设计模式

学习资料

学习手册 : https://blog.csdn.net/lovelion/article/details/17517213

《Head First设计模式》： "

[]: D:\00persional\Head First 设计模式（中文版）.pdf

"

《软件架构与设计模式》

《图解设计模式》

iterator 模式

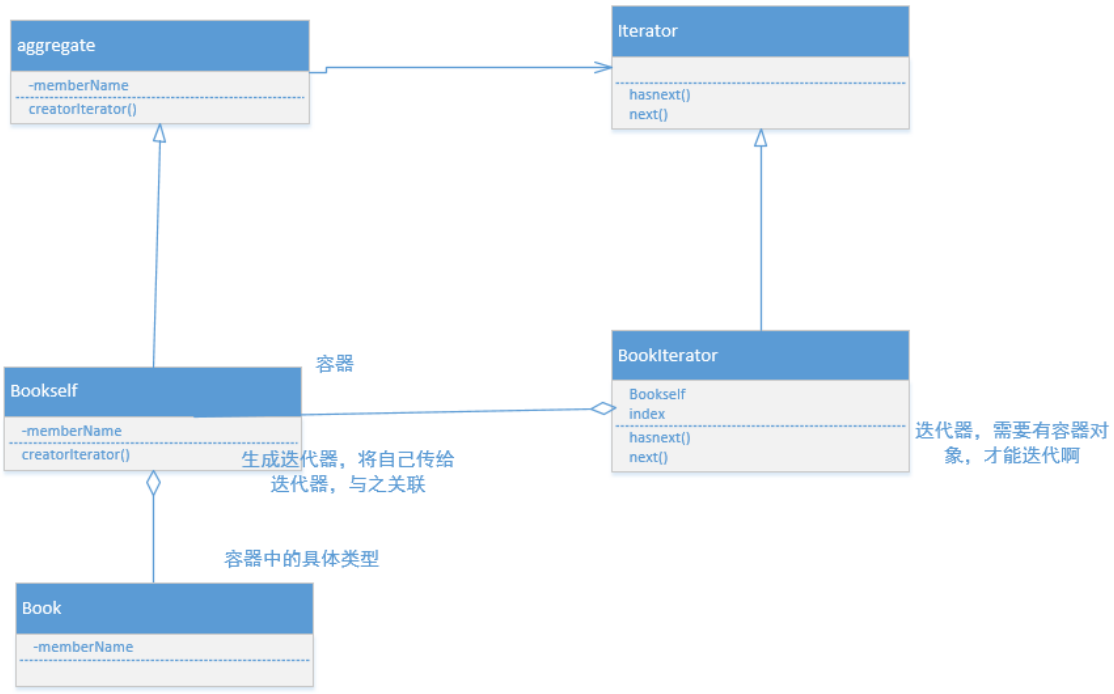
参考：https://blog.csdn.net/u012611878/article/details/78010435

