课程目标

1、掌握中介者模式的应用场景。

2、了解设计群聊的底层逻辑。

3、理解中介者模式的优缺点。

内容定位

适合参与软件框架设计开发的人群。

中介者模式

中介者模式（Mediator Pattern）又称为调解者模式或调停者模式。用一个中介对象封装一系列的对象交互，中介者使各对象不需要显示地相互作用，从而使其耦合松散，而且可以独立地改变它们之间的交互。属于行为型模式。

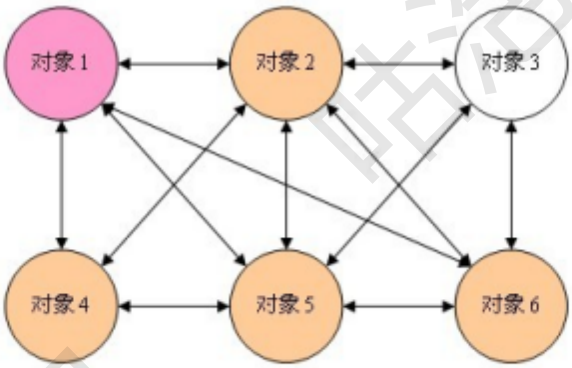
```
原文：Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.
```

中介者模式包装了一系列对象相互作用的方式，使得这些对象不必相互明显作用。从而使它们可以松散耦合。当某些对象之间的作用发生改变时，不会立即影响其他的一些对象之间的作用。保证这些作用可以彼此独立的变化。其核心思想是，通过中介者解耦系统各层次对象的直接耦合，层次对象的对外依赖通信统统交由中介者转发。

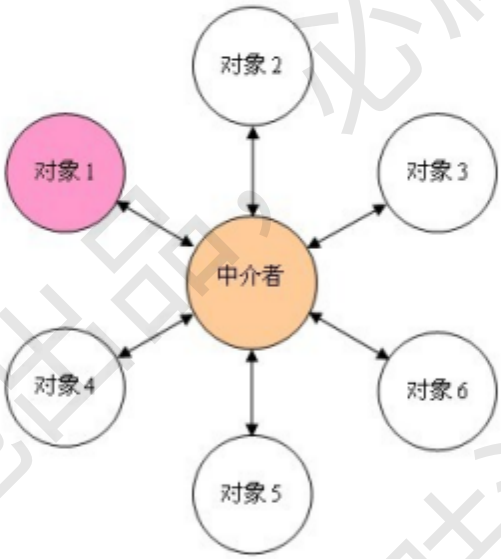
中介者模式的应用场景

在现实生活中，中介者的存在是不可缺少的，如果没有了中介者，我们就不能与远方的朋友进行交流了。各个同事对象将会相互进行引用，如果每个对象都与多个对象进行交互时，将会

形成如下图所示的网状结构。

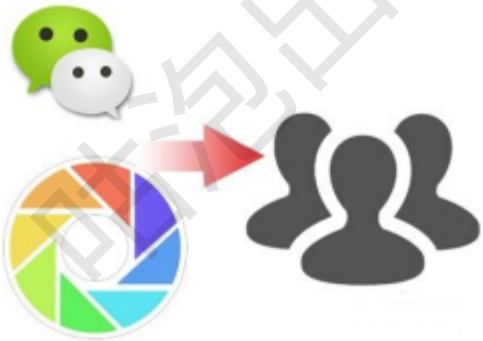


从上图可以发现，每个对象之间过度耦合，这样的既不利于信息的复用也不利于扩展。如果引入了中介者模式，那么对象之间的关系将变成星型结构，采用中介者模式之后会形成如下图所示的结构：



从上图可以发现，使用中介者模式之后，任何一个类的变化，只会影响中介者和类本身，不像之前的设计，任何一个类的变化都会引起其关联所有类的变化。这样的设计大大减少了系统的耦合度。

其实我们日常生活中每天在刷的朋友圈，就是一个中介者。还有我们所见的信息交易平台，也是中介者模式的体现。



朋友圈

数据整合中心

中介者模式是用来降低多个对象和类之间的通信复杂性。这种模式通过提供一个中介类，将系统各层次对象间的多对多关系变成一对多关系，中介者对象可以将复杂的网状结构变成以调停者为中心的星形结构，达到降低系统的复杂性，提高可扩展性的作用。

