**设计模式之Mediator(中介者)**

[**板桥里人**](http://www.jdon.com/aboutme.htm) http://www.jdon.com 2002/05/05

Mediator中介者模式定义:  
用一个中介对象来封装一系列关于对象交互行为.

为何使用Mediator?  
各个对象之间的交互操作非常多;每个对象的行为操作都依赖彼此对方,修改一个对象的行为,同时会涉及到修改很多其他对象的行为,如果使用Mediator模式,可以使各个对象间的耦合松散,只需关心和 Mediator的关系,使多对多的关系变成了一对多的关系,可以降低系统的复杂性,提高可修改扩展性.

如何使用?

首先 有一个接口,用来定义成员对象之间的交互联系方式:

|  |
| --- |
| public interface Mediator { } |

Meiator具体实现,真正实现交互操作的内容:

|  |
| --- |
| public class ConcreteMediator implements Mediator {  　　 //假设当前有两个成员. 　　 private ConcreteColleague1 colleague1 = new ConcreteColleague1();  　　 private ConcreteColleague2 colleague2 = new ConcreteColleague2();  　　 ...  } |

再看看另外一个参与者:成员,因为是交互行为,都需要双方提供一些共同接口,这种要求在Visitor Observer等模式中都是相同的.

|  |
| --- |
| public class Colleague { 　　 private Mediator mediator; 　　 public Mediator getMediator() {  　　 　　 return mediator; 　　 }  　　 public void setMediator( Mediator mediator ) {  　　 　　 this.mediator = mediator;  　　 } }  public class ConcreteColleague1 { }  public class ConcreteColleague2 { } |

每个成员都必须知道Mediator,并且和 Mediator联系,而不是和其他成员联系.

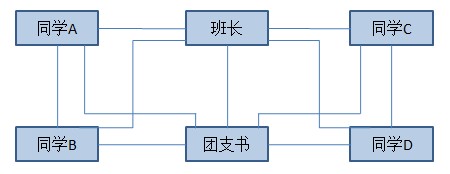
至此,Mediator模式框架完成,可以发现Mediator模式规定不是很多,大体框架也比较简单,但实际使用起来就非常灵活.

Mediator模式在事件驱动类应用中比较多,例如界面设计GUI.;聊天,消息传递等,在聊天应用中,需要有一个MessageMediator,专门负责request/reponse之间任务的调节.

MVC是J2EE的一个基本模式,View Controller是一种Mediator,它是Jsp和服务器上应用程序间的Mediator.

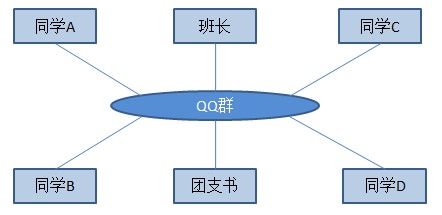
**一、初识中介者模式**

那些年，我们一起上过的大学，班级里有班长，有团书记。想一想如果没有QQ这种通讯工具的话，那么班长或者团支书该怎样下达消息呢？？同时，班级上两个同学之间也可惜沟通啊，沟通一下，院里哪个女生，哪个帅哥呀~~~如果没有QQ的话，大概就是下面的情景：



哎呀呀，看看这个乱那。如果同学的数目多起来就会变成网状的结构啦。原本把一个系统分割成一些对象是可以增强复用性的，但是现在的情况是，这些兑现之间存在着大量的联系，耦合性极高。这是很不利于复用的，同时这种情况使得系统的灵活性大大的降低，使得对系统的扩展很难，要是新转来一个学生的话，要改动的地方就多了去了。

如果现在可以使用QQ，那么可以采用另一种方式设计这个系统呢，比如做成星形的结构：

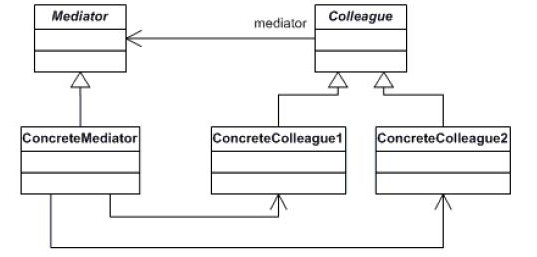


看看这种“星形结构”和“网状结构”的区别吧，显然采用星形结构就可以避免上面的网状结构存在的问题了，实际上这里的QQ就是指的是中介，这样一来每个学生对象就不用存在耦合了，同学之间需要交流可以通过一个QQ群。

本人认为，由原来的网状结构到星形结构的转变是理解中介者模式比较好的途径，下面来具体看看中介者模式，GoF说：

**中介者模式（Mediator Pattern）：**定义一个中介对象来封装系列对象之间的交互。中介者使各个对象不需要显示地相互引用，从而使其耦合性松散，而且可以独立地改变他们之间的交互。

看看结构图的形象描述吧：



对于设计模式这个东西，理解了，应该写一个代码体会一下：看看这些类，比如：Mediator，ConcreteMediator，Colleague和ConcreteColleage1到底是怎么组织的。

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #include <iostream>

2 #include <vector>

3 #include <string>

4

5 using namespace std;

6

7

8 //抽象的同事类

9 class Colleage

10 {

11 private:

12 string name;

13 string content;

14 public:

15 Colleage(string n = " "):name(n){};

16 void set\_name(string name)

17 {

18 this->name = name;

19 }

20 string get\_name()