主要是遍历容器使用， 抽象包括：集合抽象，迭代器抽象类；

容器中包含创造迭代器的方法，方便客户使用；

迭代器子类中与容器为聚类关系，因为迭代器要在容器中迭代； 关系比较确定，需要使用模板做泛型。

类图如下；

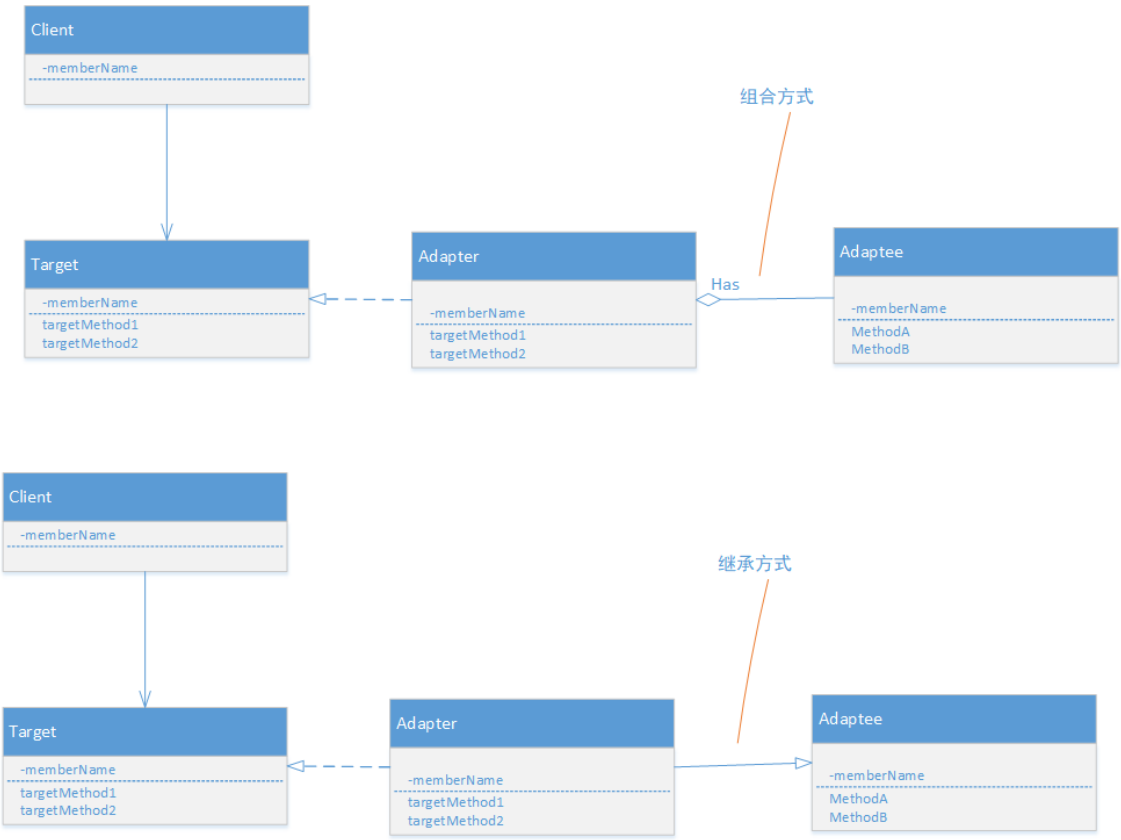


适配器模式

填补现有的类与所需的类之间的差异的模式，又称为wrapper模式，有两种实现方式

- 类适配器模式：使用继承的适配器；
- 对象适配器模式：使用委托（组合）的适配器；

  类图如下：Adapter为要实现的类；



相关的涉及模式：

Bridges模式：adapter用于连接接口（API）不同的类，而Bridege用于连接类的功能层次结构与实现结构

Decorator模式：decorator模式是在不改变接口的前提下增加功能。

Template method

核心代码如下，基类定义了模板方法，子类实现里面的方法；子类具有实现父类定义的方法的责任，实现责任迁移；

相关模式：

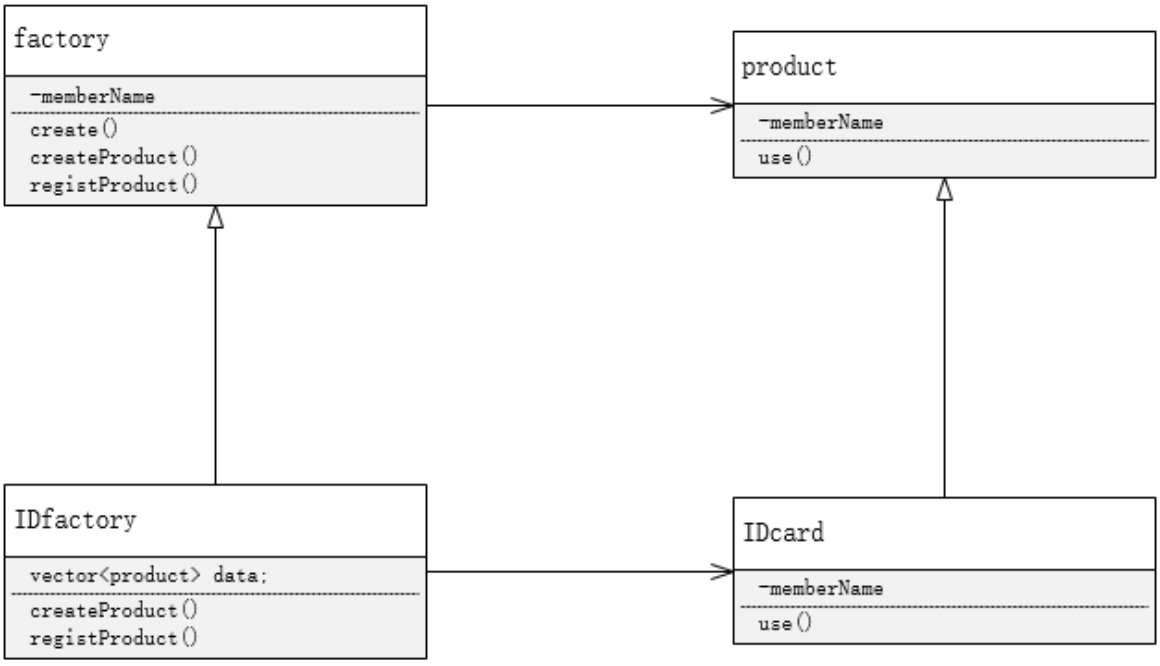
factory method：模板模式的一个实例；

strategy 模式：模板模式改变部分程序行为，策略模式用于替换整个算法；

```
public abstract class AbstractDisplay {  // 抽象类 AbstractDisplay
    public abstract void open();         // 交给子类去实现的抽象方法 (1)open
    public abstract void print();        // 交给子类去实现的抽象方法 (2) print
    public abstract void close();        // 交给子类去实现的抽象方法 (3) close
    public final void display() {        // 本抽象类中实现的 display 方法
        open();                          // 首先打开……
        for (int i = 0; i < 5; i++) {    // 循环调用 5 次 print……
            print();
        }
        close();          // ……最后关闭。这就是 display 方法所实现的功能
    }
}
```

Factory 模式

使用了模板模式，抽象工厂与产品形成一个小的framework，不依赖具体的其他的类。



SIngleton模式

核心：只能创造一个对象；

使用private构造函数，在类的内部函数里面new 一个对象；

多线程模式需要加锁；

notice:

查看汇编，private在汇编的体现；

【示例】从键盘输入一个数组，将数组写入文件再读取出来。

```c
#include<stdio.h>
#define N 5
int main(){
    //从键盘输入的数据放入a，从文件读取的数据放入b
    int a[N], b[N];
    int i, size = sizeof(int);
    FILE *fp;

    if( (fp=fopen("D:\\demo.txt", "rb+")) == NULL ){  //以二进制方式打开
        puts("Fail to open file!");
        exit(0);
    }

    //从键盘输入数据 并保存到数组a
    for(i=0; i<N; i++){
        scanf("%d", &a[i]);
    }
    //将数组a的内容写入到文件
    fwrite(a, size, N, fp);
    //将文件中的位置指针重新定位到文件开头
    rewind(fp);
    //从文件读取内容并保存到数组b
    fread(b, size, N, fp);
    //在屏幕上显示数组b的内容
    for(i=0; i<N; i++){
        printf("%d ", b[i]);
    }
    printf("\n");

    fclose(fp);
    return 0;
}
```

