



深圳大学
Shenzhen University

第14-3讲

中介者模式

软件体系结构与设计模式
Software Architecture & Design Pattern

深圳大学计算机与软件学院

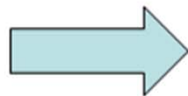
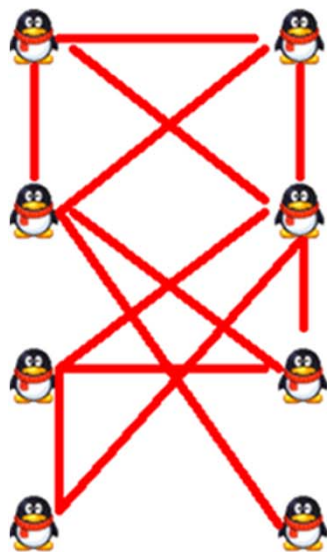


主要内容

- ◆ 中介者模式动机与定义
- ◆ 中介者模式结构与分析
- ◆ 中介者模式实例与解析
- ◆ 中介者模式效果与应用

中介者模式动机

■ QQ群示意图





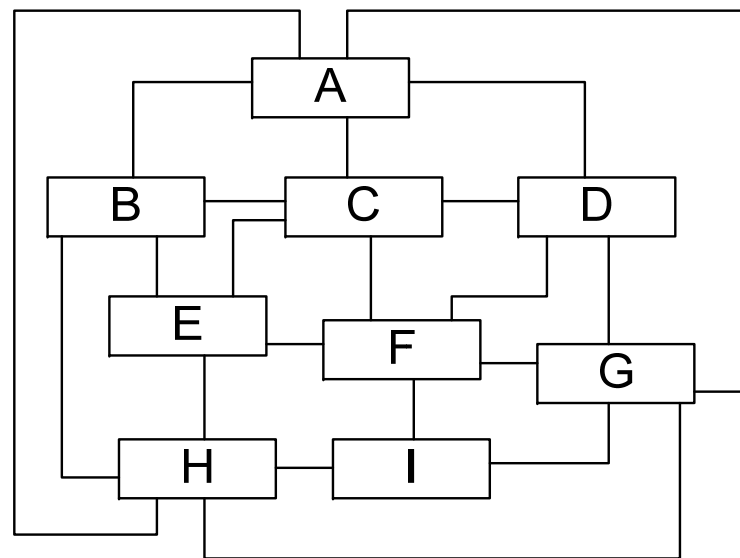
中介者模式动机

- QQ聊天的两种方式：

- (1) 用户与用户直接聊天，用户与用户之间存在多对多的联系，这将导致系统中用户之间的关系非常复杂，一个用户如果要将相同的信息或文件发送给其他所有用户，必须一个一个地发送
- (2) 通过QQ群聊天，用户只需要将信息或文件发送到群中或上传为群共享文件即可，群的作用就是将发送者所发送的信息和文件转发给每一个接收者，将极大地减少系统中用户之间的两两通信

中介者模式动机

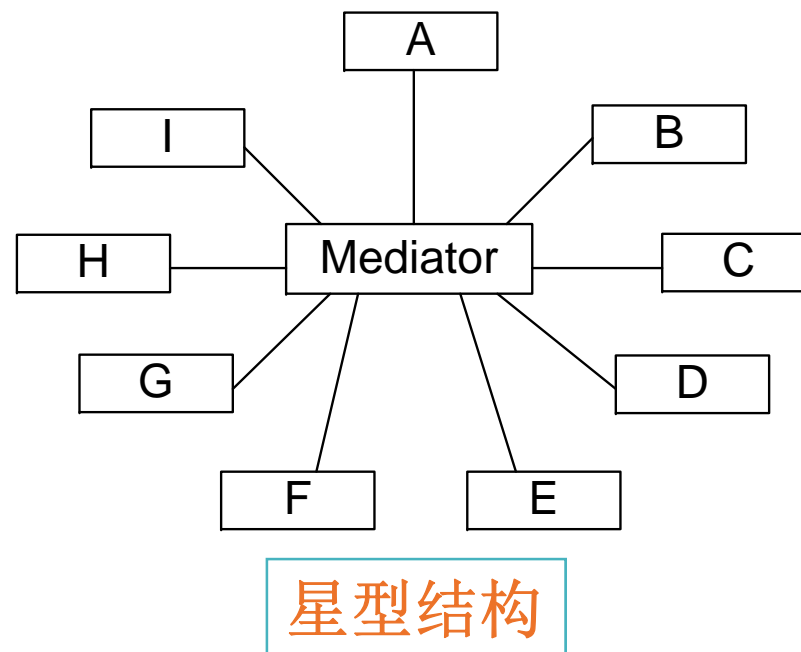
- 软件开发：
- 网状结构：多对多联系将导致系统非常复杂，几乎每个对象都需要与其他对象发生相互作用，而这种相互作用表现为一个对象与另外一个对象的直接耦合，这将导致一个过度耦合的系统



