

《网络与通信》课程实验报告

实验 2: Socket 通信编程

姓名	张泽毅	院系	计算机学院	学号	21120971
任课教师	刘通	指导教师	刘通		
实验地点	计 706	实验时间	周三 7-8 节课		
实验课表现	出勤、表现得分(10)	实验报告 得分(40)	实验总分		
	操作结果得分(50)				

实验目的:

1. 掌握 Socket 编程过程;

2. 编写简单的网络应用程序。

实验内容:

利用你选择的任何一个编程语言, 分别基于 TCP 和 UDP 编写一个简单的 Client/Server 网络应用程序。具体程序要求参见《实验指导书》。

要求以附件形式给出:

● 系统概述: 运行环境、编译、使用方法、实现环境、程序文件列表等;

● 主要数据结构;

● 主要算法描述;

● 用户使用手册;

● 程序源代码;

实验要求: (学生对预习要求的回答) (10 分)

得分:

● Socket编程客户端的主要步骤

TCP:

1、创建一个Socket对象

2、指定服务器IP地址与对应端口号

3、建立服务端与客户端的链接

4、发送数据

5、接收服务器回写数据

6、关闭Socket连接

UDP:

1、创建一个DatagramSocket对象

2、指定服务器IP地址与对应端口号

3、创建一个DatagramPacket对象

4、发送数据, 接收响应

5、关闭DatagramSocket

● Socket编程服务器端的主要步骤

TCP:

1、创建ServerSocket对象

2、绑定IP地址与端口号

- 3、等待客户端连接
- 4、连接成功后获取输入输出流
- 5、与客户端通信
- 6、关闭Socket连接

UDP:

- 1、创建DatagramSocket对象
- 2、绑定IP地址与端口号
- 3、接收客户端的数据并解析
- 4、关闭连接

实验过程中遇到的问题如何解决的？（10 分）	得分：
<p>问题 1：服务器可能无法同时处理多个客户端的相关请求 这是由于服务器进程或线程是单线程的，它按顺序处理客户端请求，每次只处理一个请求，直到该请求完成为止。我采用的处理方式是：使用多线程的办法。通过多线程，服务器可以为每个客户端连接创建一个单独的线程，这些线程可以并发运行，每个线程负责处理一个客户端的请求。并在实验过程中为了使得资源优化，采用了线程池。</p> <p>问题 2：在连接别的主机或服务器时出现连接不上、连接超时的情况 这是由于客户端没有在指定时间内链接到服务端，或是服务器没能处理传入的连接请求，导致客户端的连接被拒绝。我采用的处理方式是：由于本次实现的是实验任务，所以将服务器 IP 指定为 127.0.0.1（本机地址），这样就避免了连接问题，保证了连接的成功。</p> <p>问题 3：客户端上传了一个文件之后无法上传第二个文件 这是由于一开始为了保证一个文件能够正确地继续传输，我在客户端发送完一个文件后使用了 socket 的 shutdownOutputStream()方法来关闭输出流，但这导致了即使使用了循环也无法上传下一个文件。所以我采取的处理方法是删除这一方法并采用了如下的解决办法：每次在传输文件之前，将该文件的大小发送到服务端，服务端在接收到这一大小的数据之后就停止接收。这样既能保证输入流与输出流的始终开启以及数据的准确性，更重要的是实现了客户端能够多次上传文件的功能。</p> <p>问题 4：使用 BufferedOutputStream 传输数据之后无法在服务端看到正常的文件。 产生这一现象的原因是未刷新缓冲区。因为 BufferedOutputStream 是带有缓冲区的，如果缓冲区没有被填满，数据就可能不会进入传送通道。所以我采用的处理方式是：在使用 BufferedOutputStream 时确保在完成写入操作后关闭流或调用 flush 方法，以确保数据被正确写入目标文件。</p> <p>问题 5：TCP 在用户输入文件地址前（已登录）无法实现被踢出或被退出 由于在传输文件前只读取过登录结果的回送信息，所以服务端控制台无论输入何种指令都不会在客户端被接收到。我采用的处理方法是在用户登录成功后在客户端开启一个新的线程用来监听服务端发送的消息，从而实现在进入系统后无论何时都能通过服务端控制客户端。</p>	