C文件读写注意事项

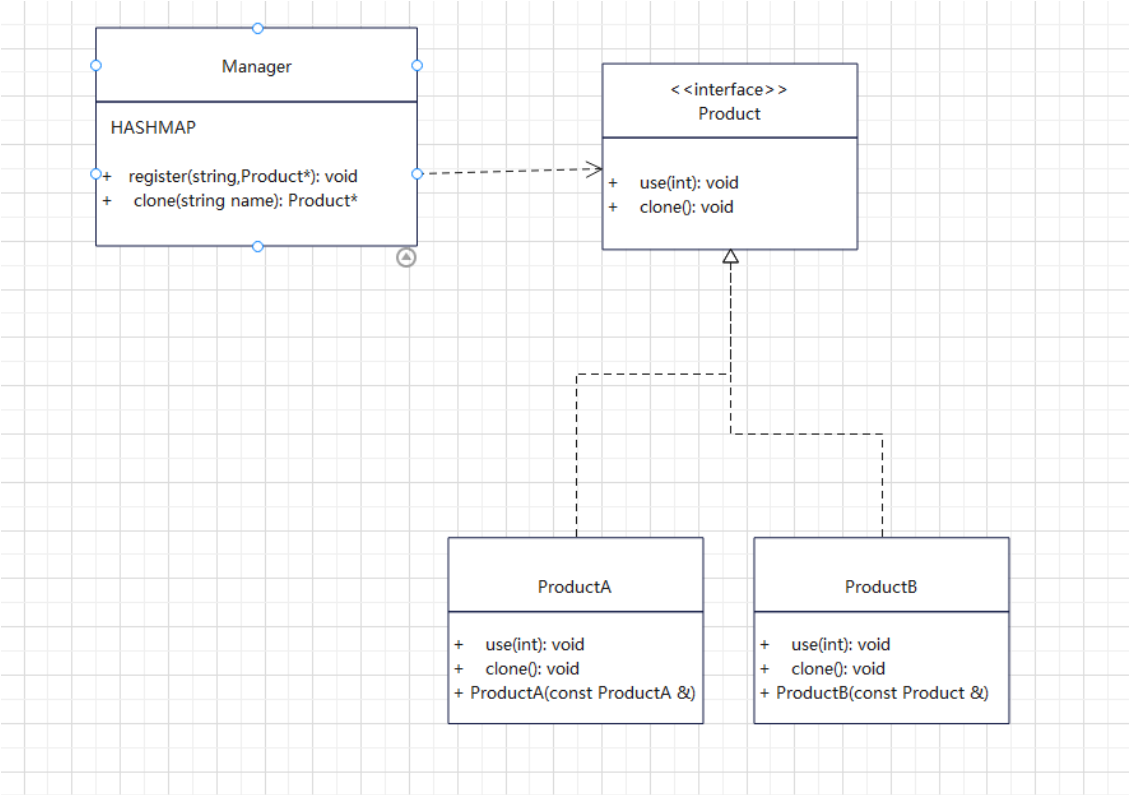
fopen, fwrite ,rb+,wb，读写方式要一致；

原型模式

原型模式通过clone已有的对象，来创建新的对象，没有通过类的名称；

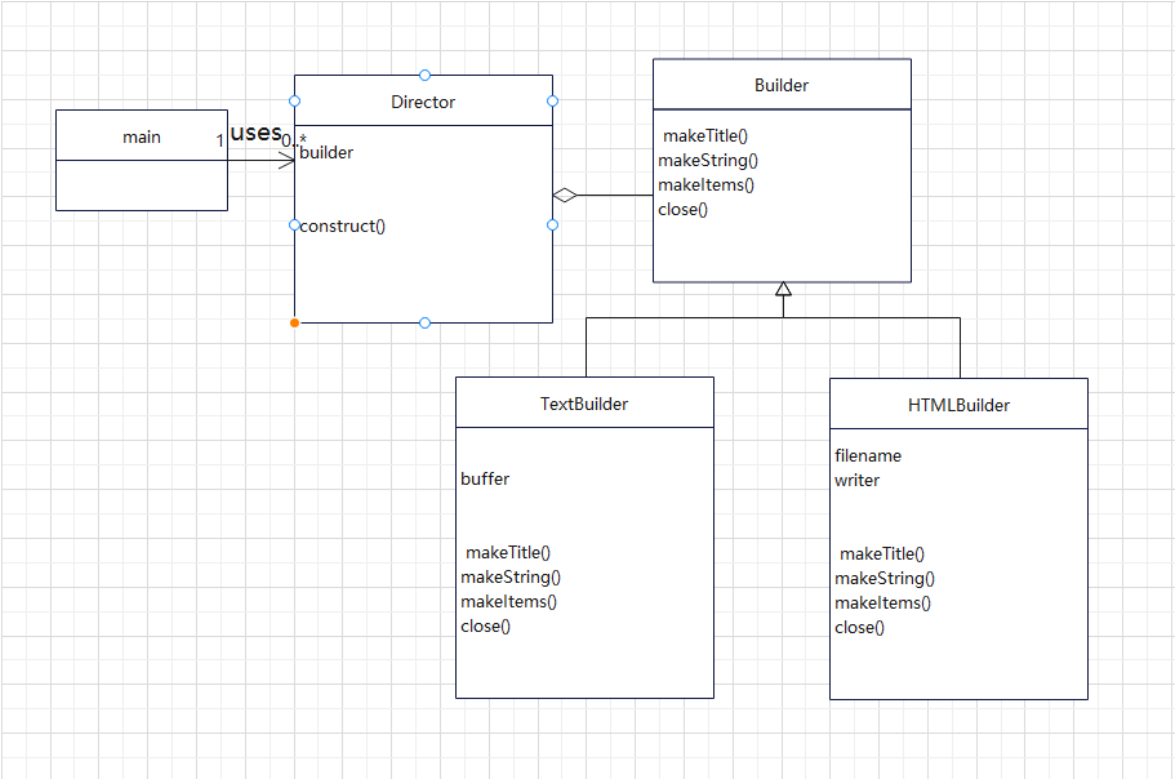
实现：

1. product为抽象类（prototype），提供基本的use，clone接口； 子类需要实现， clone方法是提供克隆自己的方法，c++通过拷贝构造函数来实现；
2. manager为管理类，提供注册接口register ,比如哈希表，map表等，保存 (字符串，需要克隆的对象)二元组，create为索引字符串找到对象指针，然后调用对象的clone方法；
3. manger只与抽象的product有关系，不涉及具体的类；



