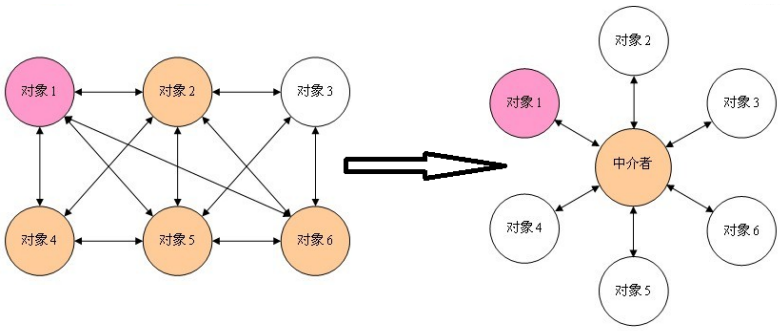
简述

在现实生活中，有很多中介者模式的身影，例如QQ游戏平台，聊天室、QQ群、短信平台和房产中介。不论是QQ游戏还是QQ群，它们都是这个中间平台与其他QQ用户进行交流，如果没有这些中间平台，我们如果想与朋友进行聊天的话，可能就需要当面才可以了。电话、短信t平台，每个用户都不要直接依赖与其他用户，只需要依赖这个中间平台就可以了，一切操作都由中间平台去分发

中介者模式（Mediator Pattern）是用来降低多个对象和类之间的通信复杂性。这种模式提供了一个中介类，该类通常处理不同类之间的通f

使用场景

对象与对象之间存在大量的关联关系，这样势必会导致系统的结构变得很复杂，多个类相互耦合，形成了网状结构，引入中介者将结构关系进行梳理



使用实例

MVC 框架，其中C（控制器）就是 M（模型）和 V（视图）的中介者

代码

模拟一个斗地主的游戏，三个人玩，其中一个人赢了，那么另外两个人就输了，赢得归赢得，输的一人一半

打牌者抽象类

```
public abstract class CardFriend {

    protected String name;
    protected int money;

    public CardFriend(String name, int money){
        this.money = money;
        this.name = name;
        System.out.println(name + "带了" + money + "来打牌");
     }

     // 由中介者来计算金额
     abstract void win(int money, AbsMediator mediator);
}
```

打牌者A

```
public class CardFriendA extends CardFriend {

    public CardFriendA(String name, int money) {
        super(name, money);
    }

    @Override
    void win(int money, AbsMediator mediator) {
        mediator.winA(money);
     }
}
```

打牌者B

```
public class CardFriendB extends CardFriend {
    public CardFriendB(String name, int money) {
        super(name, money);
    }

    @Override
    void win(int money, AbsMediator mediator) {
        mediator.winB(money);
    }

}
```

打牌者C

```
public class CardFriendC extends CardFriend {
    public CardFriendC(String name, int money) {
        super(name, money);
    }

    @Override
    void win(int money, AbsMediator mediator) {
```

```
        mediator.winC(money);
    }

}
```

抽象中介者类

有处理三个人各自赢了的方法

```
public abstract class AbsMediator {

    protected CardFriendA cardFriendA;
    protected CardFriendB cardFriendB;
    protected CardFriendC cardFriendC;

    public AbsMediator(CardFriendA a, CardFriendB b, CardFriendC c){
        this.cardFriendA = a;
        this.cardFriendB = b;
         this.cardFriendC = c;
     }

     public abstract void winA(int money);

     public abstract void winB(int money);

     public abstract void winC(int money);

}
```

具体中介者类

```
public class Mediator extends AbsMediator {
    public Mediator(CardFriendA a, CardFriendB b, CardFriendC c) {
        super(a, b, c);
    }

    @Override
    public void winA(int money) {
        int half = money / 2;
        int other = money - half;

         cardFriendA.money = cardFriendA.money + money;
         cardFriendB.money = cardFriendB.money - half;
         cardFriendC.money = cardFriendC.money - other;
         System.out.println("a 赢了 " + money + " , b 输了 " + half + " , c 输了 " + other);
     }

     @Override
     public void winB(int money) {
         int half = money / 2;
         int other = money - half;

         cardFriendB.money = cardFriendB.money + money;
         cardFriendC.money = cardFriendC.money - half;
         cardFriendA.money = cardFriendA.money - other;
         System.out.println("a 输了 " + other + " , b 赢了 " + money + " , c 输了 " + half);
```

```
    }

    @Override
    public void winC(int money) {
        int half = money / 2;
        int other = money - half;

        cardFriendC.money = cardFriendC.money + money;
        cardFriendA.money = cardFriendA.money - half;
        cardFriendB.money = cardFriendB.money - other;
        System.out.println("a 输了 " + half + " , b 输了 " + other + " , c 赢了 " + money);
    }
}
```

调用者

```
public class Main {

    public static void main(String[] args) {

        CardFriendA a = new CardFriendA("张三", 1000);
        CardFriendB b = new CardFriendB("李四", 1200);
        CardFriendC c = new CardFriendC("王五", 1300);


        System.out.println("中介者来了");
        AbsMediator mediator = new Mediator(a, b, c);

        Random random = new Random();
        for (int i = 0; i < 3; i++) {
            System.out.println(MessageFormat.format("开始第{0}盘，A的money是{1}，B的money是{2}，C的mone
            int money = random.nextInt(100);
            int winner = random.nextInt(3);
            if (winner == 0){
                a.win(money, mediator);
            }else if (winner == 1){
                b.win(money, mediator);
            }else {
                c.win(money, mediator);
            }
            System.out.println();
        }
    }

}
```

代码简述

打牌者谁赢了，谁调用赢了方法去通知中介者，它赢了多少。 中介者去自动计算 胜利者现在的金额 和 赢得金额。 以及其他输了的人的金

总结

优点：

- 简化了对象之间的关系，将系统的各个对象之间的相互关系进行封装，将各个同事类解耦，使得系统变为松耦合。
- 提供系统的灵活性，使得各个同事对象独立而易于复用。

缺点：

- 中介者模式中，中介者角色承担了较多的责任，所以一旦这个中介者对象出现了问题，整个系统将会受到重大的影响。
- 新增加一个同事类时，不得不去修改抽象中介者类和具体中介者类，此时可以使用观察者模式和状态模式来解决这个问题