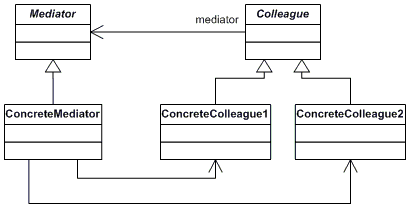
**一、Mediator Pattern (调解者模式)的定义**

Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

定义一个对象将一系列对象如何相互作用的操作进行封装。调解者通过避免一个对象明确指向其余的对象来促使自由耦合，同时能够让你单独的改变他们之间的相互作用。

**二、UML类图**



 **Mediator**  **(IChatroom)**

* 定义一个接口用于与同事级对象进行通信

 **ConcreteMediator**  **(Chatroom)**

* 通过与同事级对象进行对等来实现相互协作行为
* 知道并且维持本身的同事级对象

 **Colleague classes**  **(Participant)**

* 每个同事级对象都知道自己的调节者对象
* 每个同事级对象在与另外的同事级对象进行通信时必须要与调节者进行通信。

mediator

    中介者类的抽象父类。

concreteMediator

    具体的中介者类。

colleague

   关联类的抽象父类。

concreteColleague

    具体的关联类。  
  
**四、中介者模式的优点**  
1. 将系统按功能分割成更小的对象，符合类的最小设计原则  
2. 对关联对象的集中控制  
3. 减小类的耦合程度，明确类之间的相互关系：当类之间的关系过于复杂时，其中任何一个类的修改都会影响到其他类，不符合类的设计的开闭原则 ，而Mediator模式将原来相互依存的多对多的类之间的关系简化为Mediator控制类与其他关联类的一对多的关系，当其中一个类修改时，可以对其他关联类不产生影响（即使有修改，也集中在Mediator控制类）。  
4. 有利于提高类的重用性

**未运用中介模式前的实现：**

**[java]** [view plaincopy](http://blog.csdn.net/liuwenbo0920/article/details/7275911)

1. Person.java
2. **package** com.ibeifeng.ex1;
4. **public** **abstract** **class** Person {
5. **private** String name;
6. **private** **int** condition;
8. **public** Person(String name, **int** condition) {
9. **this**.name = name;
10. **this**.condition = condition;
11. }
13. **public** String getName() {
14. **return** name;
15. }
16. **public** **void** setName(String name) {
17. **this**.name = name;
18. }
19. **public** **int** getCondition() {
20. **return** condition;
21. }
23. **public** **void** setCondition(**int** condition) {
24. **this**.condition = condition;
25. }
27. **public** **abstract** **void** getPartner(Person person);
29. }
30. Man.java
31. **package** com.ibeifeng.ex1;
33. **public** **class** Man **extends** Person {
35. **public** Man(String name, **int** condition) {
36. **super**(name, condition);
37. }
39. **public** **void** getPartner(Person person) {
40. **if**(person **instanceof** Man) {
41. System.out.println("汗，我不是同性恋！");
42. } **else** {
43. **if**(**this**.getCondition() == person.getCondition()) {
44. System.out.println(**this**.getName() + "和" + person.getName() + "绝配");
45. } **else** {
46. System.out.println(**this**.getName() + "和" + person.getName() + "不相配");
47. }
48. }
49. }
51. }
52. Woman.java
53. **package** com.ibeifeng.ex1;
55. **public** **class** Woman **extends** Person {
57. **public** Woman(String name, **int** condition) {
58. **super**(name, condition);
59. }
61. **public** **void** getPartner(Person person) {
62. **if**(person **instanceof** Woman) {
63. System.out.println("汗，我不是同性恋！");
64. } **else** {
65. **if**(**this**.getCondition() == person.getCondition()) {
66. System.out.println(**this**.getName() + "和" + person.getName() + "绝配");
67. } **else** {
68. System.out.println(**this**.getName() + "和" + person.getName() + "不相配");
69. }
70. }
71. }
73. }
74. MainClass.java
75. **package** com.ibeifeng.ex1;
77. **public** **class** MainClass {
78. **public** **static** **void** main(String[] args) {
79. Person zhangsan = **new** Man("张三",5);
80. Person lisi = **new** Man("李四",6);
82. Person xiaofang = **new** Woman("小芳", 6);
84. zhangsan.getPartner(xiaofang);
85. lisi.getPartner(xiaofang);
86. zhangsan.getPartner(lisi);
88. }
89. }