网状结构

中介者模式动机

- 软件开发：
- 星型结构：中介者模式将系统的网状结构变成以中介者为中心的星型结构，同事对象不再直接与另一个对象联系，它通过中介者对象与另一个对象发生相互作用。系统的结构不会因为新对象的引入带来大量的修改工作



中介者模式定义

■ 对象行为型模式

中介者模式：定义一个对象来封装一系列对象的交互。中介者模式使各对象之间不需要显式地相互引用，从而使其耦合松散，而且让你可以独立地改变它们之间的交互。

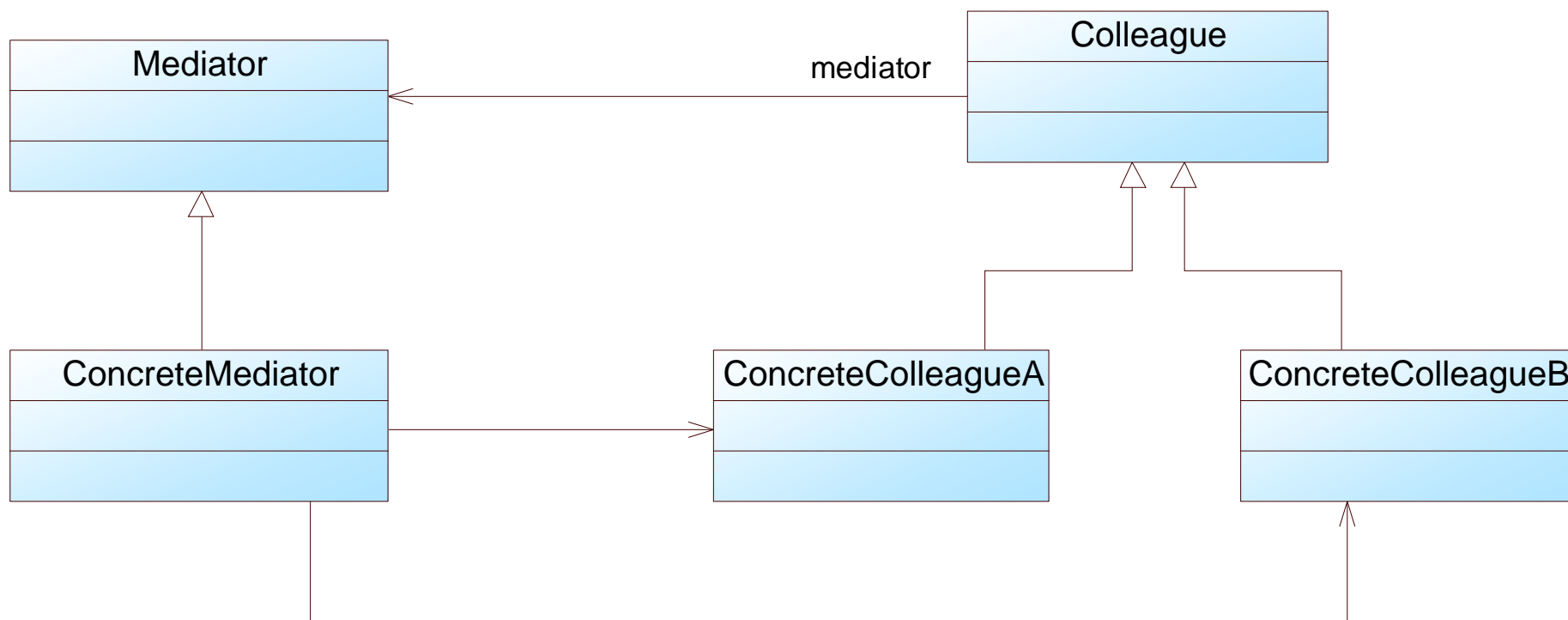
Mediator Pattern: Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