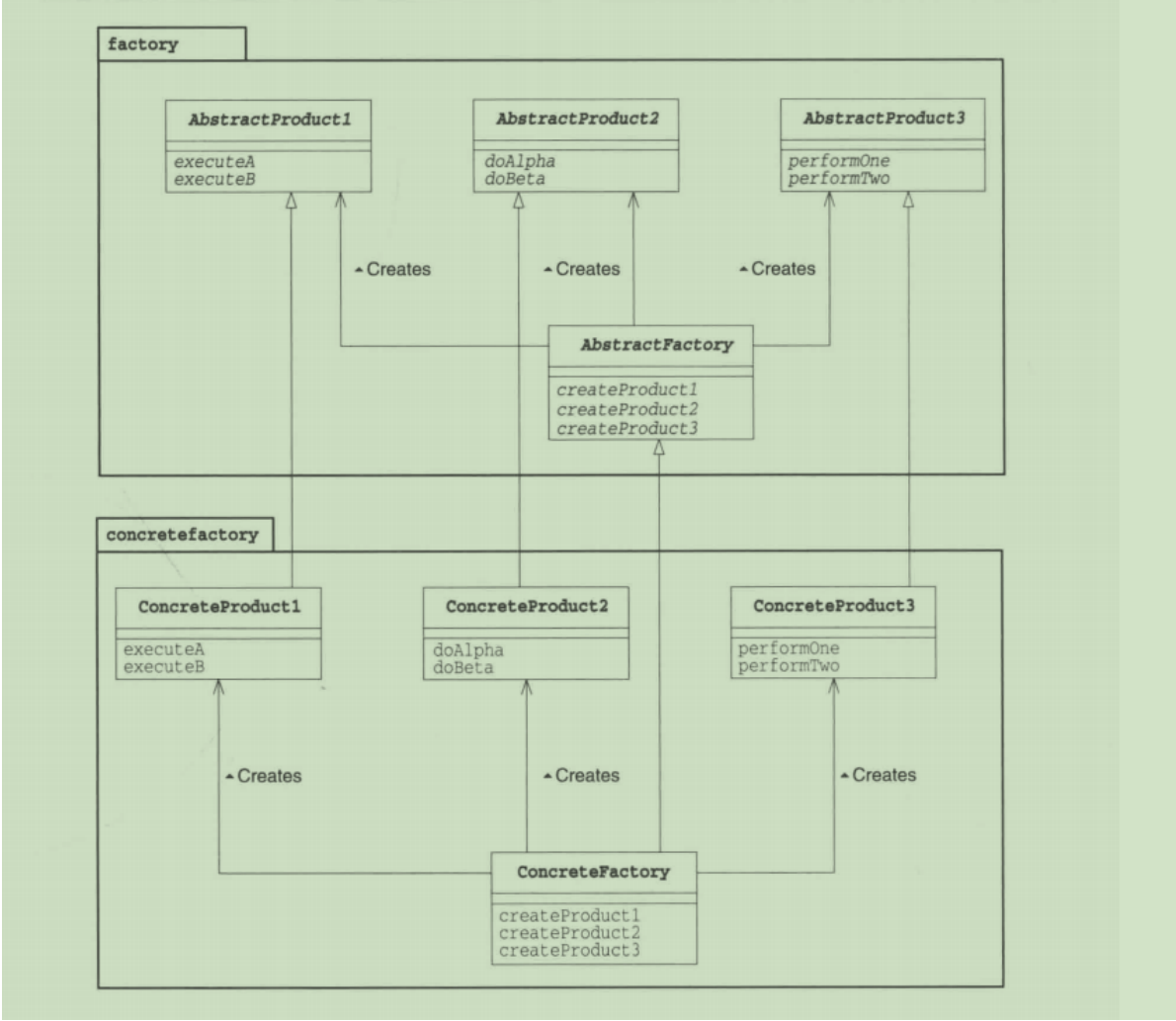
Builder模式

builder模式与模板模式有点相似，模板模式是父类确定了子类的函数调用顺序，builder模式是ditector里确定的；



abstract factory

易于拓展工厂，不利于拓展零件；

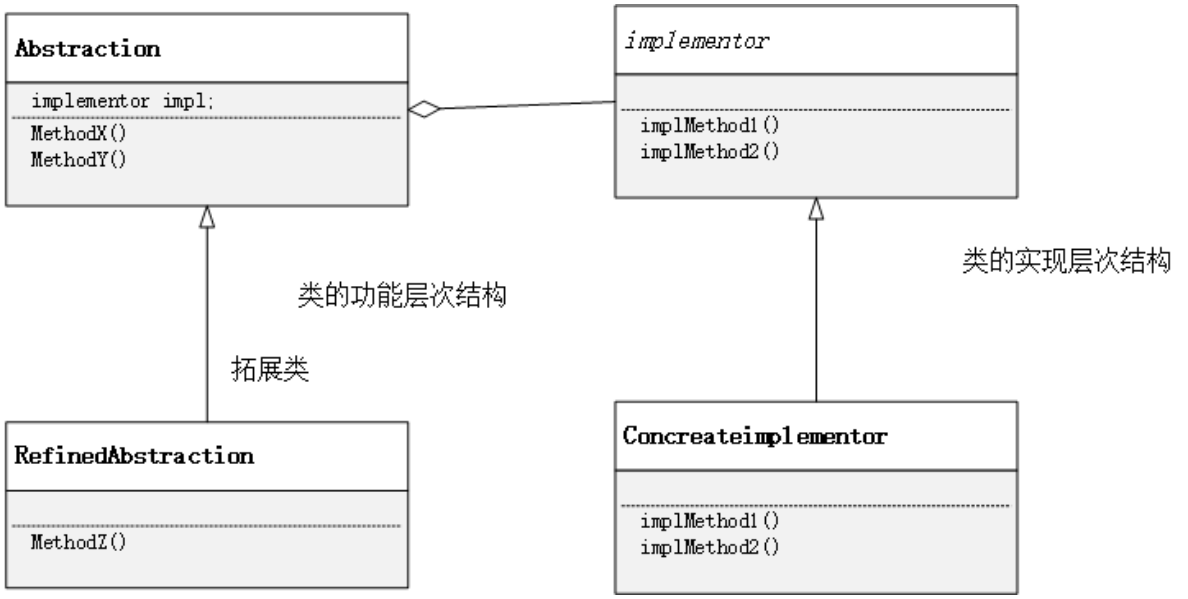


Bridge模式

- 桥接模式：桥接类的功能层次结构与类的实现层次结构；
- 类的功能层次结构：父类具有基本功能，子类增加新的功能；
- 类的实现层次结构：父类声明抽象方法定义接口（API），子类实现具体方法，方法重写；

