**lab06\_TCP协议网络编程实验项目**

学生姓名：苏家铭 合作学生：无

实验地点：济世楼330 实验时间：2023年9月27日

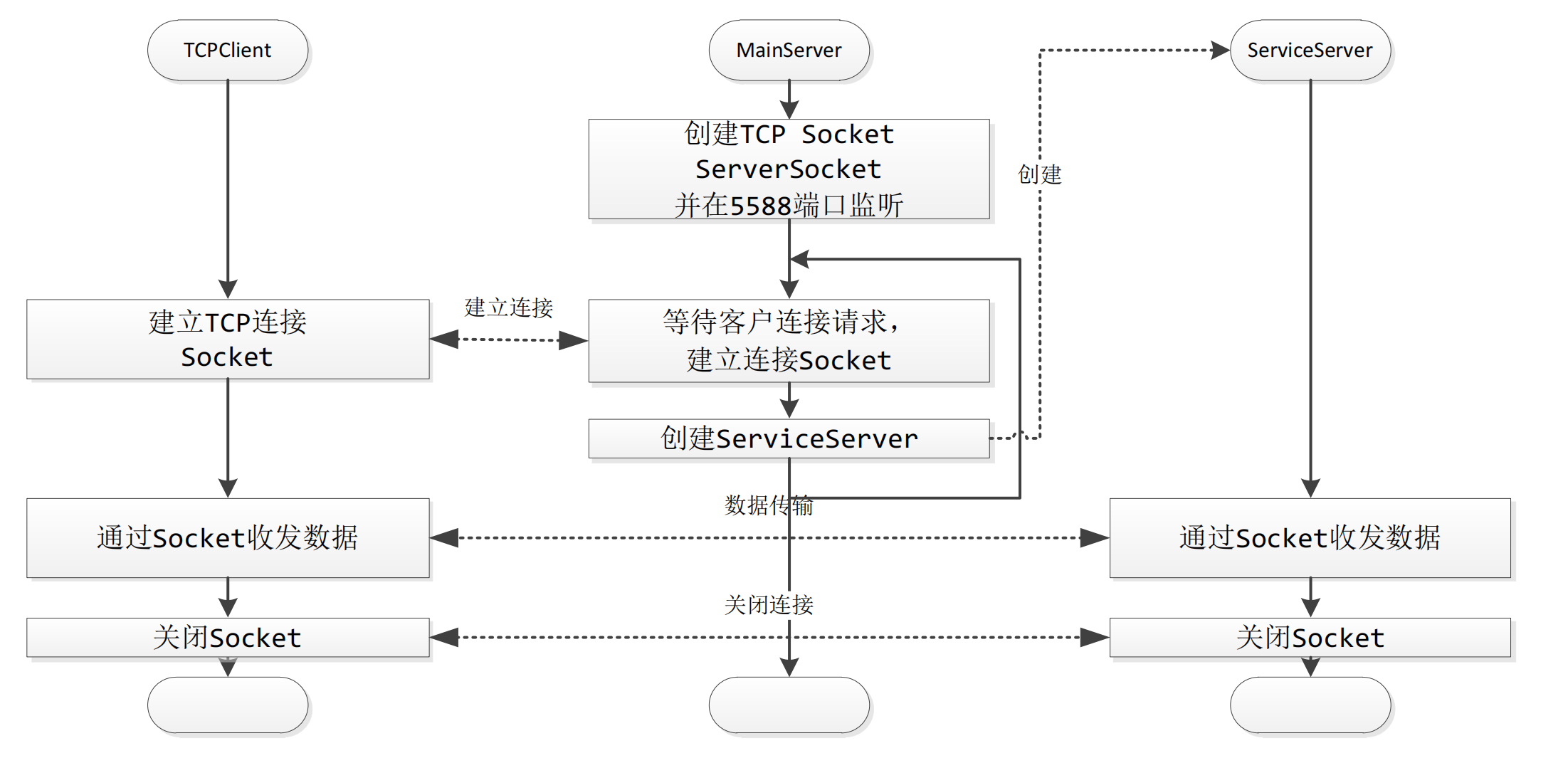
**【实验目的】**

1. 了解基于TCP网络应用服务器的基本编程架构。
2. 了解面向连接和无连接的区别，了解TCP编程基本步骤。
3. 了解并发服务原理及编程方式

**【实验原理】**

TCP采用面向连接，服务器一旦同一个客户机建立了连接，就无法同其他客户再建立连接，必须提供多线程或进程的并发服务。并发服务能力是指服务器具有同时建立多个连接，允许多个客户访问的能力，日常使用的TCP服务器均是并发服务器。