**运用了中介模式的代码：**

**[java]** [view plaincopy](http://blog.csdn.net/liuwenbo0920/article/details/7275911)

1. Person.java
2. **package** com.ibeifeng.ex2;
4. **public** **abstract** **class** Person {
5. **private** String name;
6. **private** **int** condition;
7. **private** Mediator mediator;

10. **public** Person(String name, **int** condition, Mediator mediator) {
11. **super**();
12. **this**.name = name;
13. **this**.condition = condition;
14. **this**.mediator = mediator;
15. }
17. **public** Mediator getMediator() {
18. **return** mediator;
19. }
21. **public** **void** setMediator(Mediator mediator) {
22. **this**.mediator = mediator;
23. }
25. **public** String getName() {
26. **return** name;
27. }
28. **public** **void** setName(String name) {
29. **this**.name = name;
30. }
31. **public** **int** getCondition() {
32. **return** condition;
33. }
35. **public** **void** setCondition(**int** condition) {
36. **this**.condition = condition;
37. }
39. **public** **abstract** **void** getPartner(Person person);
41. }
42. Mediator.java
43. **package** com.ibeifeng.ex2;
45. **public** **class** Mediator {
46. **private** Man man;
47. **private** Woman woman;
49. **public** **void** setMan(Man man) {
50. **this**.man = man;
51. }
52. **public** **void** setWoman(Woman woman) {
53. **this**.woman = woman;
54. }
56. **public** **void** getPartner(Person person) {
57. //将搭档设置上
58. **if**(person **instanceof** Man) {
59. **this**.setMan((Man)person);
60. } **else** {
61. **this**.setWoman((Woman)person);
62. }
63. //判断条件
64. **if**(man == **null** || woman == **null**) {
65. System.out.println("汗，我不是同性恋!");
66. } **else** {
68. **if**(man.getCondition() == woman.getCondition()) {
69. System.out.println(man.getName() + "和" + woman.getName() + "绝配");
70. } **else** {
71. System.out.println(man.getName() + "和" + woman.getName() + "不相配");
72. }
73. }
74. }
76. }
77. Man.java
78. **package** com.ibeifeng.ex2;
80. **public** **class** Man **extends** Person {
82. **public** Man(String name, **int** condition,Mediator mediator) {
83. **super**(name, condition, mediator);
84. }
86. **public** **void** getPartner(Person person) {
87. **this**.getMediator().setMan(**this**);
88. **this**.getMediator().getPartner(person);
89. }
90. }
91. Woman.java
92. **package** com.ibeifeng.ex2;
94. **public** **class** Woman **extends** Person {
96. **public** Woman(String name, **int** condition,Mediator mediator) {
97. **super**(name, condition, mediator);
98. }
100. **public** **void** getPartner(Person person) {
101. **this**.getMediator().setWoman(**this**);
102. **this**.getMediator().getPartner(person);
103. }
105. }
106. MainClass.java
107. **package** com.ibeifeng.ex2;
108. **public** **class** MainClass {
109. **public** **static** **void** main(String[] args) {
110. Mediator mediator = **new** Mediator();
111. Person zhangsan = **new** Man("张三",7,mediator);
112. Person lisi = **new** Man("李四",7,mediator);
113. Person xiaofang = **new** Woman("小芳",7,mediator);
115. zhangsan.getPartner(lisi);
117. xiaofang.getPartner(lisi);
118. }
119. }

# [中介者模式(Mediator pattern)](http://www.cnblogs.com/lds85930/articles/643155.html)

**一、引子**   
  
　　中介在现实生活中并不陌生，满大街的房屋中介、良莠不齐的出国中介……。它们的存在是因为它们能给我们的生活带来一些便利：租房、买房用不着各个小区里瞎转；出国留学也不用不知所措。  
  