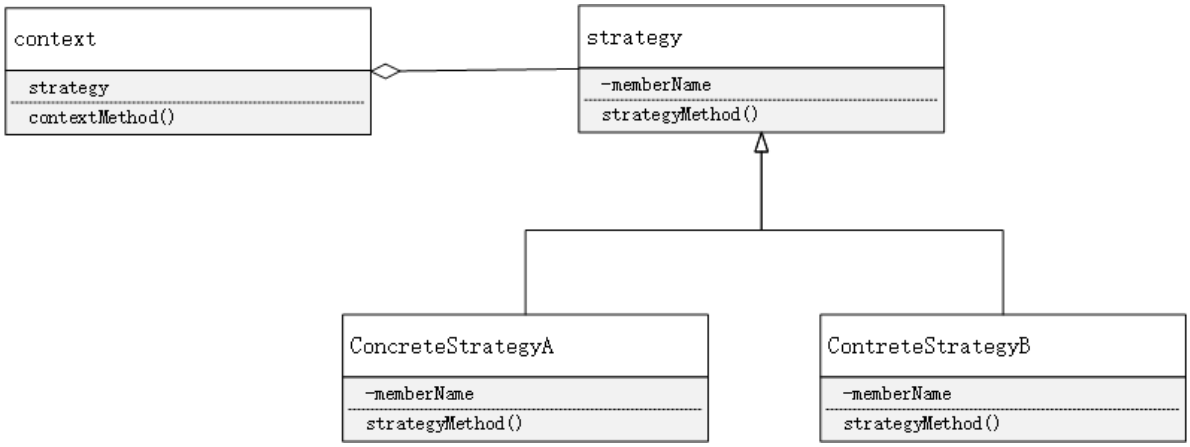
  继承是强关联，委托是弱关联；

  增加新的实现类，不影响功能层次类，反之，亦然。



strategy

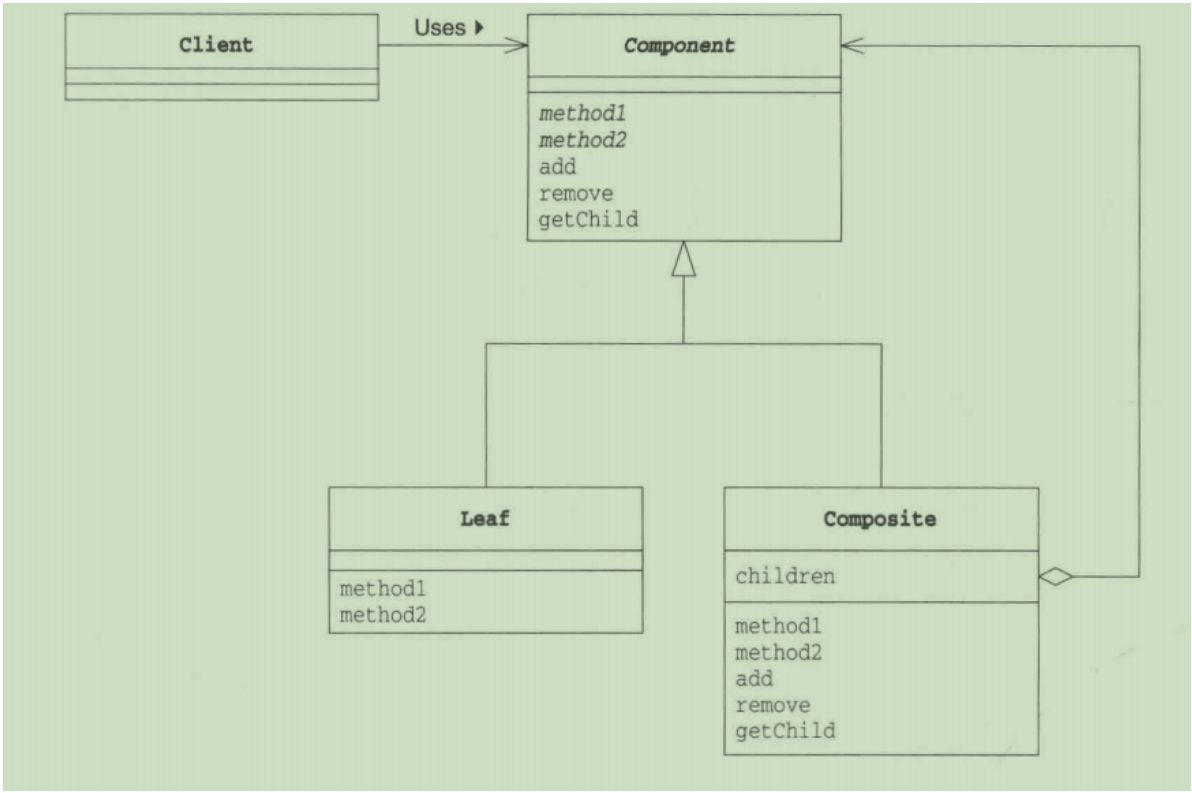
- 策略模式：整体替换算法，使用委托；



composite模式

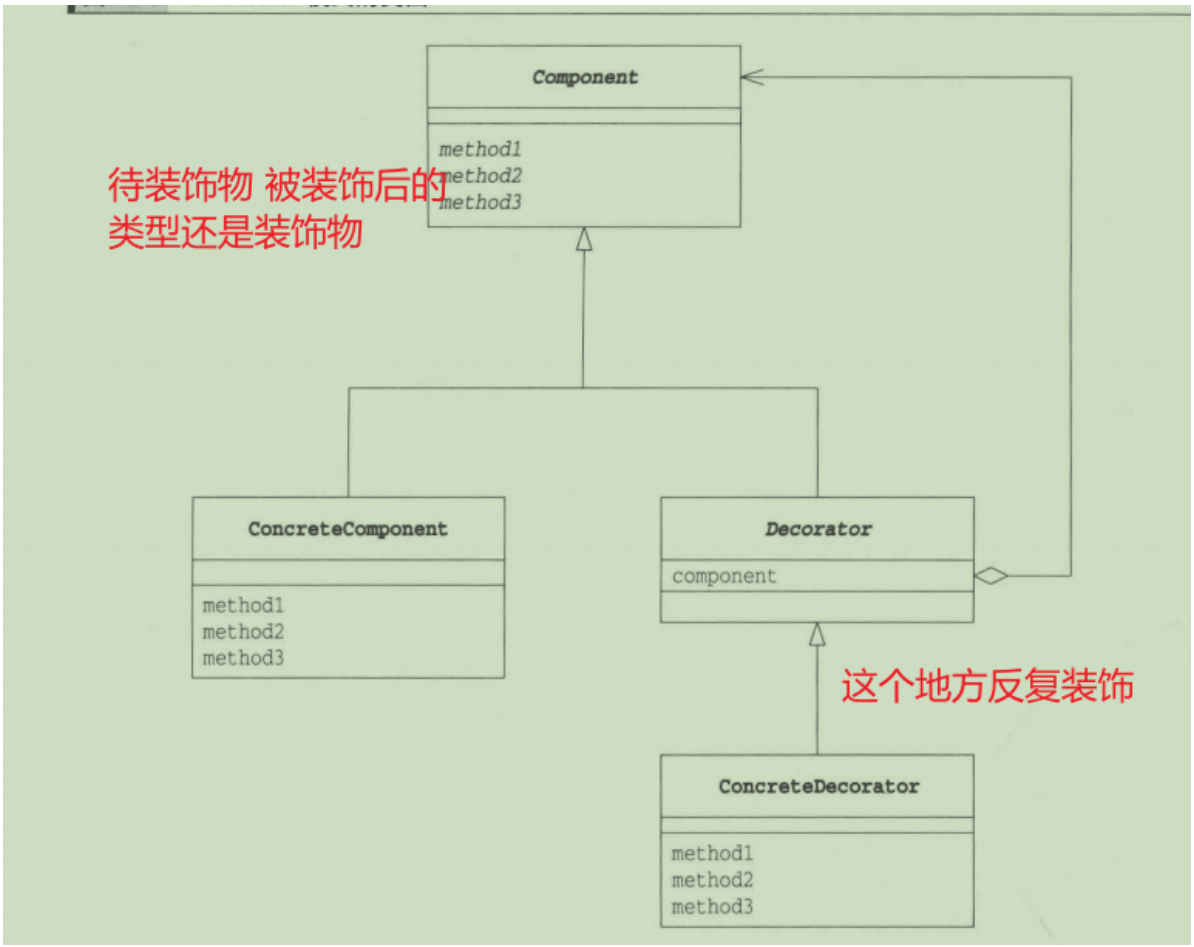
