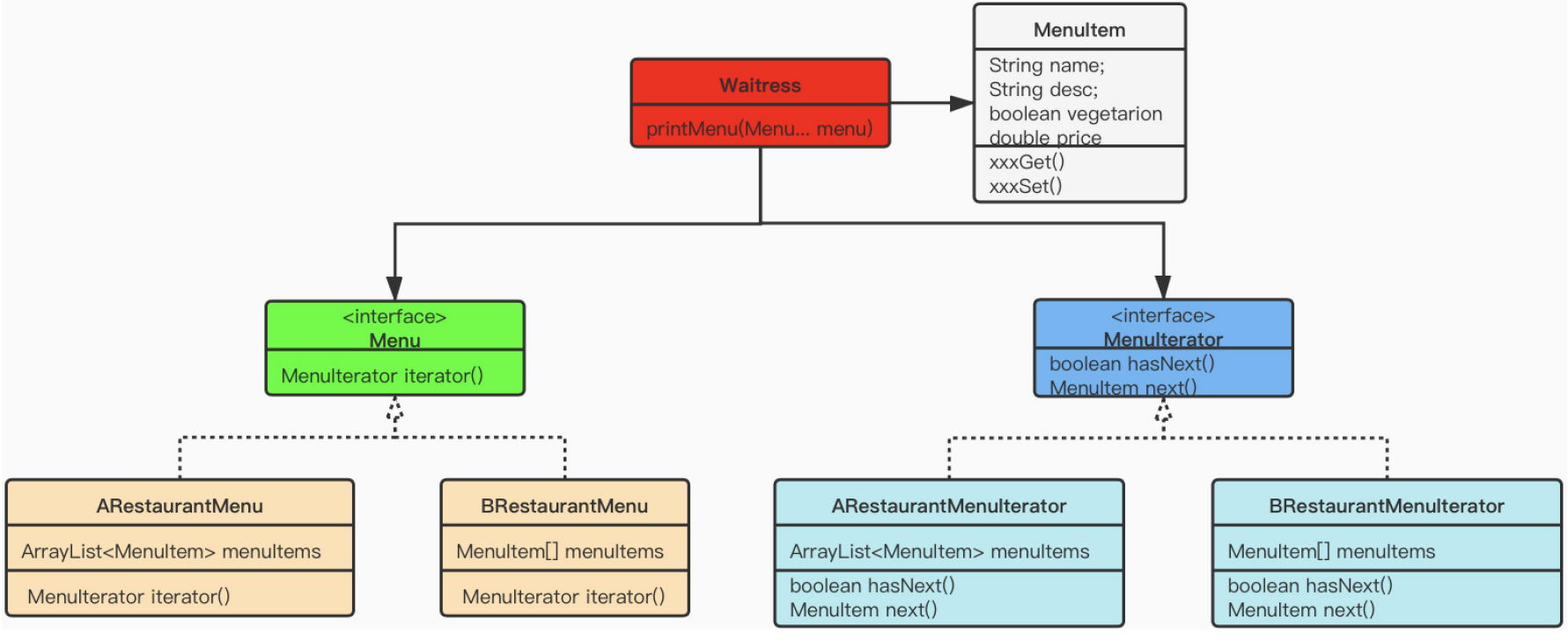


命令模式


命令模式
命令族
<interface>Command
execute()
遥控器
RemoteController
Command[] onCommands;
Command[] offCommands;
addCommand() {
// 往Command数组中添加命令
}
pushOnButton(int slot) {
onCommands[slot].execute();
}
pushOffButton(int slot) {
offCommands[slot].execute();
}
LightOnCommand
Light light
execute() {
light.on();
}
LightOffCommand
Light light
execute() {
light.off();
}
Light
on()
off()
实际供应商