创建两个网络程序进行简单通信，基本功能是客户机从发送窗口发送任意一段字符串给服务器，服务器接收来自客户机的字符串，然后将字符串返回给客户机，客户机在另一个显示窗口显示返回消息。



UDP 采用无连接服务，可同时接待多个客户访问。TCP 采用面向连接方式，服务器一旦同一个客户机建立了会话，就无法同其他客户再建立会话，严重影响用户体验。并发服务能力是指服务器具有同时建立多个会话，允许多个客

户访问的能力，日常使用的 TCP 服务器均是并发服务器。但并发服务不属于网络技术而属于编程技术研究范畴，并发服务器程序有着特殊的编程架构，典型架构由两部分代码组成，第一部分是主进程，负责请求调度，接受客户

请求申请，建立会话并为每个请求创建服务处理线程；第二部分是服务处理线程，负责具体会话处理，为客户请求提供服务，可以被主进程不断创建。图 5-21 表示动态服务器创建过程。假定有两个客户访问同一个 TCP 服务器。

TCP 服务器启动后，只有主进程在运行，开启服务端口 PORT，等待客户端访问。第一个客户访问 PORT 端口，主进程接收到请求 1，创建一个服务线程 1负责处理请求 1，随后的会话发生在客户机 1 和服务线程 1 之间，主进程等待下一次请求到来；第二个客户访问 PORT 端口，主进程接收到请求 2，再创建另一个服务线程 2 负责处理请求，随后的会话发生在客户机 2 和服务线程2 之间，主进程继续等待下一个请求。如果同时有 N 个客户访问服务器，这时候，就有 N+1 个线程(进程) 在运行；主进程永远等待客户访问。并发处理的数量理论上没限制，主要取决于计算机处理能力，涉及 CPU 核数量和内存大小。

MainServer.java代码解析：

该代码创建了一个基于TCP协议的服务器，使用TCP协议在端口5588上接受客户端连接，并为每个客户端连接创建一个独立的线程以处理其请求。以下是代码的主要功能和解释：

1. `MainServer` 类是服务器的主类，它负责创建服务器套接字并等待客户端的连接。
2. `ServerSocket` 对象 `server` 被创建，并绑定到端口号5588。这表示服务器将在端口5588上监听客户端的连接请求。
3. `ServiceServer.setUserList(threadList)` 是一个可能是静态方法的调用，用于设置用户列表。`threadList` 是一个集合，用于存储连接到服务器的客户端线程。
4. 在 `while` 循环中，服务器不断接受客户端的连接请求。每当有客户端连接到服务器时，服务器会为该客户端创建一个新的 `Socket` 对象 `client`。
5. 然后，服务器创建一个 `ServiceServer` 对象 `service`，并将客户端的 `Socket` 传递给该对象。`ServiceServer` 似乎是一个处理客户端请求的服务类。
6. 接着，服务器创建一个新的线程 `thread`，并将 `service` 对象作为参数传递给线程的构造函数。然后，服务器启动这个线程，使其运行 `ServiceServer` 中的逻辑。
7. `threadList.add(service)` 将 `service` 对象添加到 `threadList` 中，以便服务器可以跟踪和管理所有连接到服务器的客户端线程。
8. 在异常处理部分，如果发生了 `IOException`，服务器会在控制台上打印异常信息。

ServiceServer.java代码解析：

这段代码是一个用于处理客户端请求的服务类`ServiceServer`。该类实现了 `Runnable` 接口，表示可以在单独的线程中运行。这段代码用于处理客户端的请求。它通过对象流接收和发送用户信息，并可以广播文本消息给所有在线用户。以下是代码的主要功能和解释：