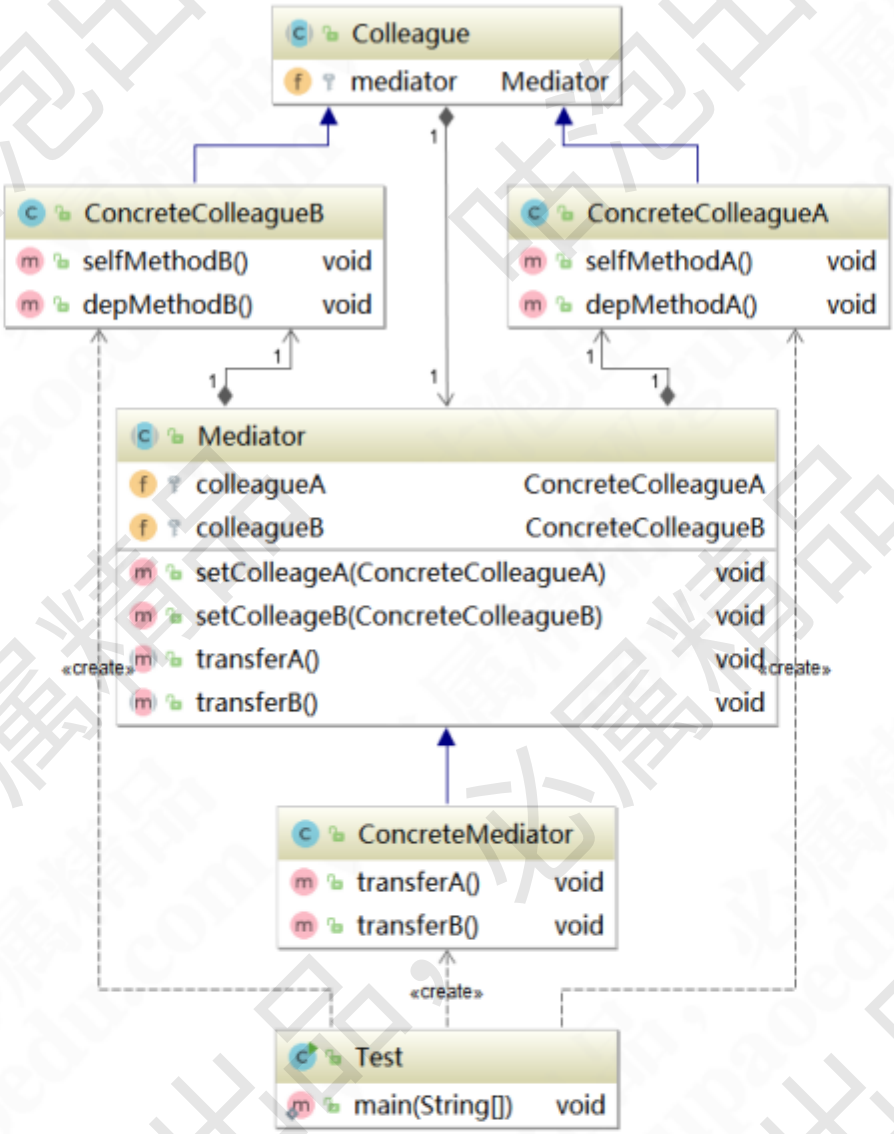
若系统各层次对象之间存在大量的关联关系，即层次对象呈复杂的网状结构，如果直接让它们紧耦合通信，会造成系统结构变得异常复杂，且其中某个层次对象发生改变，则与其紧耦合的相应层次对象也需进行修改，系统很难进行维护。而通过为该系统增加一个中介者层次对象，让其他各层次需对外通信的行为统统交由中介者进行转发，系统呈现以中介者为中心进行通讯的星形结构，系统的复杂性大大降低。

简单的说就是多个类相互耦合，形成了网状结构，则考虑使用中介者模式进行优化。总结中介者模式适用于以下场景：

1、系统中对象之间存在复杂的引用关系，产生的相互依赖关系结构混乱且难以理解；

2、交互的公共行为，如果需要改变行为则可以增加新的中介者类。

首先来看下中介者模式的通用 UML 类图：



从 UML 类图中，我们可以看到，中介者模式主要包含 4 个角色：

抽象中介者（Mediator）：定义统一的接口，用于各同事角色之间的通信；

具体中介者（ConcreteMediator）：从具体的同事对象接收消息，向具体同事对象发出命令，协调各同事间的协作；

抽象同事类（Colleague）：每一个同事对象均需要依赖中介者角色，与其他同事间通信时，交由中介者进行转发协作；

具体同事类（ConcreteColleague）：负责实现自发行为（Self-Method），转发依赖方法（Dep-Method）交由中介者进行协调。

使用中介者模式设计群聊场景

假设我们要构建一个聊天室系统，用户可以向聊天室发送消息，聊天室会向所有的用户显示消息。实际上就是用户发信息与聊天室显示的通信过程，不过用户无法直接将信息发给聊天室，而是需要将信息先发到服务器上，然后服务器再将该消息发给聊天室进行显示。具体代码如下。