问题 6: TCP 无法通过用户名来调用方法直接找到对应的 Socket 并关闭连接

上述情况是因为在多用户连接的情况下,服务器需要维护一个映射关系来跟踪每个用户与其对应的 Socket。所以我采用的处理方式是:定义一个 `disconnectUser` 方法,这个方法会迭代已连接的 Socket 列表 (`connectedSockets`),并使用 `getUsernameFromSocket` 方法获取每个 Socket 对应的用户名。如果找到了匹配的用户名,它会向客户端发送一条踢出消息,然后关闭连接并从连接列表中移除这个 Socket。

问题 7: TCP 无法直接通过调用 Socket 的函数来获得用户名

上述情况是因为因为 Socket 对象本身并没有直接存储用户名的属性。所以,我采用的处理方式是:首先通过 `socket.getInetAddress().getHostAddress()` 方法可以获取到给定 Socket 对应的客户端 IP 地址。随后遍历 `connectedClients` Map: `connectedClients` Map 存储了已连接客户端的信息,其中键是用户名,值是客户端的 IP 地址。再检查每个连接的客户端是否与给定 Socket 的客户端 IP 地址匹配:通过遍历 `connectedClients` Map,对比每个键值对中的值(即客户端 IP 地址)是否与给定 Socket 的客户端 IP 地址匹配。最后返回匹配的用户名:如果找到匹配的客户端 IP 地址,则返回对应的用户名(即 Map 的键),否则返回 `null` 表示未找到匹配的用户名。这意味着在客户端连接到服务器时,需要将客户端的 IP 地址和用户名添加到 `connectedClients` 集合中。

问题 8: UDP 服务端实现用户身份验证问题

这个问题实际上与 UDP 本身的性质有关,UDP 在运行过程中没有服务端与客户端的明确的连接状态。这就可以认为,用户没有和 TCP 一样的在线状态,所以用户每上传一个文件,就相当于开启了一次新的连接。我采用的处理方式是:首先对每一个用户的用户名及密码进行验证,验证成功后开启一个线程用于该用户的操作,在该线程中使用 UDP 的传输方式,这样在用户端和逻辑上就可以视作每个用户采用 UDP 传输方式进行了多次文件的传输。

问题 8: UDP 协议下无法找到所有发送过消息的客户端

这是由于在 `connectedClients` 这个 Map 中,使用 `InetAddress` 作为键来跟踪客户端,`InetAddress` 不是唯一的,因此会导致覆盖问题。我采用的解决方式是:将 `connectedClients` 中的键改为用户名(即客户端的唯一标识),这样就可以跟踪每个客户端的消息。

问题 9: 基于 UDP 的传输系统中,show 是想展示已发送过消息的客户端,但是登陆成功(不发信息)后也会显示该用户

这是由于在处理登录成功后,将用户添加到 `connectedClients` 中,但没有考虑到用户是否发送消息的状态。所以我采取的处理方式是:在 `Server` 类中,添加一个 Map,用于存储用户是否发过消息的状态,在 `handleClientMessage` 方法中,当用户发送消息时,将该用户的状态设置为已发送消息。

本次实验的体会(结论)(10 分)

得分:

本次 Socket 实验初看实验指导书与网上的资料感觉原理并不难，但是在实操的过程中，我碰到了很多书上没有涉及到的问题，这需要在阅读的过程中不断查阅资料，不断实践，才能够了解这一实验的整个过程及原理。

本次实验耗时很长，其中包含了很多新的基础知识需要靠自己来学习运用。实验也包含了有很多的难点，比如 TCP 与 UDP 的通信、IO 流等等，其中所涉及到的传输方式、字符、缓冲区等等重要的相关知识对我来说也都是全新的，因此我在实践的过程中学到了很多。在 TCP 传输实验的过程中，我从一个用户一次传输开始，一步步修改最终实现多用户多线程版本，实现了全部的目标。在 UDP 实验的过程中，我也运用到了多线程的思想，但是我仍留有一个遗憾：在 UDP 的用户传输数据时，似乎没有最好的体现 UDP 传输数据的特点，这也会是我今后会尝试去改善的一个方向

因为在本次实验之前对 Java 语言有一定的学习了解，感觉其封装做的较好，所以本次实验使用了 Java 进行编程。通过这次实验，我不仅对 Java 编程有了更加深入的了解，也让我对计算机网络的基本通信方式和原理有了更加深入的认识。这为之后的学习奠定了基础，也让我用更加贴近实际应用的角度了解了网络通信。