- 又称为调停者模式
- 在中介者模式中，通过引入中介者来简化对象之间的复杂交互
- 中介者模式是迪米特法则的一个典型应用
- 对象之间多对多的复杂关系转化为相对简单的一对多关系



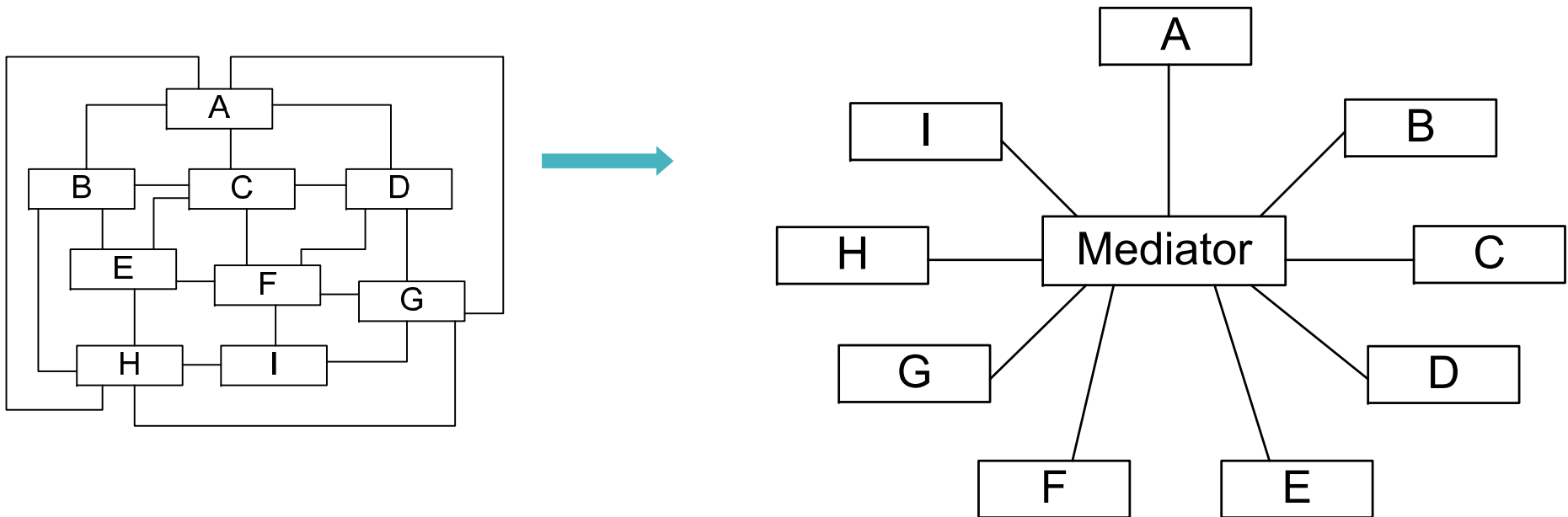
中介者模式结构

- 中介者模式包含如下角色：
 - Mediator: 抽象中介者
 - ConcreteMediator: 具体中介者
 - Colleague: 抽象同事类
 - ConcreteColleague: 具体同事类



中介者模式分析

- 中介者模式可以使对象之间的关系数量急剧减少：





中介者模式分析

■ 中介者类的职责

- 中转作用（结构性）：各个同事对象不再需要显式地引用其他同事，当需要和其他同事进行通信时，可通过中介者来实现间接调用
- 协调作用（行为性）：中介者可以更进一步的对同事之间的关系进行封装，同事可以一致地和中介者进行交互，而不需要指明中介者需要具体怎么做，中介者根据封装在自身内部的协调逻辑对同事的请求进行进一步处理，将同事成员之间的关系行为进行分离和封装



中介者模式分析

■ 抽象中介者类示例代码：

```
public abstract class Mediator {  
    protected ArrayList<Colleague> colleagues = new ArrayList<Colleague>();  
    //用于存储同事对象  
  
    //注册方法，用于增加同事对象  
    public void register(Colleague colleague) {  
        colleagues.add(colleague);  
    }  
  
    //声明抽象的业务方法  
    public abstract void operation();  
}
```



