# 中介者模式简介

对于那些对象复杂交互的系统，中介者模式提供了一种简化复杂交互的解决方案，它通过引入中介者将原本对象之间的两两交互转化为每一个对象与中介者之间的交互，中介者可以对对象之间的通信进行控制与协调，降低原有系统的耦合度，使得系统更加灵活，也更易于扩展。

## 概述

1. 在QQ聊天中存在两种聊天方式：第一种是用户与用户直接聊天，第二种是通过QQ群聊天。一个用户如果要与其他用户聊天或发送文件，通常需要加其他用户为好友，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用户，必须一个一个地发送，于是QQ群产生了，如果使用QQ群，一个用户就可以像多个用户发送相同的信息和文件而无须一一进行发送，只需要将信息或文件发送到群众或作为群共享即可，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用户。通过引入群的机制，将极大减少系统中用户之间的两两通信，用户与用户之间的联系可以通过群来实现。在软件系统总，某些类/对象之间的相互调用关系错综复杂，类似QQ用户之间的关系，此时需要一个类似“QQ群”的中间类来协调这些类/对象之间的复杂关系，以降低系统的耦合度。中介者模式为此而诞生，它通过在系统中增加中介者对象来降低原有类/对象之间的复杂引用关系。
2. 如果一个系统中对象之间的联系呈现为网状结构，如下图所示：



对象之间存在大量的多对多联系，将导致系统非常复杂，这些对象既会影响其他对象，也会被其他对象所影响，这些对象被称为同事对象，它们之间通过彼此的相互作用实现系统的行为。在网站结构中，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，这将导致一个过渡的耦合的系统。中介者模式可以使对象之间的关系急剧减少，通过引入中介者对象，可以将系统的网状结构编程以中介者为中心的星形结构。如下图所示：



在这个星型结构中，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。中介者对象的存在保证了对象结构上的稳定，也就是说，系统的结构不会因为新对象的引入带来大量的修改工作。

1. 如果一个系统中对象之间的存在多堆垛的相互关系，可以将对象之间的一些交互行为从各个对象分离处理，集中封装在一个中介者对象中，并由该中介者进行统一协调，这样对象之间的多对多关系就转化为相对简单的一对多关系。
2. 中介者模式（Mediator Pattern）的定义如下：定义一个对象来封装一系列对象的交互，中介者模式使各个对象之间不需要显式地相互引用，从而使其耦合松散，而且用户可以独立地改变他们之间的交互。
3. 中介者模式又称为调停者模式，它是一种对象行为型模式。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。

## 结构

在中介者模式中引入了用于协调其他对象/类之间相互调用的中介者类，为了让系统具有更好的灵活性和可扩展性，通常还提供了抽象中介者。其结构图如下：



由结构图可知，中介者模式包含以下4个角色：

1. Mdeiator（抽象中介者）：它定义一个接口，该接口用于与同事对象之间进行通信。
2. ConcreteMediator（具体中介者）：它是抽象中介者的子类，通过协调各个同事对象来实现协作行为。它维持了各个同事对象的引用。
3. Colleague（抽象同事类）：它定义各个具体同事类公有的方法，并声明了一些抽象方法供子类实现，同时它维持了一个抽象中介者的引用，其子类可以通过该引用与具体中介者进行通信。
4. ConcreteColleague（具体同事类）：它是持续同时类的子类，每一个同时对象在需要和其他同时对象通信时需要先与中介者通信，通过中介者间接完成与其他同时类的通信；在具体同事类中实现了在抽象同事类中声明的抽象方法。

