

中介者模式

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信，并支持松耦合，使代码易于维护。中介者模式属于行为型模式。

（对象类与对象类之间的交互通信统一由另外一个中介类来控制，对象通过中介类对其他对象交互，中介类起着控制器的作用。）

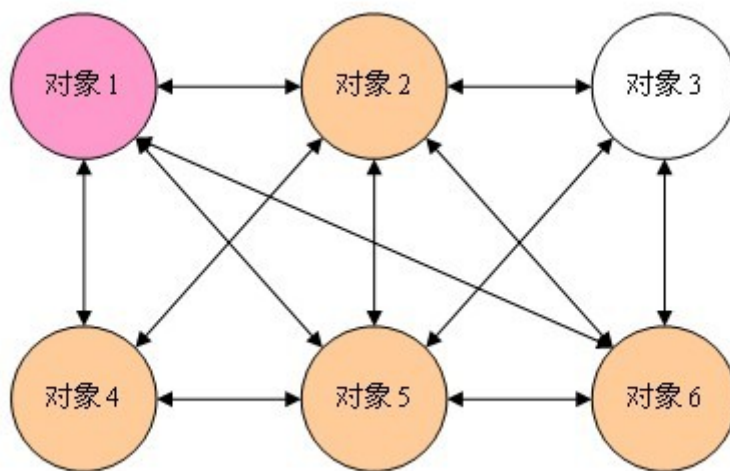
介绍

意图：

用一个中介对象来封装一系列的对象交互，中介者使各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。

为什么要使用中介者模式 (多个类相互耦合, 形成了网状结构。)

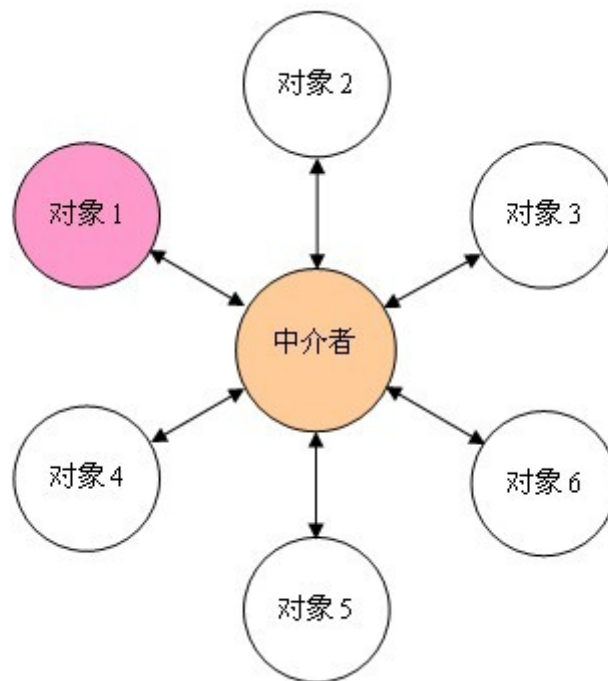
一般来说, 同事类之间的关系是比较复杂的, 多个同事类之间互相关联时, 他们之间的关系会呈现为复杂的网状结构, 这是一种过度耦合的架构, 即不利于类的复用, 也不稳定。例如在下图中, 有六个同事类对象, 假如对象1发生变化, 那么将会有4个对象受到影响。如果对象2发生变化, 那么将会有5个对象受到影响。也就是说, 同事类之间直接关联的设计是不好的。



如何解决: (将上述网状结构分离为星型结构。)

如果引入中介者模式, 那么同事类之间的关系将变为星型结构, 从图中可以看到, 任何一个类的变动, 只会影响的类本身, 以及中介者, 这样就减小了系统的耦合。一个好的设计, 必定不会把所有的对象关系处理逻辑封