思考题：（10 分）

思考题 1：（4 分）

得分：

你所用的编程语言在 Socket 通信中用到的主要类及其主要作用。

- **Socket:** Socket 类是 Java 中用于创建套接字连接的类。在客户端中,通过创建 Socket 对象,与服务器建立 TCP 连接。(getInputStream()获取输入流、getOutputStream()获取输出流、close()关闭连接)
- **Scanner:** Scanner 类用于从标准输入读取用户输入的信息,例如用户名和密码。
- **PrintWriter:** PrintWriter 类用于向服务器发送数据,它可以将数据以文本形式发送到服务器的输出流。
- **BufferedReader:** BufferedReader 类用于从服务器接收数据,它可以从服务器的输入流中读取文本数据。
- **Thread:** Thread 类用于创建多线程。在代码中创建一个用于读取服务器响应的新线程。
- **File:** File 类用于操作文件,例如获取文件名、文件长度等。
- **FileInputStream:** FileInputStream 类用于从本地文件读取数据。
- **BufferedInputStream:** BufferedInputStream 类用于提高文件输入性能,它可以从 FileInputStream 中读取数据并进行缓冲。
- **BufferedOutputStream:** BufferedOutputStream 类用于提高输出性能,它可以将数据写入到 Socket 的输出流并进行缓冲。
- **ServerSocket:** ServerSocket 类是 Java 中用于创建服务器套接字的类。通过创建 ServerSocket 对象,可以监听指定端口,等待客户端连接。(accept()等待客户端链接、close()关闭 socket 连接)
- **ThreadPoolExecutor:** ThreadPoolExecutor 是 Java 中用于创建线程池的类。在服务器端,通过创建线程池来管理并发连接,以便能够同时处理多个客户端的请求。
- **Map:** Map 是 Java 中用于存储键值对的接口。在设计的代码中,使用 userDatabase 和 connectedClients 分别存储用户的用户名和密码以及已连接的客户端信息。
- **List:** List 是 Java 中用于存储列表数据的接口。在设计的代码中,使用

connectedSockets 来存储已连接的客户端 Socket 对象。

- **Executors:** Executors 类提供了工厂方法来创建不同类型的线程池。
- **MyRunnable:** 这是一个自定义的类，用于处理客户端连接的具体逻辑。它实现了 Runnable 接口，可以在线程池中执行。
- **BufferedWriter:** BufferedWriter 类用于向客户端发送数据，它可以将数据以文本形式发送到客户端的输出流。
- **FileOutputStream:** FileOutputStream 类用于将数据写入本地文件。
- **UUID:** UUID 类用于生成唯一标识符，用于创建保存文件的唯一文件名。
- **DatagramSocket:** DatagramSocket 类是 Java 中用于实现 UDP 通信的类。在客户端中，通过创建 DatagramSocket 对象，可以发送和接收 UDP 数据包。
- **DatagramPacket:** DatagramPacket 类用于表示 UDP 数据包，包括要发送的数据和目标主机的地址和端口。它被用来封装要发送的消息和接收到的消息。
- **InetAddress:** InetAddress 类用于表示 IP 地址。在代码中，通过 InetAddress 获取目标服务器的 IP 地址。

思考题 2: (6 分)

得分:

说明 TCP 和 UDP 编程的主要差异和特点。

TCP 编程的主要特点:

1. **面向连接:** TCP 是一种面向连接的协议，在进行数据通信之前，客户端与服务端必须通过三次握手建立一个连接，而这个连接是一个可靠的双向通信通道，它能够保证数据的可靠传输。
2. **可靠性:** TCP 提供可靠的数据传输机制，它能够保证数据按照顺序到达目标。在传输的过程中不会丢失、重复、损坏。如果数据在传输的过程中发生了丢失，TCP 会将其自动重传以保证数据的正确性。
3. **流式传输:** TCP 是面向字节流的协议，它将数据视为连续的字节流，而不是分散的数据包，这使得 TCP 适用于传输大文件等数据。

UDP 编程的主要特点:

