6. 中介者模式

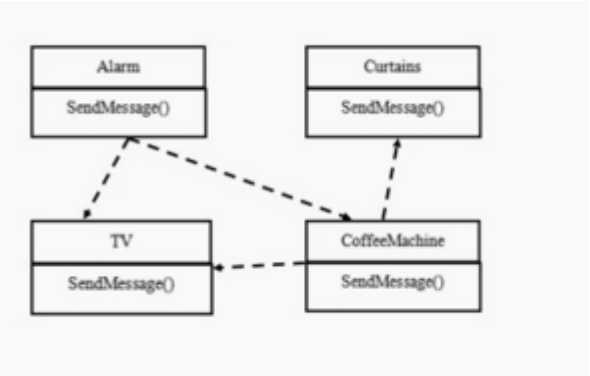
中介者（Mediator）模式的定义：定义一个中介对象来封装一系列对象之间的交互，使原有对象之间的耦合松散，且可以独立地改变它们之间的交互。中介者模式又叫调停模式，它是迪米特法则的典型应用。

智能家庭项目

1） 智能家庭包括各种设备，闹钟、咖啡机、电视机、窗帘 等

2） 主人要看电视时，各个设备可以协同工作，自动完成看电视的准备工作，比如流程为：闹铃响起->咖啡机开始做咖啡->窗帘自动落下->电视机开始播放

传统方案解决智能家庭管理问题



传统的方式的问题分析

1） 当各电器对象有多种状态改变时，相互之间的调用关系会比较复杂

2） 各个电器对象彼此联系，你中有我，我中有你，不利于松耦合.

3） 各个电器对象之间所传递的消息(参数)，容易混乱

4） 当系统增加一个新的电器对象时，或者执行流程改变时，代码的可维护性、扩展性都不理想 考虑中介者模式

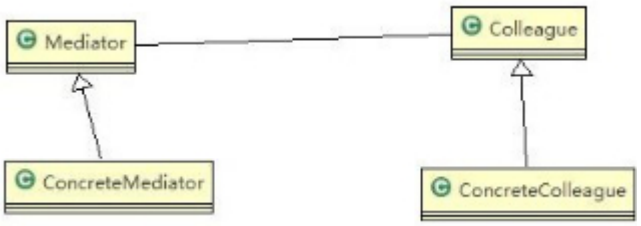
中介者模式基本介绍

1） 中介者模式（Mediator Pattern），用一个中介对象来封装一系列的对象交互。中介者使各个对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互

2） 中介者模式属于行为型模式，使代码易于维护

3） 比如 MVC 模式，C（Controller 控制器）是 M（Model 模型）和 V（View 视图）的中介者，在前后端交互时起到了中间人的作用

中介者模式的原理类图



对原理类图的说明-即(中介者模式的角色及职责)