组合模式：使容器与内容具有一致的访问性，例如文件与目录，一般需要递归模式；

类图



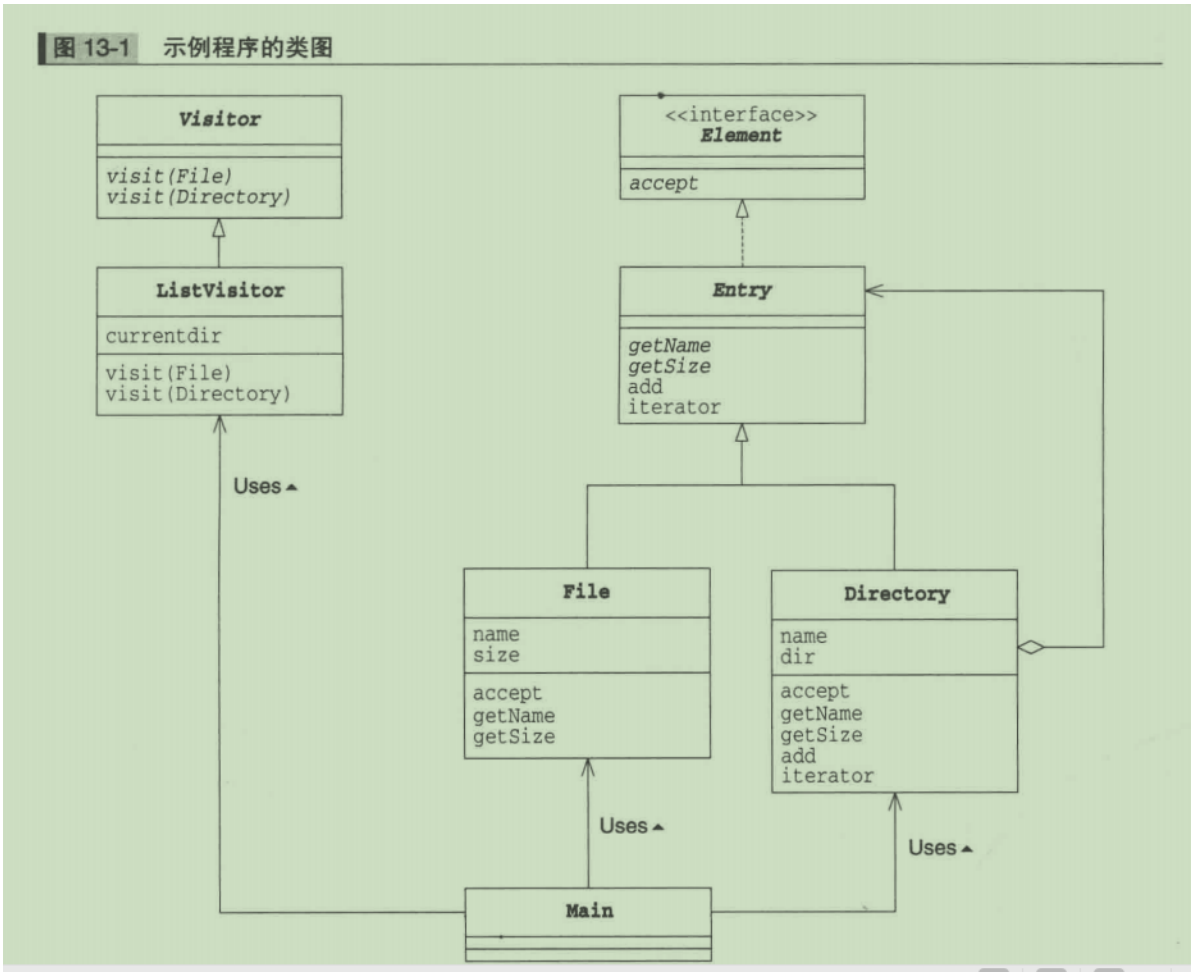
Decorator模式

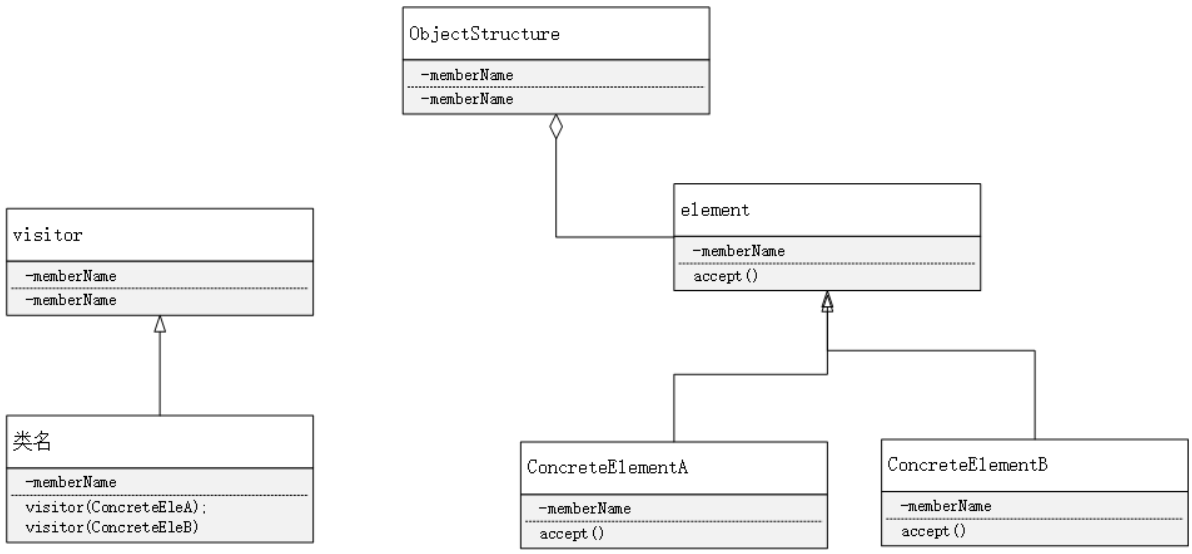
装饰模式：装饰物与被装饰物的一致性，都是一个类型；



visitor模式