1. **面向无连接:** UDP 是一种面向无连接的协议，它不需要在通信之前建立连接。每个 UDP 数据包都是独立的，具有自己的目标地址和端口，这使得 UDP 适用于快速通信领域。
2. **由于 UDP 是面向无连接的协议，所以它不提供可靠的数据传输，数据包可以在任何顺序到达，甚至可能丢失、重复损坏。UDP 适用于那些能够容忍一些数据丢失的应用，比如视频传输等等。**
3. **报文传输:** UDP 以数据报的形式传输数据，每个数据报都有一个独立的大小和边界。这使得 UDP 适用于那些需要精确控制消息边界的应用。

TCP 和 UDP 编程的主要差异:

TCP 和 UDP 编程的主要差异在于连接性和可靠性。TCP 是面向连接的，提供可靠的数据传输，保证数据的顺序和完整，而 UDP 是面向无连接的，不提供可靠性保证。这也使得对于传输速率而言，UDP 要明显快于 TCP。

所以在 TCP 编程中，我们只需对应好服务器 IP 和端口，就能保证安全的传输数据。而 UDP 需要考虑的内容更多，尤其是关于多用户同时访问服务器时，用户的数据安全性，所以需要小心地进行系统设计。

指导教师评语：

日期：

附件：

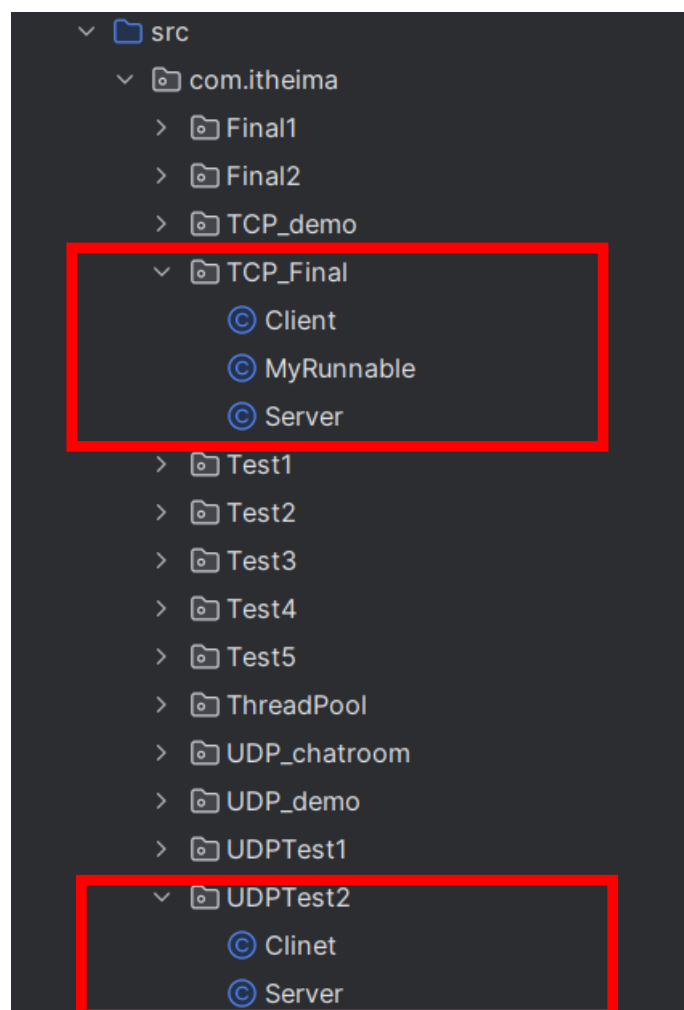
系统概述：

- 运行环境：Idea
- JDK：Oracle OpenJDK version 17.0.7
- 操作系统：Windows 11

使用方法：

先运行 Server 服务端，再运行 Clinet 客户端，如需要多个用户同时登录，运行多次 Clinet 即可。具体的操作说明请见下表。

程序文件列表：



TCP:

● 主要数据结构：

1. Map: Map 是 Java 中的数据结构，用于存储键值对。在代码中，使用了两个 Map 数据结构：
 - userDatabase: 用于存储用户的用户名和密码信息，以进行用户身份验证。
 - connectedClients: 用于存储已连接的客户端信息，其中键为用户名，值为客户端的 IP 地址。
2. List: List 是 Java 中的数据结构，用于存储有序的元素列表。在代码中，使用