1. `userList`：这是一个静态的集合，用于存储连接到服务器的所有用户（客户端）的 `ServiceServer` 实例。这个集合似乎是用来跟踪所有在线用户的。
2. `user`：`User` 对象，表示客户端用户的信息，包括用户ID等。
3. `client`：`Socket` 对象，表示与客户端建立的网络连接。
4. `is`：`ObjectInputStream` 对象，用于从客户端接收对象数据流。
5. `os`：`ObjectOutputStream` 对象，用于向客户端发送对象数据流。
6. `out`：`PrintWriter` 对象，用于向客户端发送文本消息。
7. 在构造函数中，初始化了 `client`、`is`、`os` 等对象，以便与客户端进行通信。
8. `run` 方法是 `Runnable` 接口的实现，表示在单独的线程中运行。在该方法中，服务器首先接收来自客户端的 `User` 对象，然后进入一个循环，等待接收来自客户端的文本消息。
9. 服务器接收到客户端发送的文本消息后，会在控制台上输出消息内容，并调用 `Response` 方法将消息发送给所有在线用户（通过 `userList`）。
10. `sendResponse` 方法用于向客户端发送消息，它将消息通过 `PrintWriter` 发送给客户端。
11. `Response` 方法用于将消息广播给所有在线用户，通过迭代 `userList` 中的每个 `ServiceServer` 实例来发送消息。
12. `setUserList` 方法用于设置 `userList` 集合，以便 `ServiceServer` 类知道哪些用户当前在线。

TcpClient.java代码解析：

这段代码是一个简单的Java图形用户界面（GUI）应用程序，用于与服务器进行TCP通信，支持文本消息的发送和接收。它通过图形用户界面提供了一个友好的用户界面，允许用户输入消息并与服务器进行交互。

以下是代码的主要功能和解释：

1. `TcpClient` 类继承了 `JFrame`，表示一个图形用户界面窗口。
2. 在 `main` 方法中，创建了两个 `TcpClient` 的实例，分别代表两个客户端用户，分别使用不同的用户名（"Jin" 和 "zhang"）。
3. `user`：`User` 对象，表示客户端用户的信息，包括用户ID等。
4. `serviceSocket`：`Socket` 对象，用于与服务器建立连接。
5. `out`：`PrintWriter` 对象，用于向服务器发送文本消息。
6. `in`：`BufferedReader` 对象，用于从服务器接收文本消息。
7. `os`：`ObjectOutputStream` 对象，用于向服务器发送对象数据流。
8. `is`：`ObjectInputStream` 对象，用于从服务器接收对象数据流。
9. 构造函数根据提供的用户ID创建了一个 `User` 对象，并初始化了界面组件。
10. `btnExitActionPerformed` 方法用于退出客户端应用程序，关闭相关的输入输出流和套接字。
11. `btnSendActionPerformed` 方法用于发送文本消息到服务器。它首先将消息发送到服务器，然后等待服务器的响应，并在收到响应后将消息显示在客户端的文本聊天窗口中。
12. `btnConnectActionPerformed` 方法用于建立与服务器的连接。它创建了一个 `Socket` 对象，通过该套接字与服务器进行通信，并初始化输入输出流。之后，客户端可以开始发送和接收消息。
13. `initComponents` 方法用于初始化GUI界面，包括聊天窗口、消息输入框、连接按钮等。

User.java代码解析：

这段代码定义了一个 `User` 类，实现了 `Serializable` 接口，用于表示用户信息。由于实现了 `Serializable` 接口，可以轻松地将 `User` 对象序列化为字节流，以便在网络上传输或保存到文件中。

以下是代码的主要功能和解释：

1. `User` 类实现了 `Serializable` 接口，表示该类的对象可以进行序列化，可以在网络上传输或保存到文件中。
2. `userID` 和 `name` 是该类的私有成员变量，分别用于存储用户的ID和名称。
3. 提供了两个构造方法：

* 无参数构造方法：用于创建默认的 `User` 对象。
* 带参数的构造方法：接受一个字符串参数 `user`，用于初始化 `userID` 属性。

1. 提供了 `getUserID` 和 `setUserID` 方法，用于获取和设置用户的ID。
2. 提供了 `getName` 和 `setName` 方法，用于获取和设置用户的名称。

**【实验设备】**

硬件：济世楼330机房电脑

IDE：Eclipse

**【实验步骤】**

在Eclipse中新建project运行代码MainServer.java,ServiceServer.java,TcpClient.java和User.java，启动客户端后在界面中输入传输的数据可以在服务器处理后的回显中看到相应的文本。