中介者模式分析

■ 具体中介者类示例代码：

```
public class ConcreteMediator extends Mediator {  
    //实现业务方法，封装同事之间的调用  
    public void operation() {  
        .....  
        ((Colleague)(colleagues.get(0))).method1(); //通过中介者调用同事  
        类的方法  
        .....  
    }  
}
```



中介者模式分析

■ 抽象同事类示例代码：

```
public abstract class Colleague {  
    protected Mediator mediator; //维持一个抽象中介者的引用  
  
    public Colleague(Mediator mediator) {  
        this.mediator=mediator;  
    }  
  
    public abstract void method1(); //声明自身方法， 处理自己的行为  
  
    //定义依赖方法， 与中介者进行通信  
    public void method2() {  
        mediator.operation();  
    }  
}
```



中介者模式分析

■ 具体同事类示例代码：

```
public class ConcreteColleague extends Colleague {  
    public ConcreteColleague(Mediator mediator) {  
        super(mediator);  
    }  
  
    //实现自身方法  
    public void method1() {  
        .....  
    }  
}
```



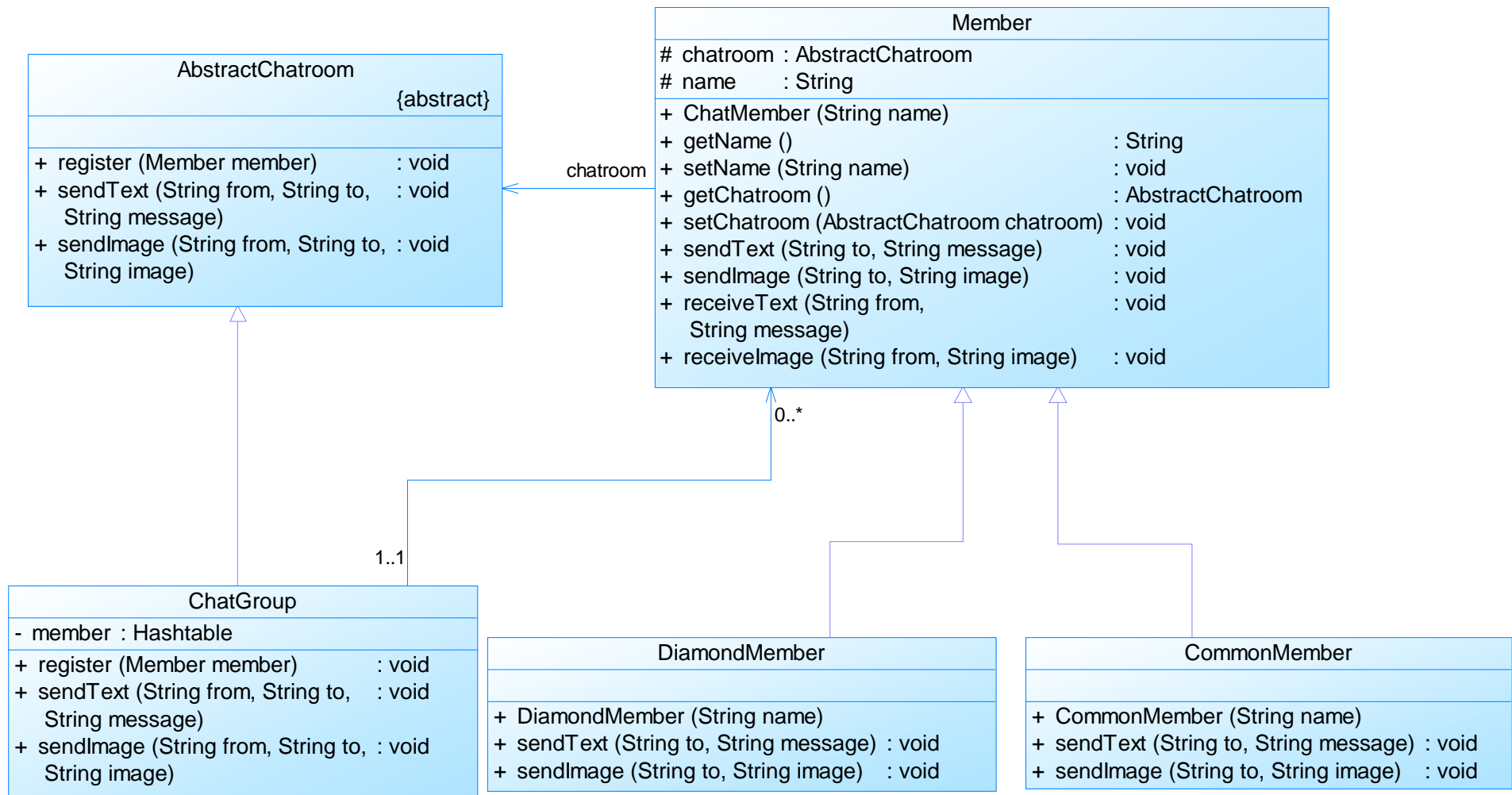
中介者模式实例

■ 虚拟聊天室：实例说明

- 某论坛系统欲增加一个虚拟聊天室，允许论坛会员通过该聊天室进行信息交流，普通会员(CommonMember)可以给其他会员发送文本信息，钻石会员(DiamondMember)既可以给其他会员发送文本信息，还可以发送图片信息。该聊天室可以对不雅字符进行过滤，如“日”等字符；还可以对发送的图片大小进行控制。用中介者模式设计该虚拟聊天室。

中介者模式实例与解析

■ 虚拟聊天室：参考类图



中介者模式实例与解析

■ 中介者模式实例

□ 虚拟聊天室：参考代码

■ DesignPatterns之mediator包

```
AbstractChatroom.java ✕
1 package mediator;
2
3 public abstract class AbstractChatroom
4 {
5     public abstract void register(Member member);
6     public abstract void sendText(String from,String to,String message);
7     public abstract void sendImage(String from,String to,String message);
8 }
```

ChatGroup.java

```
3 import java.util.*;
4 public class ChatGroup extends AbstractChatroom
5 {
6     private Hashtable members=new Hashtable();
7
8     public void register(Member member)
9     {
10         if(!members.contains(member))
11         {
12             members.put(member.getName(),member);
13             member.setChatroom(this);
14         }
15     }
16
17     public void sendText(String from,String to,String message)
18     {