了 `connectedSockets` 列表来存储已连接的客户端 `Socket` 对象。

● 主要算法描述

服务端：

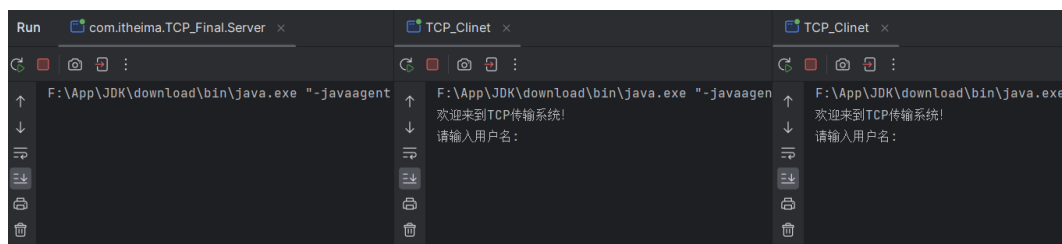
1. 初始化：
 - 启动服务器并监听指定的 10088 端口。
 - 构建用户列表。
 - 启动一个控制台监听线程，在运行过程中可以在服务台控制台输入：`show`、`kick+用户名`、`exit`，并执行各个指令对应的操作。
 - 初始化一个线程池，用于处理多个用户端请求。
2. 接收客户端连接：
 - 只要接收到客户端连接请求就 `accept`。
 - 将该连接交给线程池进行处理。
3. 处理客户端请求：
 - 接收来自客户端上传的用户名与密码，与用户列表进行对比验证。
 - 读取用户传来的文件名并用 `UUID` 库生成随机文件名创建文件输出流。
 - 如果收到来自客户端的 `quit`，就断开连接。
 - 接收文件大小（长度）
 - 接收文件内容至上述大小（接收文件是循环的）
 - 向客户端回传文件上传成功或失败的确认信息
4. 监听服务端控制台输入：
 - `Show`：显示当前连接的用户及其 `IP`。
 - `Kick+用户名`：用于踢出指定用户，与其断开连接，客户端下线。
 - `Exit`：与全部客户端断开连接，全部客户端下线，服务端关闭。

客户端：

1. 与服务器连接：
 - 与指定的服务器及 10088 端口创建连接。
2. 登录
 - 根据欢迎词及提示输入用户名与密码。
 - 将用户名与密码发送至服务端进行验证。
3. 发送与接收数据
 - 根据提示输入文件路径（或 `quit` 退出）并摁下回车。
 - 向服务器发送所要传输文件的大小。
 - 发送文件内容。
 - 接收服务器回写信息。