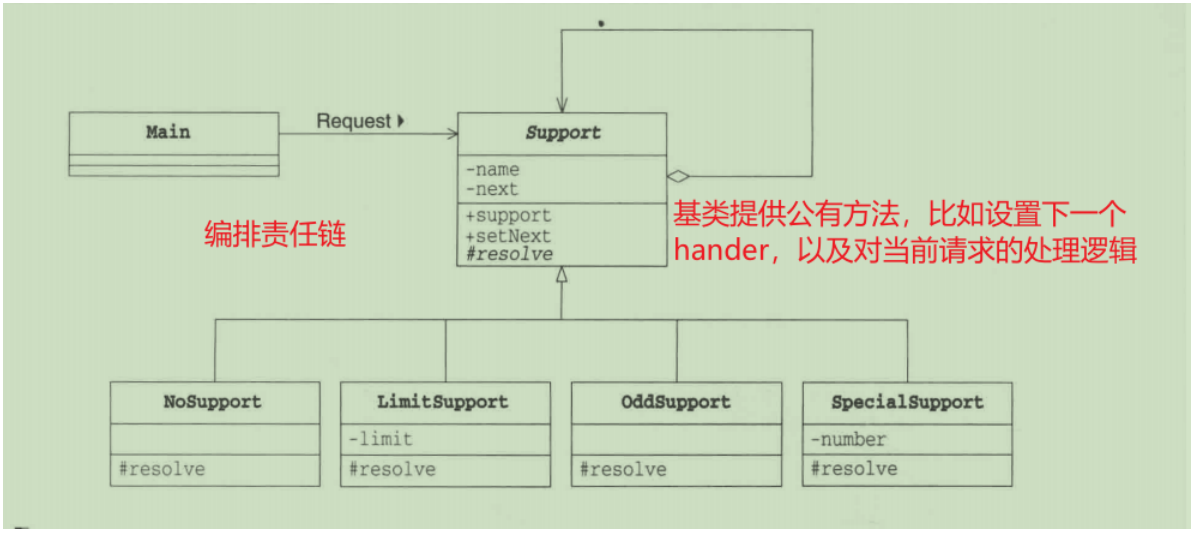
- visitor模式：访问复杂的数据结构，将数据结构与处理分离；
- 双重分发：element提供接受visitor的accept（visitor v）方法；visitor提供访问方法，visitor(element e);
- 适合：增加visitor，不适合增加数据结构，一般要求数据结构是确定的；