创建 User 类：

```
public class User {
    private String name;
    private ChatRoom chatRoom;

    public User(String name, ChatRoom chatRoom) {
        this.name = name;
        this.chatRoom = chatRoom;
    }

    public void sendMessage(String msg) {
        this.chatRoom.showMsg(this, msg);
    }

    public String getName() {
        return name;
    }
}
```

创建 ChatRoom 类：

```
public class ChatRoom {
    public void showMsg(User user, String msg) {
        System.out.println("[" + user.getName() + "] :" + msg);
    }
}
```

编写客户端测试代码：

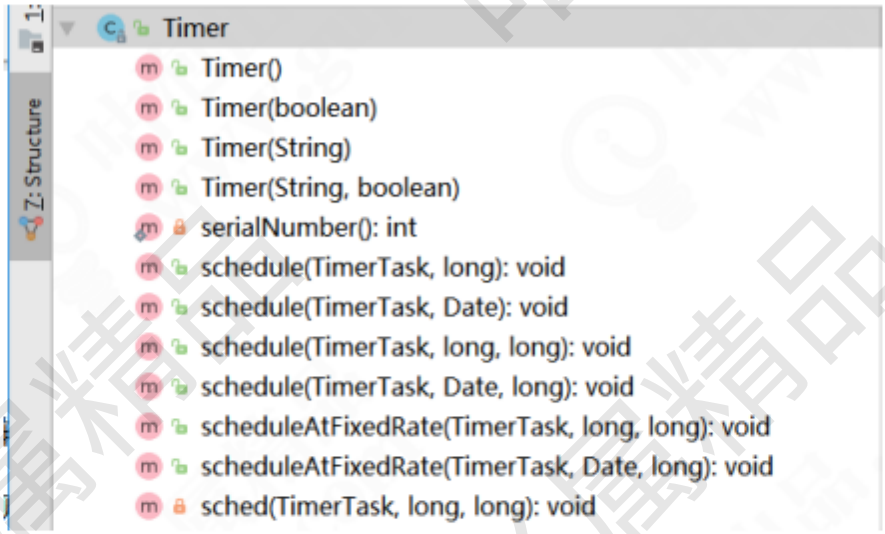
```
    public static void main(String[] args) {
        ChatRoom room = new ChatRoom();

        User tom = new User("Tom",room);
        User jerry = new User("Jerry",room);
        tom.sendMessage("Hi! I am Tom.");
        jerry.sendMessage("Hello! My name is Jerry.");
    }
```

运行结果如下：

中介者模式在源码中的体现

首先来看 JDK 中的 Timer 类，打开 Timer 的结构我们发现 Timer 类中有很多的 schedule()重载方法，如下图：



我们任意点开其中一个方法，发现所有方法最终都是调用了私有的 sched()方法，我们来看一下它的源码：

```
public class Timer {
    ...
  public void schedule(TimerTask task, long delay) {
      if (delay < 0)
          throw new IllegalArgumentException("Negative delay.");
      sched(task, System.currentTimeMillis()+delay, 0);
  }
    ...
    private void sched(TimerTask task, long time, long period) {
      if (time < 0)
          throw new IllegalArgumentException("Illegal execution time.");

      if (Math.abs(period) > (Long.MAX_VALUE >> 1))
          period >>= 1;

      synchronized(queue) {
          if (!thread.newTasksMayBeScheduled)
              throw new IllegalStateException("Timer already cancelled.");

          synchronized(task.lock) {
              if (task.state != TimerTask.VIRGIN)
                  throw new IllegalStateException(
                      "Task already scheduled or cancelled");
              task.nextExecutionTime = time;
              task.period = period;
              task.state = TimerTask.SCHEDULED;
          }

          queue.add(task);
          if (queue.getMin() == task)
```

```
                queue.notify();
            }
        }
        ...
}
```

而且，我们发现，不管是什么样的任务都被加入到一个队列中顺序执行。我们把这个队列中的所有对象称之为“同事”。同事之间通信都是通过 Timer 来协调完成的，Timer 就承担了中介者的角色。

中介者模式的优缺点

优点：

1、减少类间依赖，将多对多依赖转化成了一对多，降低了类间耦合；

2、类间各司其职，符合迪米特法则。

缺点：

中介者模式中将原本多个对象直接的相互依赖变成了中介者和多个同事类的依赖关系。当同事类越多时，中介者就会越臃肿，变得复杂且难以维护。