1） Mediator 就是抽象中介者,定义了同事对象到中介者对象的接口

2） Colleague 是抽象同事类

3） ConcreteMediator 具体的中介者对象，实现抽象方法，他需要知道所有的具体的同事类,即以一个集合来管理

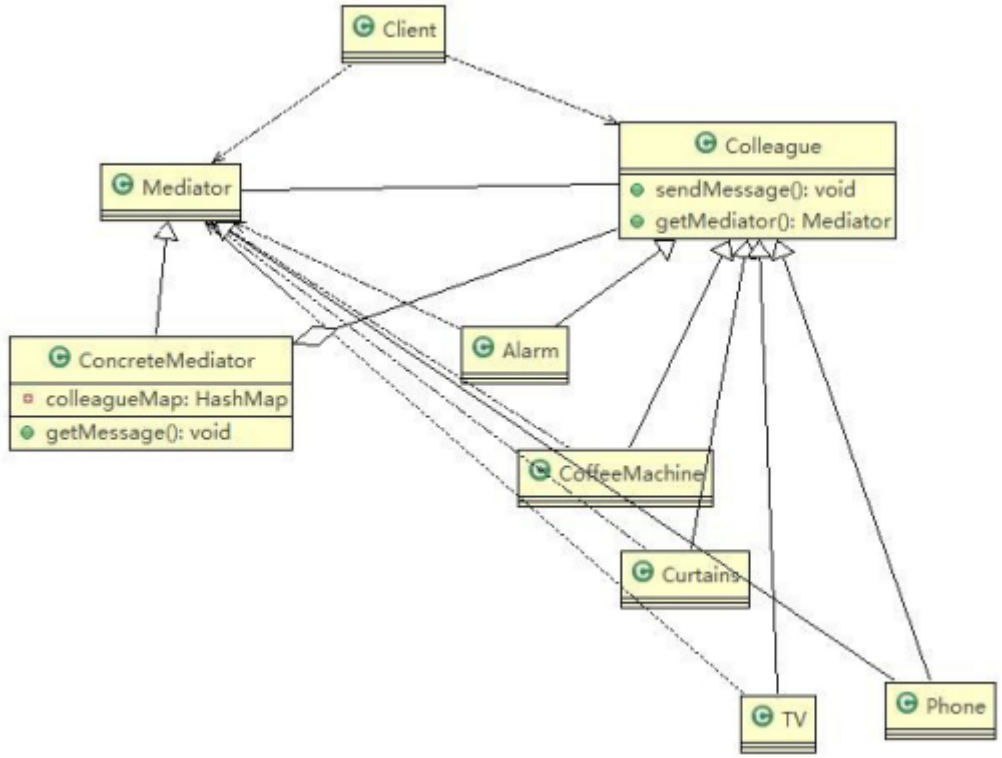
HashMap,并接受某个同事对象消息，完成相应的任务

4） ConcreteColleague 具体的同事类，会有很多，每个同事只知道自己的行为， 而不了解其他同事类的行为(方法)， 但 是他们都依赖中介者对象

中介者模式应用实例-智能家庭管理

1） 应用实例要求

完成前面的智能家庭的项目，使用中介者模式



（代码看代码文件夹，此处引用另一个列子）

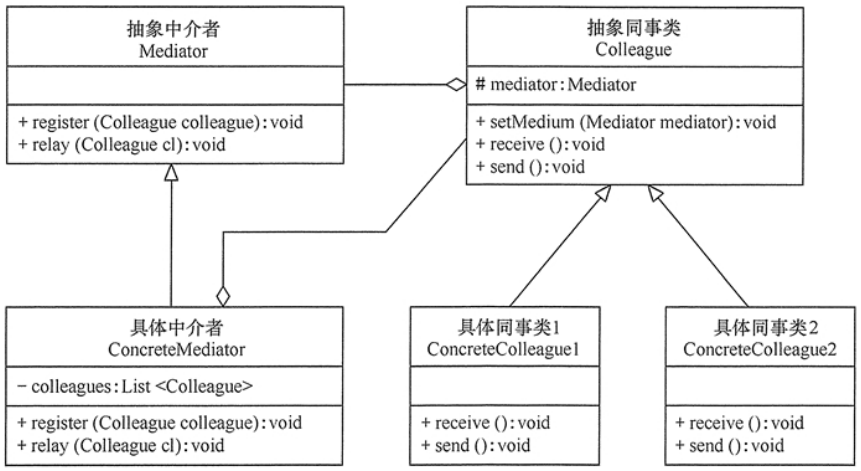


图1 中介者模式的结构图

```
package com.UMediator;

import java.util.*;
public class MediatorPattern
{
    public static void main(String[] args)
    {
        Mediator md=new ConcreteMediator();
        Colleague c1,c2;
        c1=new ConcreteColleague1();
```

```
        c2=new ConcreteColleague2();
        c1.setMedium(md);
        c2.setMedium(md);
        c1.send();
        System.out.println("-------------");
        c2.send();
    }
}
//抽象中介者
abstract class Mediator
{
    public abstract void register(Colleague colleague);
    public abstract void relay(Colleague cl); //转发
}
//具体中介者
class ConcreteMediator extends Mediator
{
    private List<Colleague> colleagues=new ArrayList<Colleague>();
    public void register(Colleague colleague)
    {
        if(!colleagues.contains(colleague))
        {
            colleagues.add(colleague);
        }
    }
    public void relay(Colleague cl)
    {
        for(Colleague ob:colleagues)
        {
            if(!ob.equals(cl))
            {
                ((Colleague)ob).receive();
            }
        }
    }
}
//抽象同事类
abstract class Colleague
{
    protected Mediator mediator;
    public void setMedium(Mediator mediator)
    {
        this.mediator=mediator;
        mediator.register(this);
    }
    public abstract void receive();
    public abstract void send();
```

```
}
//具体同事类
class ConcreteColleague1 extends Colleague
{
    public void receive()
    {
        System.out.println("具体同事类1收到请求。");
    }
    public void send()
    {
        System.out.println("具体同事类1发出请求。");
        mediator.relay(this); //请中介者转发
    }
}
//具体同事类
class ConcreteColleague2 extends Colleague
{
    public void receive()
    {
        System.out.println("具体同事类2收到请求。");
    }
    public void send()
    {
        System.out.println("具体同事类2发出请求。");
        mediator.relay(this); //请中介者转发
    }
}6.
```

中介者模式的注意事项和细节

1） 多个类相互耦合，会形成网状结构，使用中介者模式将网状结构分离为星型结构，进行解耦

2） 减少类间依赖，降低了耦合，符合迪米特原则

3） 中介者承担了较多的责任，一旦中介者出现了问题，整个系统就会受到影响

4） 如果设计不当，中介者对象本身变得过于复杂，这点在实际使用时，要特别注意

模式的应用场景

前面分析了中介者模式的结构与特点，下面分析其以下应用场景。

1. 当对象之间存在复杂的网状结构关系而导致依赖关系混乱且难以复用时。
2. 想创建一个运行于多个类之间的对象，又不想生成新的子类时。

模式的扩展

在实际开发中，通常采用以下两种方法来简化中介者模式，使开发变得更简单。

1. 不定义中介者接口，把具体中介者对象实现成为单例。
2. 同事对象不持有中介者，而是在需要的时直接获取中介者对象并调用。