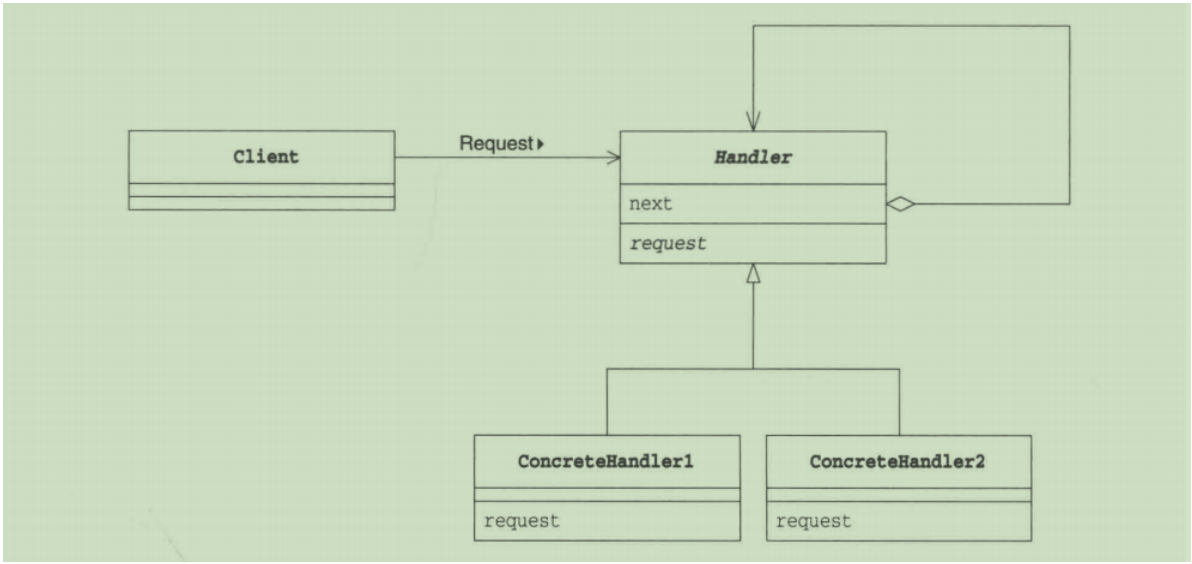




责任链模式

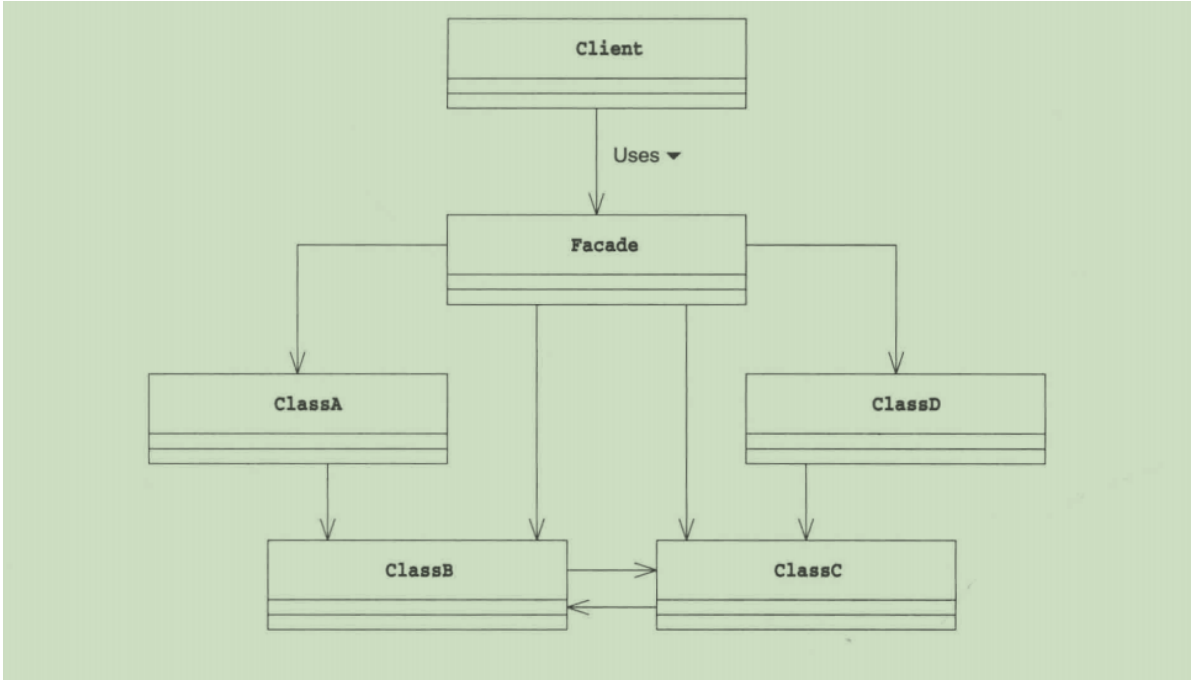
- 责任链模式：多个对象组成一条责任链，按顺序找出谁应该负责；弱化请求方与处理方的关系；
- 缺点：响应不及时，但是问题不大，逻辑判断比较快；





facade模式

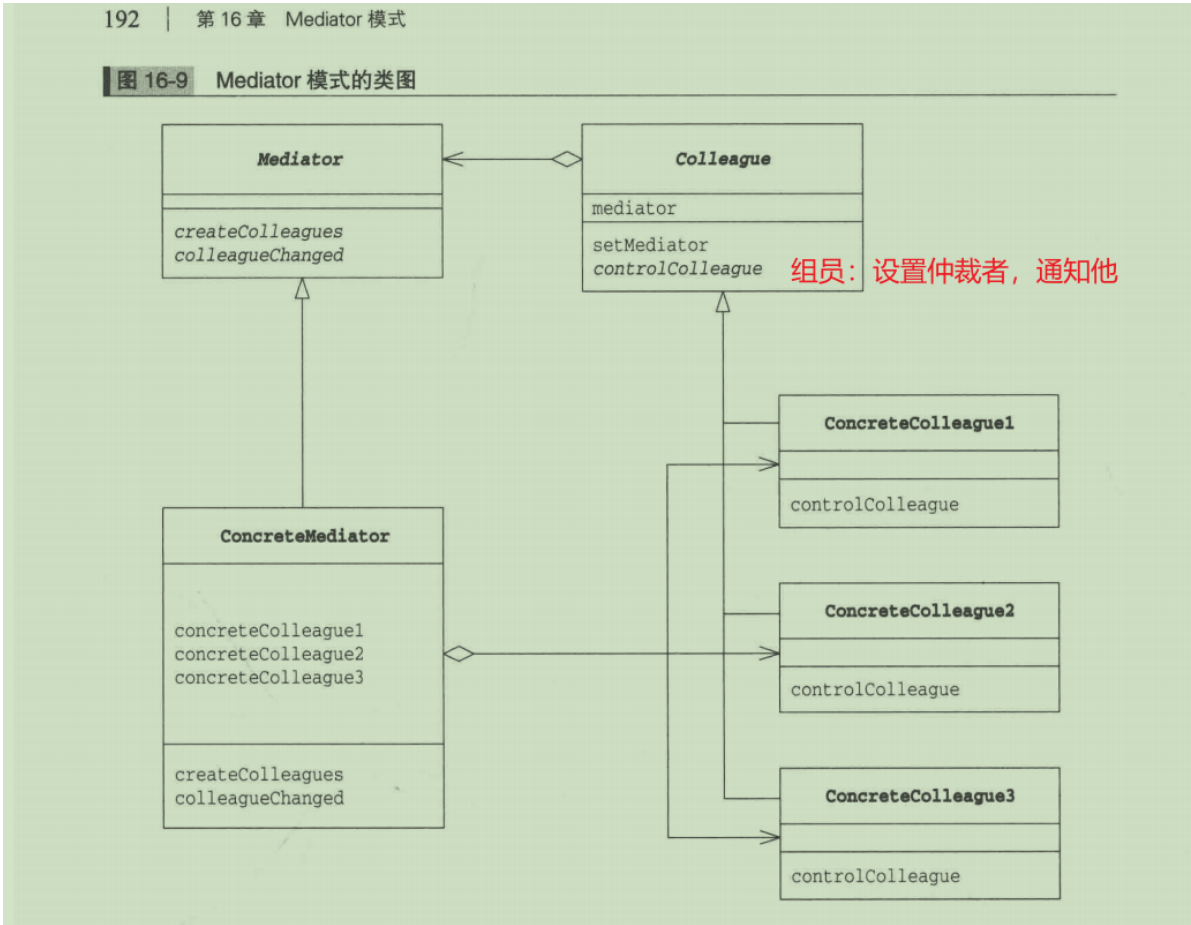
外观模式：提供复杂调用关系的对外API；facade单方向的调用其他类的方法；



Mediator模式

仲裁者模式：组员向仲裁者报告，仲裁者向组员下达指令；组员之间不在相互通信；

与观察者模式很像，组员要通知仲裁者，组员为被观察者；仲裁者为观察者，调用被观察者的方法；



state模式

与策略模式类图一样，将每个状态当做一个类，上下文类负责切换，具体状态类需要知道其他的状态，有点耦合，具体的状态类有点多；

commond模式

参考：https://zhuanlan.zhihu.com/p/22620827

将命令抽象为类，提供execute(), 具体的类实现，比如画颜色命令，画位置命令；命令管理类负责维护一个队列，在增加一个invoker