**【实验现象】**

下面是运行代码时的结果：

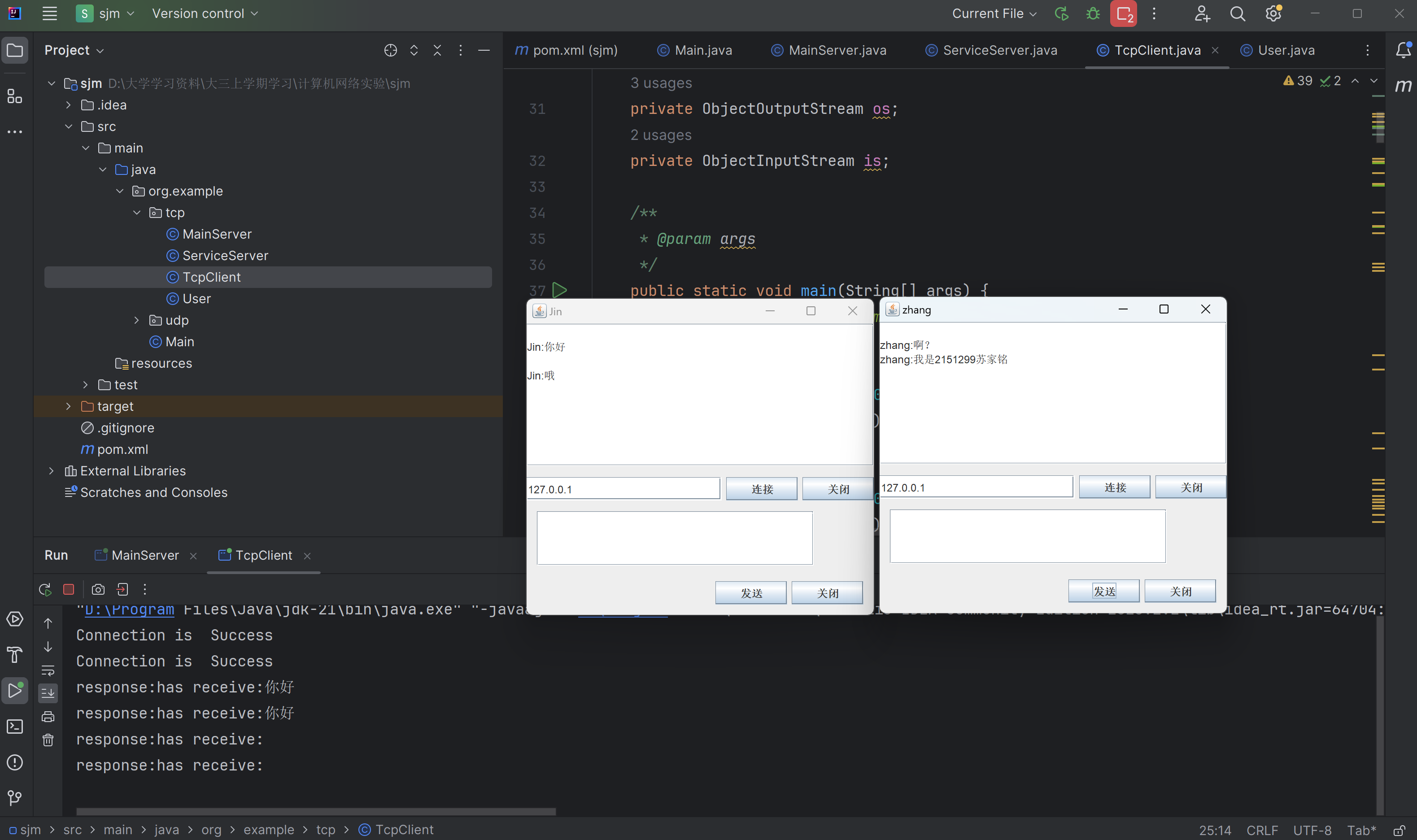
MainServer 类创建了一个TCP服务器，监听端口5588。