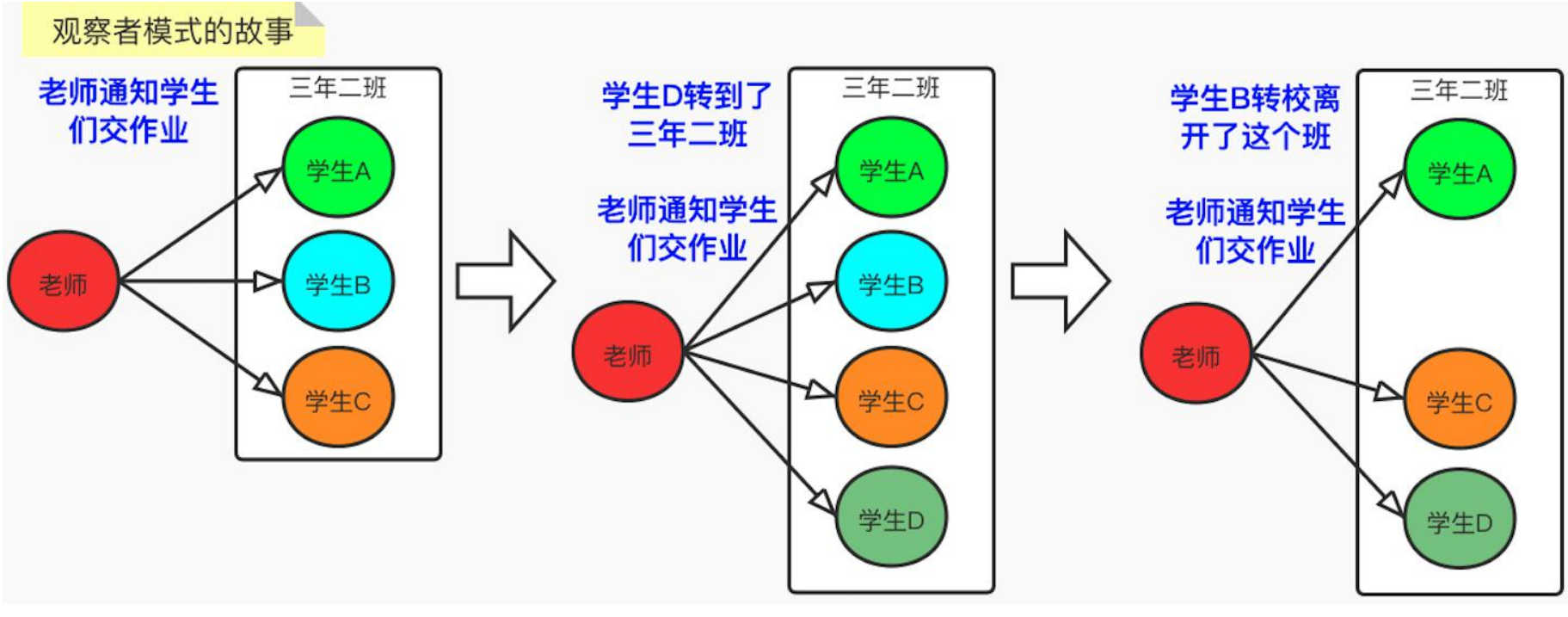
迭代器模式

定义： 提供一个方法顺序访问一个聚合对象中的各个元素，而又不暴露其内部的表示。



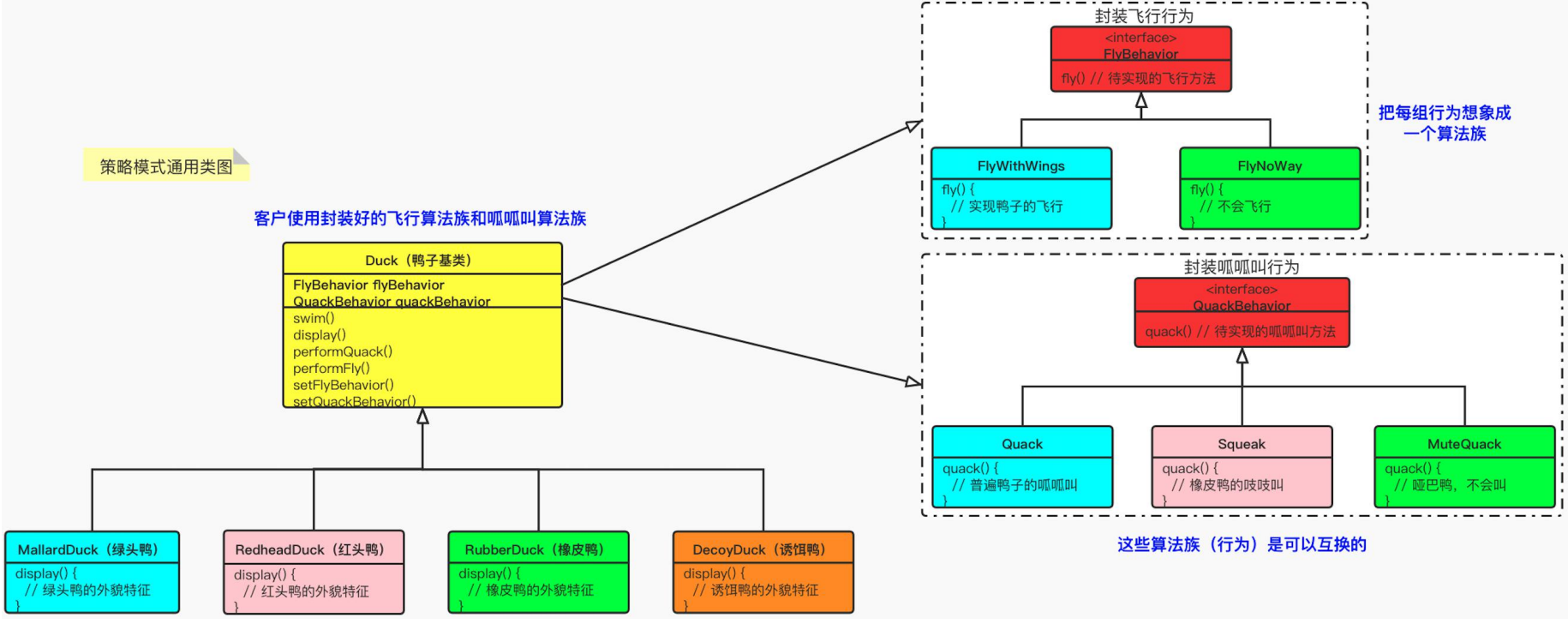
观察者模式

定义：定义了对象之间的一对多依赖，这样一来，当一个对象改变状态时，它的所有依赖者都会收到通知并自动更新。



策略模式