21 {

22 return this->name;

23 }

24 void set\_content(string content)

25 {

26 this->content = content;

27 }

28 string get\_content()

29 {

30 return content;

31 }

32 virtual void talk(){};

33

34 };

35 //具体的同事类:班长

36 class Monitor : public Colleage

37 {

38 public:

39 Monitor(string n = ""):Colleage(n){};

40 virtual void talk()

41 {

42 cout<<"大班长 说："<<get\_content()<<endl;

43 }

44 };

45 //具体的同事类:团支书

46 class TuanZhiShu : public Colleage

47 {

48 public:

49 TuanZhiShu(string n = ""):Colleage(n){};

50 virtual void talk()

51 {

52 cout<<"团支书 说："<<get\_content()<<endl;

53 }

54 };

55 //具体的同事类:同学A

56 class StudentA : public Colleage

57 {

58 public:

59 StudentA(string n = ""):Colleage(n){};

60 virtual void talk()

61 {

62 cout<<"学生 A 说："<<get\_content()<<endl;

63 }

64 };

65 //具体的同事类:同学B

66 class StudentB : public Colleage

67 {

68 public:

69 StudentB(string n = ""):Colleage(n){};

70 virtual void talk()

71 {

72 cout<<"学生 B 说："<<get\_content()<<endl;

73 }

74 };

75 //抽象中介者

76 class Mediator

77 {

78 public:

79 vector<Colleage\*> studentList;

80 virtual void add\_student(Colleage \*student)

81 {

82 studentList.push\_back(student);

83 };

84 virtual void notify(Colleage \*student){};

85 virtual void chart(Colleage \*student1,Colleage \*student2){};

86 };

87 //具体中介者qq通讯平台

88 class QQMediator : public Mediator

89 {

90 public:

91 virtual void notify(Colleage \*student)

92 {

93 student->talk();

94 for(int i = 0 ; i < studentList.size() ; ++i)

95 {

96 //不是说话者

97 if(student != studentList[i])

98 {

99 studentList[i]->talk();

100 }

101 }

102 };

103 virtual void chart(Colleage \*student1,Colleage \*student2)

104 {

105 student1->talk();

106 student2->talk();

107 }

108 };

109

110

111 int main()

112 {

113 QQMediator qq;

114 Monitor studentMonitor("Vincent");

115 TuanZhiShu studentTuanZhiShu("Robort");

116 StudentA studentA("Sam");

117 StudentB studentB("Tom");

118 /\*----------------------班长发通知-----------------------------\*/

119 cout<<"下面的班长发布一个通知的场景："<<endl;

120 //将同学们加入到qq群中

121 qq.add\_student(&studentMonitor);

122 qq.add\_student(&studentTuanZhiShu);

123 qq.add\_student(&studentA);

124 qq.add\_student(&studentB);

125 //设置大家的回复信息

126 studentMonitor.set\_content("明天下午2点开年级会，收到回复^^。");

127 studentTuanZhiShu.set\_content("知道了，肯定到!!");

128 studentA.set\_content("收到了，但是可能晚点到!!");

129 studentB.set\_content("收到了，但是明天要去面试!!");

130 //开始发通知

131 qq.notify(&studentMonitor);

132 /\*--------------------两个同学私下交流--------------------------\*/

133 cout<<endl<<"下面是两个同学的私下交流："<<endl;

134 studentMonitor.set\_content("你觉得咱们“软件项目管理老师”讲的怎么样？");

135 studentA.set\_content("我觉得讲的不够生动，还点名，不太好!!!");

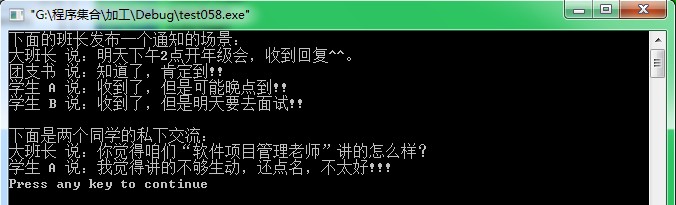
136 qq.chart(&studentMonitor,&studentA);

137 return 0;

138 }

[复制代码](javascript:void(0);)

下面是运行的结果：



看看，这样就利用qq这个平台完成了同学之间的交流。并且同学之间并没有相互联系，联系都是通过qq实现的，如果现在要对象进行扩展，比如也可以通过飞信啊，只需要加入**class FeiXin:public Mediator**就可以了，如果想扩展同学C只需要，**class StudentC:public Colleage**即可。并且在客户端的使用也是很方便的哦~~这时回过头想一想如果不用“中介者模式”的话，系统会是什么样呢？？？

**二、使用中介者模式的场合和优缺点**

**使用终结者模式的场合**

1.一组定义良好的对象，现在要进行复杂的通信。

2.定制一个分布在多个类中的行为，而又不想生成太多的子类。

可以看出，中介对象主要是用来封装行为的，行为的参与者就是那些对象，但是通过中介者，这些对象不用相互知道。呵呵~~~

**使用中介者模式的优点：**

1.降低了系统对象之间的耦合性，使得对象易于独立的被复用。

2.提高系统的灵活性，使得系统易于扩展和维护。

**使用中介者模式的缺点：**

中介者模式的缺点是显而易见的，因为这个“中介“承担了较多的责任，所以一旦这个中介对象出现了问题，那么整个系统就会受到重大的影响。