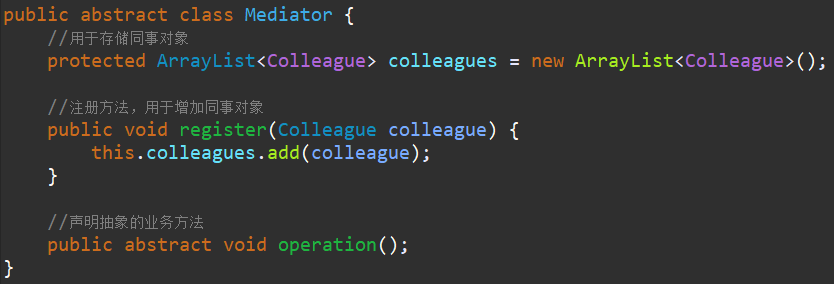
# 实现

## 实现原理

中介者的核心在于中介者类的引入，在中介者模式中，中介者类承担了两个方面的职责：

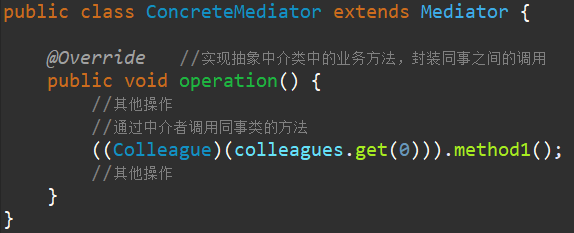
1. 中转作用（结构型）：通过中介者提供的中转作用，各个同事对象不再需要显式地引用其他同事，当需要和其他同事进行通信时可通过中介者实现间接调用。该中转作用属于中介者在结构上的支持。
2. 协调作用（行为性）：中介者模式可以更进一步地对同事之间的关系进行封装，同事可以一致地和中介者进行交互，而不需要指明中介者需要具体怎么做，中介者根据封装在自身内部的协调逻辑对同事的请求进行进一步处理，将同事成员之间的关系行为进行分离和封装。该协调作用属于中介者在行为上的支持。

1、在中介者模式中，典型的抽象中介者代码如下：



2、在抽象中介者可以定义一个同事类的集合，用于存储同事对象并提供注册方法，同时声明了具体中介者所具有的方法。在具体中介者类中将实现这些抽象方法，典型的具

体中介者类代码如下：

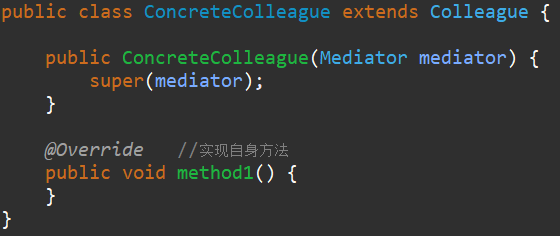


在具体中介者类中将调用同事类的方法，在调用时可以增加一些自己的业务代码对调用进行控制。

3、在抽象同事类中维持了一个抽象中介者的引用，用于调用抽象中介者的方法。典型的抽象同事类代码如下：



4、在抽象同事类中声明了同事类的抽象方法，而在具体同事类中将实现这些方法，典型的具体同事类代码如下：



在具体同时类ConcreteColleague中实现了在抽象同事中声明的方法，其中方法method1( )是同事类的自身方法（Self-Method），用于处理自己的行为；而方法method2( )是依赖方法（Depend-Method），用于调用在中介者定义的方法，依赖中介者来完成相应的行为，例如调用另一个同事类的相关方法。

## 实例

某软件公司要开发一套CRM系统，其中包含一个客户信息管理模块，所设计的“客户信息管理窗口”界面效果如下图所示：



通过分析发现，界面组件之间存在较为复杂的交互关系；如果删除一个客户，则将从客户列表（List）中删掉对应的项，客户选择组合框（ComboBox）中的客户名称也将减少

一个；如果增加一个客户信息，则客户列表中将增加一个客户，并且组合框中也将增加一项。为了更好地处理界面组件之间的交互，现使用中介者模式设计该系统。为了协调界面组件对象之间的复杂交互关系，可引入一个中介者类，其结构如下：



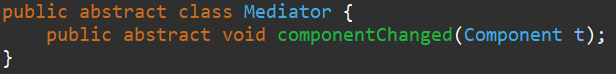
以上只是一个结构示意图，在具体实现时为了确保系统具有更好的灵活性和扩展性，需要定义抽象中介者和抽象组件类，其中抽象组件类是所有具体组件类的公共父类，完整类图如下：



其中，Component充当抽象同事类，Button、List、ComboBox和TextBox充当具体同事类，Mediator充当抽象中介者类，ConcreteMediator充当具体中介者类。ConcreteMediator维持了对具体同事类的引用，为了简化ConcreteMediator类的代码，在其中只定义了一