装在本类中，而是使用一个专门的类来管理那些不属于自己的行为。



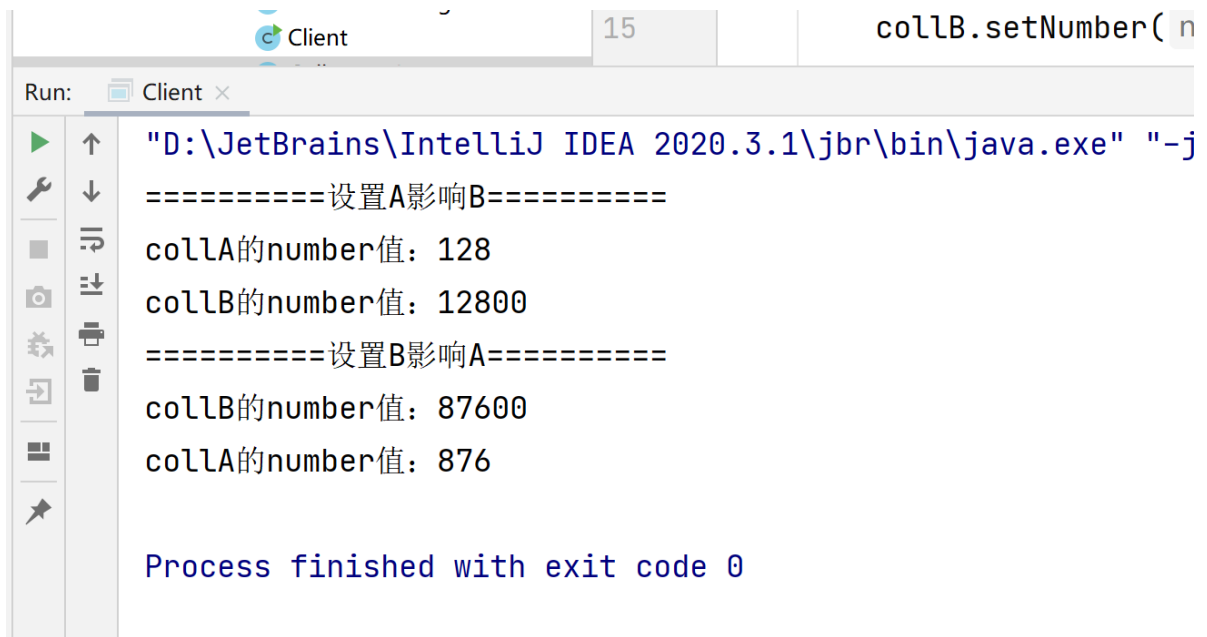
我们使用一个例子来说明一下什么是同事类：有两个类A和B，类中各有一个数字，并且要保证类B中的数字永远是类A中数字的100倍。也就是说，当修改类A的数时，将这个数乘以100赋给类B，而修改类B时，要将数除以100赋给类A。类A类B互相影响，就称为同事类。代码如下：

```
1
2    //抽象同事类
3    abstract class AbstractColleague {
4        protected int number;
5
6        public int getNumber() {
7            return number;
```

```
8      }
9
10     public void setNumber(int number){
11         this.number = number;
12     }
13     //抽象方法，修改数字时同时修改关联对象
14     public abstract void setNumber(int
number, AbstractColleague coll);
15 }
16
17
18 class ColleagueA extends AbstractColleague{
19     public void setNumber(int number,
AbstractColleague coll) {
20         this.number = number;
21         coll.setNumber(number*100);
22     }
23 }
24
25
26
27 class ColleagueB extends AbstractColleague{
28
29     public void setNumber(int number,
AbstractColleague coll) {
30         this.number = number;
31         coll.setNumber(number/100);
32     }
33 }
34
```

```
35  //实现
36  public class Client {
37      public static void main(String[] args){
38
39          AbstractColleague collA = new
ColleagueA();
40          AbstractColleague collB = new
ColleagueB();
41
42          System.out.println("=====设置A
影响B=====");
43          collA.setNumber(1288, collB);
44          System.out.println("collA的number
值: "+collA.getNumber());
45          System.out.println("collB的number
值: "+collB.getNumber());
46
47          System.out.println("=====设置B
影响A=====");
48          collB.setNumber(87635, collA);
49          System.out.println("collB的number
值: "+collB.getNumber());
50          System.out.println("collA的number
值: "+collA.getNumber());
51      }
52  }
```