　　中介者模式在程序设计中也起到了类似的作用。  
  
　　**二、定义与结构**  
  
　　GOF给中介者模式下的定义是：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。简单点来说，将原来两个直接引用或者依赖的对象拆开，在中间加入一个“中介”对象，使得两头的对象分别和“中介”对象引用或者依赖。  
  
　　当然并不是所有的对象都需要加入“中介”对象。如果对象之间的关系原本一目了然，中介对象的加入便是“画蛇添足”。  
  
　　来看下中介者模式的组成部分吧。  
  
　　1) 抽象中介者（Mediator）角色：抽象中介者角色定义统一的接口用于各同事角色之间的通信。  
  
　　2) 具体中介者（Concrete Mediator）角色：具体中介者角色通过协调各同事角色实现协作行为。为此它要知道并引用各个同事角色。  
  
　　3) 同事（Colleague）角色：每一个同事角色都知道对应的具体中介者角色，而且与其他的同事角色通信的时候，一定要通过中介者角色协作。  
  
　　来自《设计模式》一书的类图：

|  |
| --- |
| http://soft.yesky.com/imagelist/05/10/quk664w154gr.jpg |

　　由于中介者的行为与要使用的数据与具体业务紧密相关，抽象中介者角色提供一个能方便很多对象使用的接口是不太现实的。所以抽象中介者角色往往是不存在的，或者只是一个标示接口。如果有幸能够提炼出真正带有行为的抽象中介者角色，我想同事角色对具体中介者角色的选择也是策略的一种应用。  
  
　　“恰到好处，过犹不及”。适合自己系统的便是最好的。  
  
　　**三、进一步讨论**  
  
　　是否还记得应用广泛的MVC分为哪三层？模型层（Model）、表现层（View）还有控制层（Control\Mediator）。控制层便是位于表现层与模型层之间的中介者。笼统地说MVC也算是中介者模式在框架设计中的一个应用。  
  
　　由于中介者模式在定义上比较松散，在结构上和观察者模式、命令模式十分相像；而应用目的又与结构模式“门面模式”有些相似。  
  
　　在结构上，中介者模式与观察者模式、命令模式都添加了中间对象——只是中介者去掉了后两者在行为上的方向。因此中介者的应用可以仿照后两者的例子去写。但是观察者模式、命令模式中的观察者、命令都是被客户所知的，具体哪个观察者、命令的应用都是由客户来指定的；而大多中介者角色对于客户程序却是透明的。当然造成这种区别的原因是由于它们要达到的目的不同。   
  
　　从目的上看，中介者模式与观察者模式、命令模式便没有了任何关系，倒是与前面讲过的门面模式有些相似。  
  
　　但是门面模式是介于客户程序与子系统之间的，而中介者模式是介于子系统与子系统之间的。这也注定了它们有很大的区别：门面模式是将原有的复杂逻辑提取到一个统一的接口，简化客户对逻辑的使用。它是被客户所感知的，而原有的复杂逻辑则被隐藏了起来。而中介者模式的加入并没有改变客户原有的使用习惯，它是隐藏在原有逻辑后面的，使得代码逻辑更加清晰可用。  
  
　　前面已经陆陆续续的将中介者模式的特点写了出来。这里再总结一下。使用中介者模式最大的好处就是将同事角色解耦。这带来了一系列的系统结构改善：提高了原有系统的可读性、简化原有系统的通信协议——将原有的多对多变为一对多、提高了代码的可复用性……  
  
　　但是中介者角色集中了太多的责任，所有有关的同事对象都要由它来控制。这不由得让我想起了简单工厂模式，但是由于中介者模式的特殊性——与业务逻辑密切相关，不能采用类似工厂方法模式的解决方法。因此建议在使用中介者模式的时候注意控制中介者角色的大小。  
  
　　讨论了这么多关于中介者模式的特点。可以总结出中介者模式的使用时机：一组对象以定义良好但是复杂的方式进行通信，产生了混乱的依赖关系，也导致对象难以复用。  
  
　　**四、总结**  
  
　　中介者模式很容易在系统中应用，也很容易在系统中误用。当系统出现了“多对多”交互复杂的对象群，不要急于使用中介者模式，而要先反思你的系统在设计上是不是合理。