19         Member member=(Member)members.get(to);
20         String newMessage=message;
21         newMessage=message.replaceAll("▣","*");
22         member.receiveText(from,newMessage);
23     }
```

ChatGroup.java

```
24
25- public void sendImage(String from,String to,String image)
26 {
27     Member member=(Member)members.get(to);
28     //模拟图片大小判断
29     if(image.length()>5)
30     {
31         System.out.println("图片太大，发送失败！");
32     }
33     else
34     {
35         member.receiveImage(from,image);
36     }
37 }
38 }
```

```
3 public abstract class Member
4 {
5     protected AbstractChatroom chatroom;
6     protected String name;
7
8     public Member(String name)
9     {
10         this.name=name;
11     }
12
13     public String getName()
14     {
15         return name;
16     }
17
18     public void setName(String name)
19     {
20         this.name=name;
21     }
22
23     public AbstractChatroom getChatroom()
24     {
25         return chatroom;
26     }
```

Member.java

```
27
28 public void setChatroom(AbstractChatroom chatroom)
29 {
30     this.chatroom=chatroom;
31 }
32
33 public abstract void sendText(String to,String message);
34 public abstract void sendImage(String to,String image);
35
36 public void receiveText(String from,String message)
37 {
38     System.out.println(from + "发送文本给" + this.name + ", 内容为: " + message);
39 }
40
41 public void receiveImage(String from,String image)
42 {
43     System.out.println(from + "发送图片给" + this.name + ", 内容为: " + image);
44 }
45 }
```

CommonMember.java

```
1 package mediator;
2
3 public class CommonMember extends Member
4 {
5     public CommonMember(String name)
6     {
7         super(name);
8     }
9
10    public void sendText(String to, String message)
11    {
12        System.out.println("普通会员发送信息: ");
13        chatroom.sendText(name, to, message); //发送
14    }
15
16    public void sendImage(String to, String image)
17    {
18        System.out.println("普通会员不能发送图片!");
19    }
20 }
```