个Button对象和一个TextBox对象。

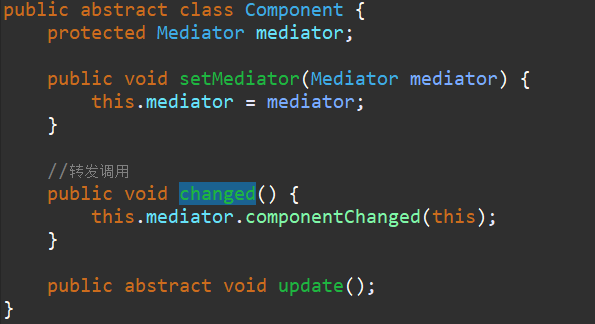
1. Mediator：抽象中介者类



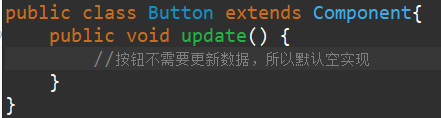
1. ConcreteMediator：具体中介者类



1. Component：抽象组件类，充当抽象同事类



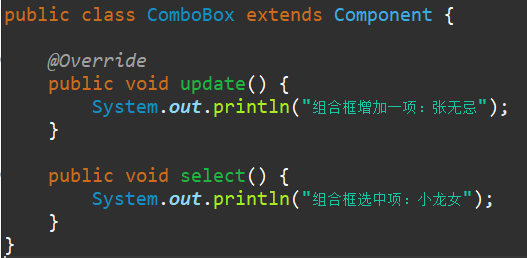
1. Button：按钮类，充当具体同事类



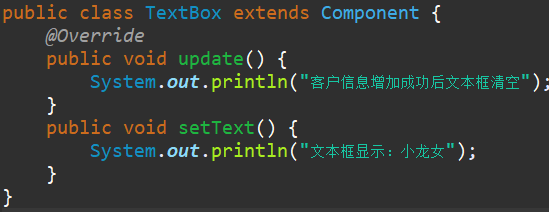
1. List：列表框类，充当具体同事类



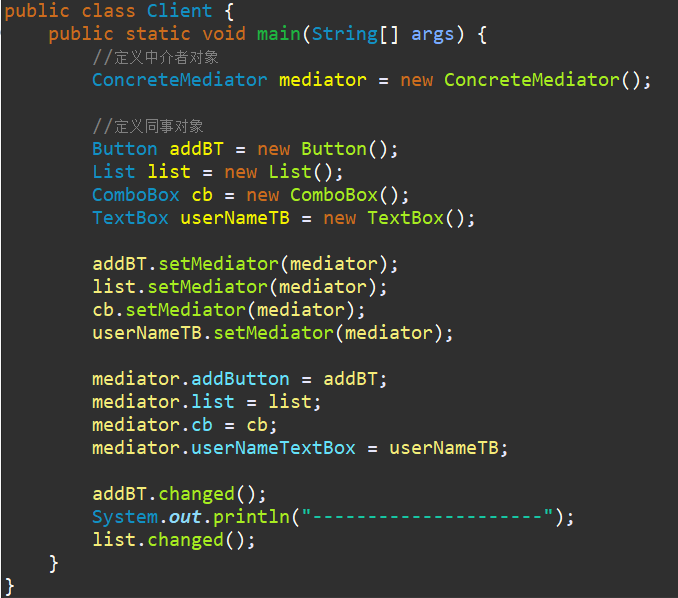
1. ComboBox：组合框类，充当具体同事类



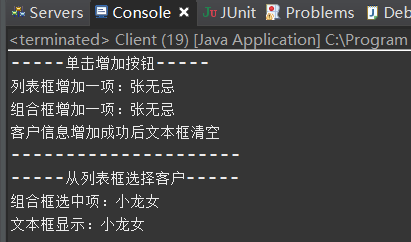
1. TextBox：文本框类，充当具体同时类



1. Client：客户端测试类



运行结果如下：



## 实例扩展

对上一节中的实例进行改进，使窗口的下端能够及时显示当前系统中客户信息的数量。如下图所示：

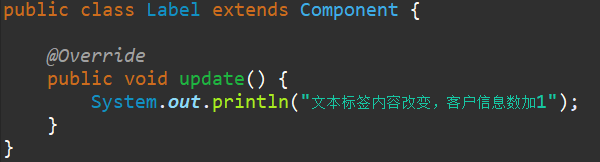


从图中不难发现，可以通过增加一个文本标签（Label）来显示客户信息总数，而且当用户单击“增加”按钮或者“删除”按钮时将改变文本标签的内容。由于使用了中介者模式，在原有系统中增加新的组件（即新的同事类）将变得很容易，至少有两种解决方案：

方案1：增加一个界面组件类Label，修改原有的具体中介者类ConcreteMediator，增加一个对Label对象的引用，然后修改componentChanged( )方法中其他组件对象的业务处理代码，原有组件类无须任何修改，客户端代码也需要针对新增组件进行适当修改。

方案2：与方案1相同，首先增加一个Label类，但不修改原有具体中介者类ConcreteMediator的代码，而是增加一个ConcreteMediator的子类SubConcreteMediator来实现对Label对象的引用，然后在新增的中介者类SU币ConcreteMedia投入中通过覆写componentChanged( )方法实现所有组件（包括新增的Label组件）之间的交互。同样，原有组件类无须做任何修改，客户端代码做少许修改。