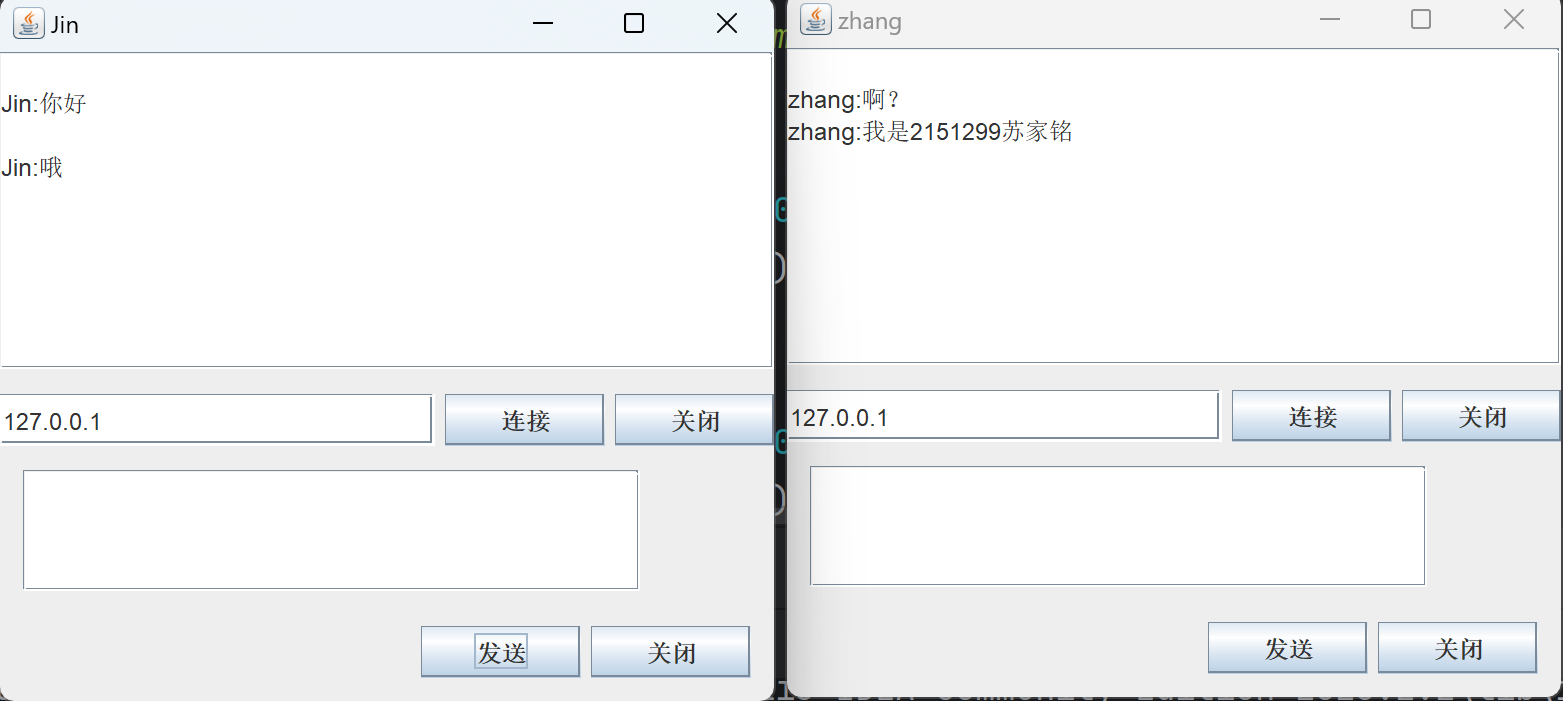
TcpClient 类创建了两个TCP客户端，每个客户端提供了一个图形用户界面，用户可以在界面中输入文本消息并发送到服务器。