DiamondMember.java

```
1 package mediator;
2
3 public class DiamondMember extends Member
4 {
5     public DiamondMember(String name)
6     {
7         super(name);
8     }
9
10    public void sendText(String to,String message)
11    {
12        System.out.println("钻石会员发送信息: ");
13        chatroom.sendText(name,to,message); //发送
14    }
15
16    public void sendImage(String to,String image)
17    {
18        System.out.println("钻石会员发送图片: ");
19        chatroom.sendImage(name,to,image); //发送
20    }
21 }
```



中介者模式效果与应用

■ 中介者模式优点：

- 简化了对象之间的交互，它用中介者和同事的一对多交互代替了原来同事之间的多对多交互，将原本难以理解的网状结构转换成相对简单的星型结构
- 可将各同事对象解耦
- 可以减少子类生成，中介者模式将原本分布于多个对象间的行为集中在一起，改变这些行为只需生成新的中介者子类即可，这使得各个同事类可被重用，无须直接对同事类进行扩展



中介者模式效果与应用

■ 中介者模式缺点：

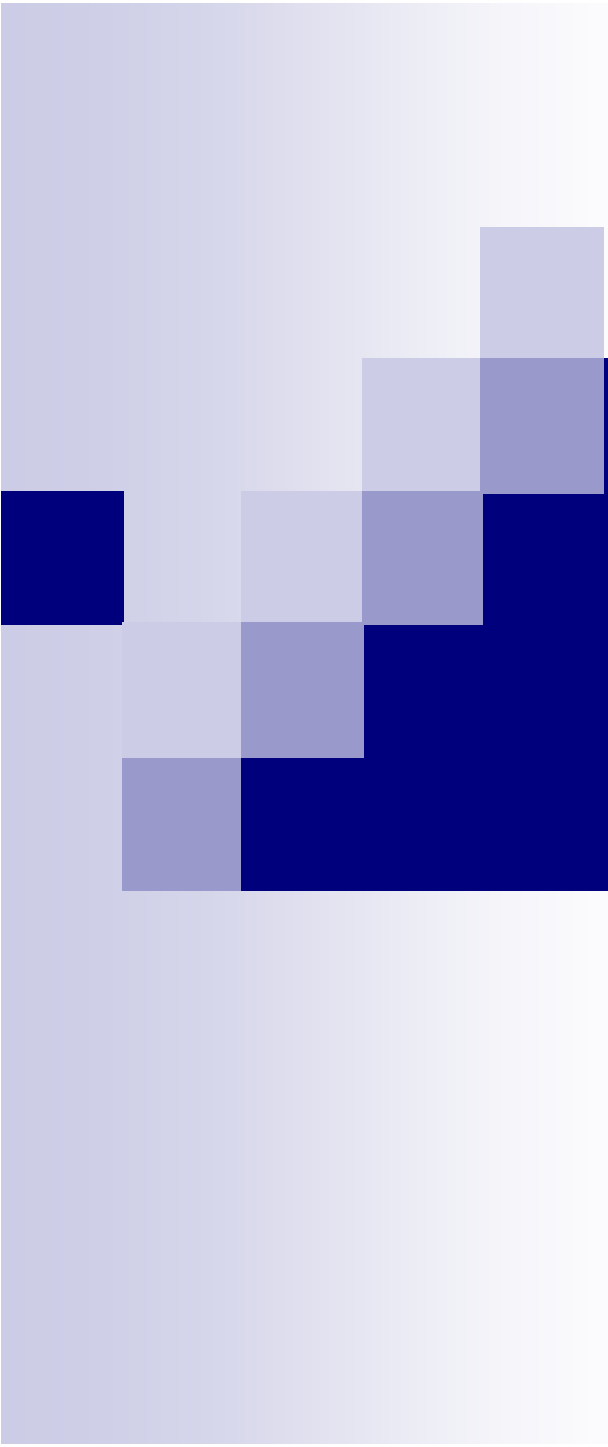
- 在具体中介者类中包含了大量的同事之间的交互细节，可能会导致具体中介者类非常复杂，使得系统难以维护



中介者模式效果与应用

■ 在以下情况下可以使用中介者模式：

- 系统中对象之间存在复杂的引用关系，系统结构混乱且难以理解
- 一个对象由于引用了其他很多对象并且直接和这些对象通信，导致难以复用该对象
- 想通过一个中间类来封装多个类中的行为，又不想生成太多的子类



谢谢