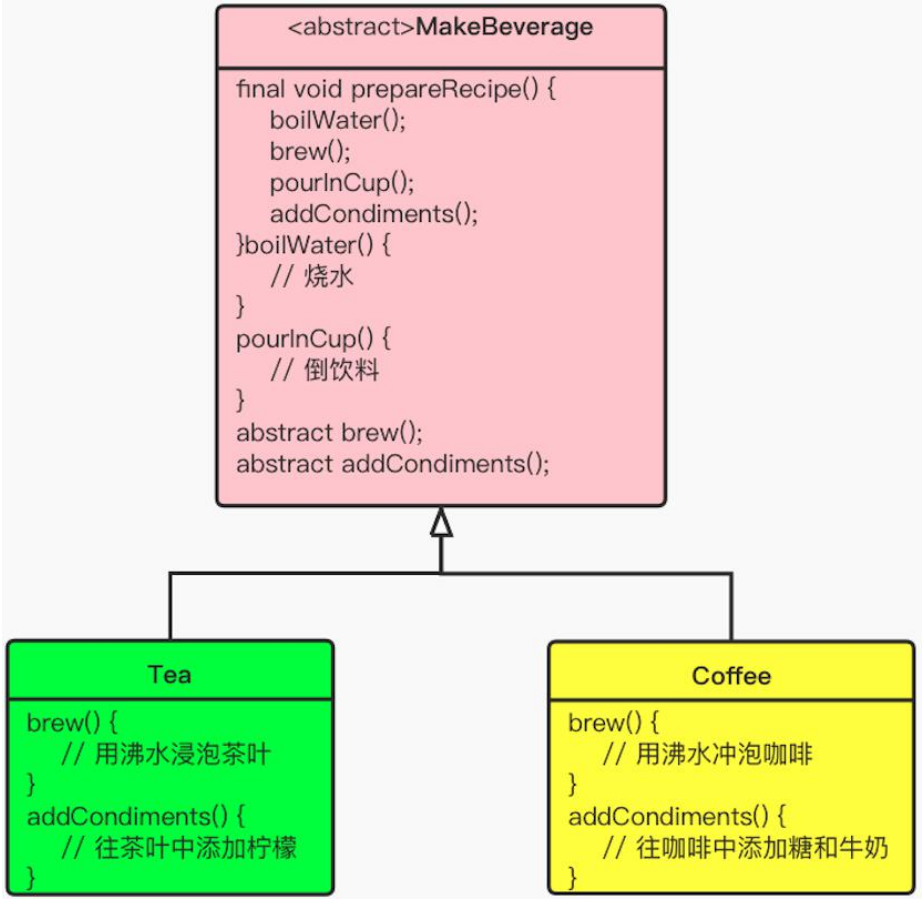
定义：定义了算法族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法的客户。



模板方法模式

定义：
在一个方法中定义一个算法的骨架，而将一些步骤延迟到子类中。

模板方法使得子类可以在不改变算法结构的情况下，重新定义算法中的某些步骤。



结束