结果



```
Client 15 collB.setNumber(n

Run: Client x
"D:\JetBrains\IntelliJ IDEA 2020.3.1\jbr\bin\java.exe" "-j
=====设置A影响B=====
collA的number值: 128
collB的number值: 12800
=====设置B影响A=====
collB的number值: 87600
collA的number值: 876

Process finished with exit code 0
```

上面的代码中，类A类B通过直接的关联发生关系，假如我们要使用中介者模式，类A类B之间则不可以直接关联，他们之间必须要通过一个中介者来达到关联的目的。

```
1  abstract class AbstractColleague {
2      protected int number;
3
4      public int getNumber() {
5          return number;
6      }
7
8      public void setNumber(int number){
9          this.number = number;
10     }
11     //注意这里的参数不再是同事类，而是一个中介者
```

```
12     public abstract void setNumber(int
    number, AbstractMediator am);
13 }
14
15 class ColleagueA extends AbstractColleague{
16
17     public void setNumber(int number,
    AbstractMediator am) {
18         this.number = number;
19         am.AaffectB();
20     }
21 }
22
23 class ColleagueB extends AbstractColleague{
24
25     @Override
26     public void setNumber(int number,
    AbstractMediator am) {
27         this.number = number;
28         am.BaffectA();
29     }
30 }
31
32 abstract class AbstractMediator {
33     protected AbstractColleague A;
34     protected AbstractColleague B;
35
36     public
    AbstractMediator(AbstractColleague a,
    AbstractColleague b) {
```

```
37         A = a;
38         B = b;
39     }
40
41     public abstract void AaffectB();
42
43     public abstract void BaffectA();
44
45 }
46 class Mediator extends AbstractMediator {
47
48     public Mediator(AbstractColleague a,
49 AbstractColleague b) {
50         super(a, b);
51     }
52
53     //处理A对B的影响
54     public void AaffectB() {
55         int number = A.getNumber();
56         B.setNumber(number*100);
57     }
58
59     //处理B对A的影响
60     public void BaffectA() {
61         int number = B.getNumber();
62         A.setNumber(number/100);
63     }
64 }
65 public class Client {
```



```
66     public static void main(String[] args){
67         AbstractColleague collA = new
ColleagueA();
68         AbstractColleague collB = new
ColleagueB();
69
70         AbstractMediator am = new
Mediator(collA, collB);
71
72         System.out.println("=====通过设
置A影响B=====");
73         collA.setNumber(1000, am);
74         System.out.println("collA的number值
为: "+collA.getNumber());
75         System.out.println("collB的number值
为A的10倍: "+collB.getNumber());
76
77         System.out.println("=====通过设
置B影响A=====");
78         collB.setNumber(1000, am);
79         System.out.println("collB的number值
为: "+collB.getNumber());
80         System.out.println("collA的number值
为B的0.1倍: "+collA.getNumber());
81
82     }
83 }
```

结果

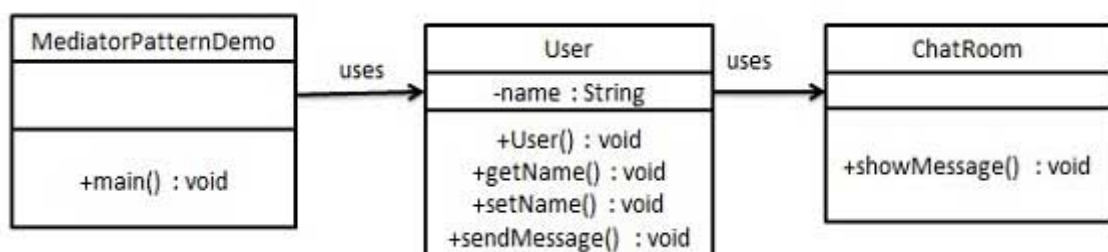
```
"D:\JetBrains\IntelliJ IDEA 2020.3.1\jbr\bin\java
=====通过设置A影响B=====
collA的number值为: 1000
collB的number值为A的10倍: 100000
=====通过设置B影响A=====
collB的number值为: 1000
collA的number值为B的0.1倍: 10

Process finished with exit code 0
```