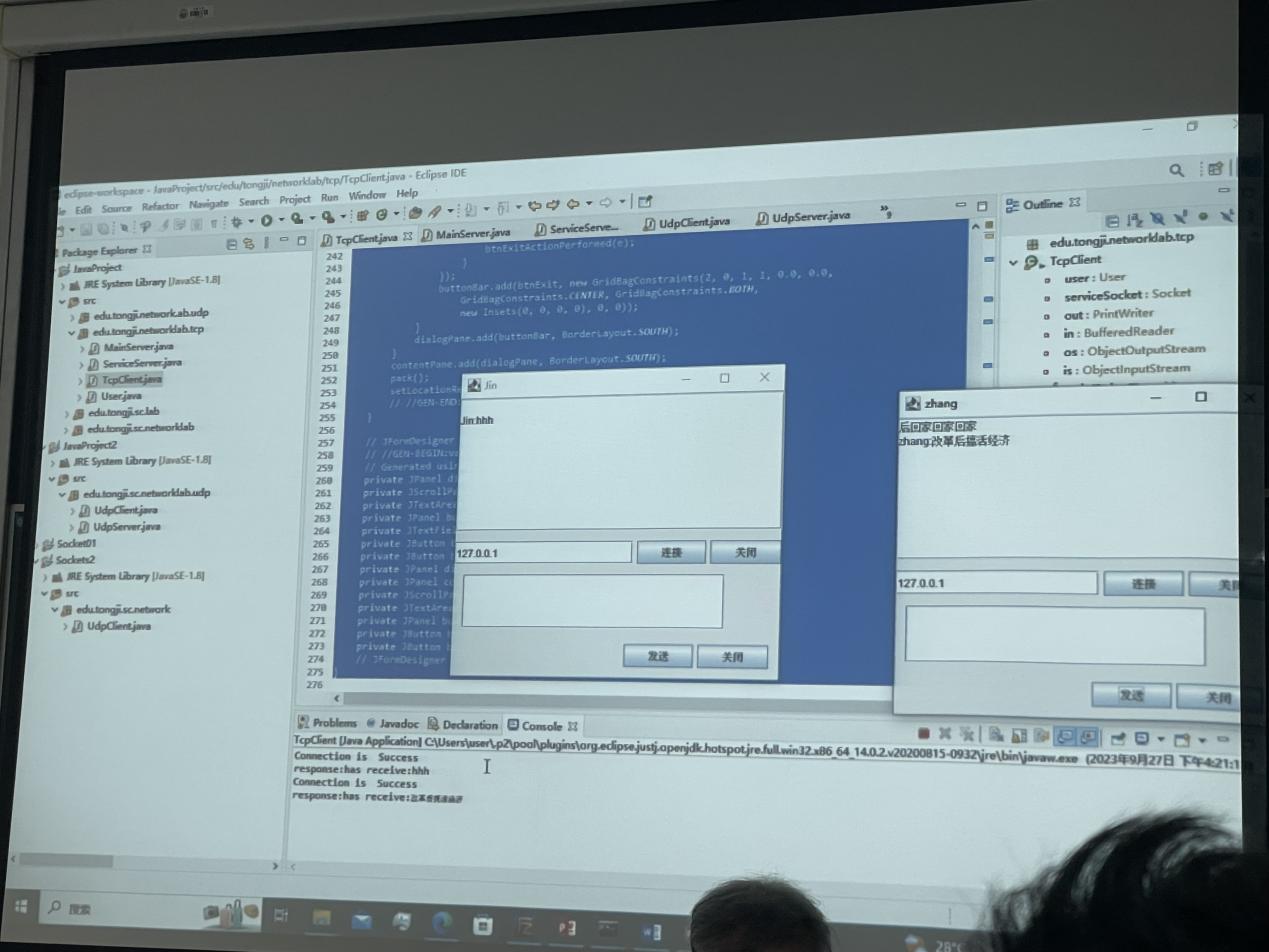
当你运行 MainServer 时，它会启动并等待在端口5588上接收TCP连接。

当你运行任何一个 TcpClient 时，它会创建一个TCP套接字，连接到服务器的IP地址和端口5588。然后，你可以在客户端的界面中输入文本消息并点击发送按钮，该消息将被发送到服务器。

服务器接收到来自客户端的消息，然后将该消息广播给所有连接到服务器的客户端，包括其他 TcpClient 实例。每个客户端都会将服务器的广播消息显示在界面的文本聊天区域中。







**【分析讨论】**

TCP和UDP协议的比较分析：

1. 可靠性与实时性： UDP适用于实时性要求高的应用，如实时音视频传输，因为它具有低延迟。然而，TCP提供了可靠性的数据传输，适用于需要稳定性和数据完整性的应用。
2. 数据包丢失： UDP允许数据包丢失，而TCP会尝试重新发送丢失的数据包。这使得UDP适用于某些场景，如在线游戏中的快速数据更新，但也可能导致数据丢失。
3. 连接和多客户端： TCP建立了连接，适用于需要多客户端连接的服务器应用。UDP不需要连接，可以广播消息给多个客户端，适用于一对多通信。
4. 性能开销： UDP的性能开销较低，适用于对延迟要求较高的应用。TCP的性能开销较高，但提供了数据可靠性。
5. 适用场景： UDP适用于实时通信，如语音通话、视频会议等。TCP适用于文件传输、Web浏览、电子邮件传输等需要可靠性的应用。