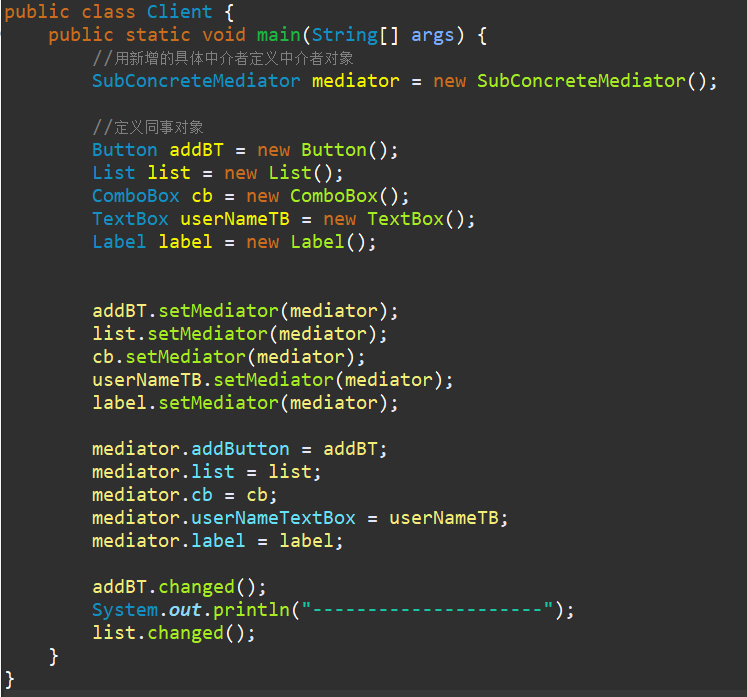
由于方案2无须修改ConcreteMediator类，更符合开闭原则，因此选择方案2对新增Label类进行处理。新增Label类代码如下：

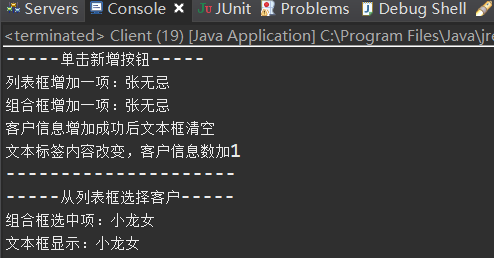


新增的中介者类SubConcreteMediator类代码如下：



修改先前的客户端代码如下：



 测试结果如下：

由于本实例中不同的组件类（即不同的同事类）所拥有的方法并不完全相同，因此中介者类没有针对抽象同事类编程，导致在具体中介者类中需要维持对具体同事类引用，客户端无法完全透明地对待所有同事类和中介者类。在某些情况下，如果设计得当，可以在客户端透明地对同事类和中介者类编程，这样系统将具有更好的灵活性和可扩展性。

在中介者模式的实际使用过程中，如果需要引入新的具体同事类，只需要继承抽象同事类并实现其中的方法即可，由于具体同事类之间并无直接的引用关系，因此原有所有同事类无须进行任何修改，它们与新增同事对象之间的交互可以通过修改或者增加具体中介者类来实现；如果需要在原有系统中增加新的具体中介者类，只需要继承抽象中介者类（或已有的具体中介者类并覆写其中定义的方法即可），在新的具体中介者中可以通过不同的方式处理对象之间的交互，也可以增加对新增同事的引用和调用。在客户端中只需要修改少许代码（如果引入配置文件可以不用修改任何代码）就可以实现中介者类的更换。

# 优缺点及适用环境

中介者模式将一个网状的系统结构变成一个以中介者对象为中心的星型结构，在这个星型结构中使用中介者对象与其他对象的一对多关系来取代原有对象之间的多对多关系。中介者模式在事件驱动类软件中的应用较为广泛，特别是基于GUI（Graphical User Interface，图形用户界面）的应用软件。此外，类与类之间存在错综复杂的关联关系的系统中，中介者模式也得到了较好的应用。

## 优点

中介者模式的优点主要如下：

1. 中介者模式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，一对多关系更容易理解、维护和扩展，将原有难以理解的网站结构转换成相对简单的星型结构。
2. 可将各同时对象解耦。中介者有利于各同事之间的松耦合，可以独立地改变和复用每一个同事和中介者，增加新的中介者类和新的同事类都比较方便，更好地符合开闭原则。
3. 可以减少子类生成，中介者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以被重用，无须直接对同事类进行扩展。

## 缺点

中介者模式的缺点主要如下：在具体中介者类中包含了大量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。

## 适用环境

在以下情况下可以考虑使用中介者模式：

1. 系统中对象之间存在复杂的引用关系，造成系统结构混乱且难以理解。
2. 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。
3. 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，此时可以通过引入中介者类来实现，在中介者类定义对象交互的公共行为，如果需要改变行为则可以修改已定义的交互行为或增加新的具体中介者类。