实现

我们通过聊天室实例来演示中介者模式。实例中，多个用户可以向聊天室发送消息，聊天室向所有的用户显示消息。我们将创建两个类 *ChatRoom* 和 *User*。*User* 对象使用 *ChatRoom* 方法来分享他们的消息。

MediatorPatternDemo，我们的演示类使用 *User* 对象来显示他们之间的通信



1.创建中介类。

ChatRoom.java

```
1  import java.util.Date;
2
3  public class ChatRoom {
4      public static void showMessage(User user,
5      String message){
6          System.out.println(new
7      Date().toString()
8      + " [" + user.getName() + "] : " +
9      message);
10     }
11 }
```

2.创建 user 类。

User.java

```
1  public class User {
2      private String name;
3
4      public String getName() {
5          return name;
6      }
7  }
```

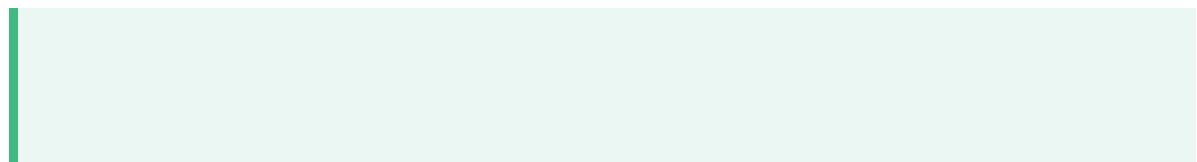
```
8     public void setName(String name) {
9         this.name = name;
10    }
11
12    public User(String name){
13        this.name = name;
14    }
15
16    public void sendMessage(String message){
17        ChatRoom.showMessage(this,message);
18    }
19 }
```

3.使用 *User* 对象来显示他们之间的通信

MediatorPatternDemo.java

```
1  public class MediatorPatternDemo {
2      public static void main(String[] args) {
3          User robert = new User("张三");
4          User john = new User("李四");
5
6          robert.sendMessage("Hi! 李四!");
7          john.sendMessage("Hello! 张三!");
8      }
9  }
```

4.执行程序，输出结果：



```
run: MediatorPatternDemo x
"D:\JetBrains\IntelliJ IDEA 2020.3.1\jbr\bin\java.exe" "-javaagent:
Wed Mar 30 12:41:37 CST 2022 [张三] : Hi! 李四!
Wed Mar 30 12:41:38 CST 2022 [李四] : Hello! 张三!

Process finished with exit code 0
```

优点

- 减少类间的依赖，将原有的一对多的依赖变成一对一的依赖，是的对象之间的关系更易维护和理解。
- 避免同事之间过度耦合，同事类只依赖于中介者，使同事类更易被复用，中介类和同事类可以相对独立地演化。
- 中介者模式将对象的行为和协作抽象化，将对象在小尺度的行为上与其他对象的相互作用分开处理

缺点

- 中介者模式降低了同事对象的复杂性，但增加了中介者类的复杂性。
- 中介者类经常充满了各个具体同事类的关系协调代码，这种代码是不能复用的。

主要解决：

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，同时若一个对象发生改变，我们也需要跟踪与之相关联的对象，同时做出相应的处理。

使用场景

- 系统中对象之间存在比较复杂的引用关系，导致它们之间的依赖关系结构混乱而且难以复用该对象。
- 想通过一个中间类来封装多个类中的行为，而又不想生成太多的子类。

应用实例：

- 1、中国加入 WTO 之前是各个国家相互贸易，结构复杂，现在是各个国家通过 WTO 来互相贸易。
- 2、机场调度系统。
- 3、MVC 框架，其中C（控制器）就是 M（模型）和 V（视图）的中介者。