● 用户手册

■ 启动服务端：

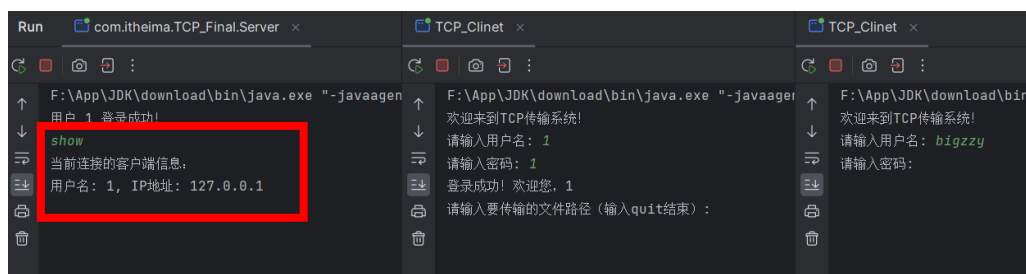


■ 用户登录：

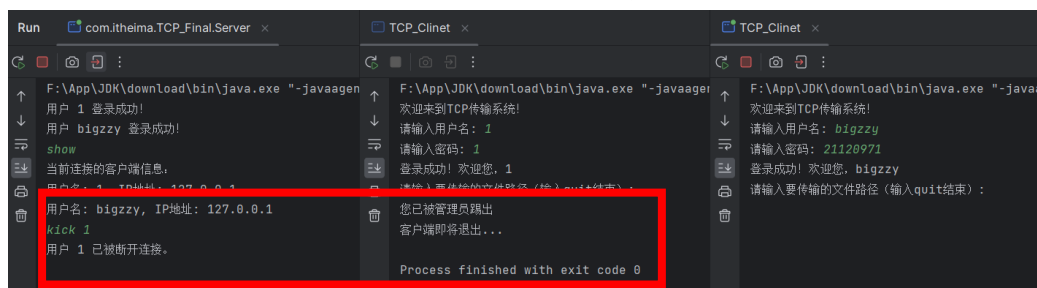


此时，2 个用户成功登陆系统，在服务器端也会实时显示用户登录情况。

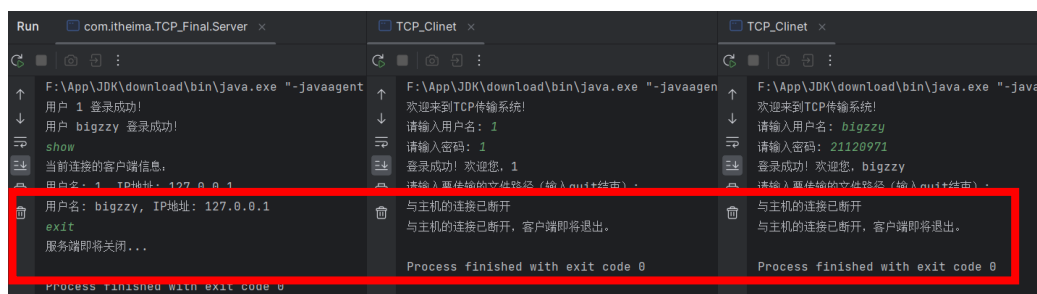
■ 服务端相关指令（show、kick、exit）：



上面两图分别展示了 2 个用户登陆成功后，通过 show 操作展示的用户信息。以及 1 个用户成功登录后（另一用户尚未登陆），通过 show 操作展示的用户信息。



使用“kick+用户名”的指令可以指定某一用户下线，图中选择用户 1 下线。用户下线时服务端与客户端都会有相应的提示信息。



使用“exit”指令可以断开所有连接并关闭服务端。

■ 客户端上传文件：

```
Run com.itheima.TCP_Final.Server x
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\IdeaC\download\IntelliJ IDEA Co
用户 1 登录成功!
用户 bigzzy 登录成功!
show
当前连接的客户端信息:
用户名: 1, IP地址: 127.0.0.1
用户名: bigzzy, IP地址: 127.0.0.1
用户 1 上传了文件: 1.jpg该文件在服务端的名字为: c160c78e5a57409488413f0170e9338d.jpg
用户 bigzzy 上传了文件: 2.jpg该文件在服务端的名字为: e13bf7696e7f4c23b33e65132eb7e64e.jpg
```

用户上传文件成功后，服务端会有相关信息的显示，包含用户名、原文件名、文件在服务端的新文件名。

```
TCP_Cliet x
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\IdeaC\dc
欢迎来到TCP传输系统!
请输入用户名: 1
请输入密码: 1
登录成功! 欢迎您, 1
请输入要传输的文件路径(输入quit结束):
F:\App\IdeaC\code\ComputerNet\Lab_02\Clietdir\1.jpg
上传成功!
请输入要传输的文件路径(输入quit结束):

TCP_Cliet x
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\IdeaC\down
欢迎来到TCP传输系统!
请输入用户名: bigzzy
请输入密码: 21120971
登录成功! 欢迎您, bigzzy
请输入要传输的文件路径(输入quit结束):
F:\App\IdeaC\code\ComputerNet\Lab_02\Clietdir\2.jpg
上传成功!
请输入要传输的文件路径(输入quit结束):

Run com.itheima.TCP_Final.Server x
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\IdeaC\dc
用户 1 登录成功!
用户 bigzzy 登录成功!
show
当前连接的客户端信息:
用户名: 1, IP地址: 127.0.0.1
用户名: bigzzy, IP地址: 127.0.0.1
用户 1 上传了文件: 1.jpg该文件在服务端的名字为: c160c78e5a57409488413f0170e9338d.jpg
用户 bigzzy 上传了文件: 2.jpg该文件在服务端的名字为: e13bf7696e7f4c23b33e65132eb7e64e.jpg
1已退出系统
```

用户通过输入文件路径选择需要上传的文件，服务端会反馈上传成功与否的信息。失败的原因可能是：输入了一个不存在的文件路径或是文件不存在。

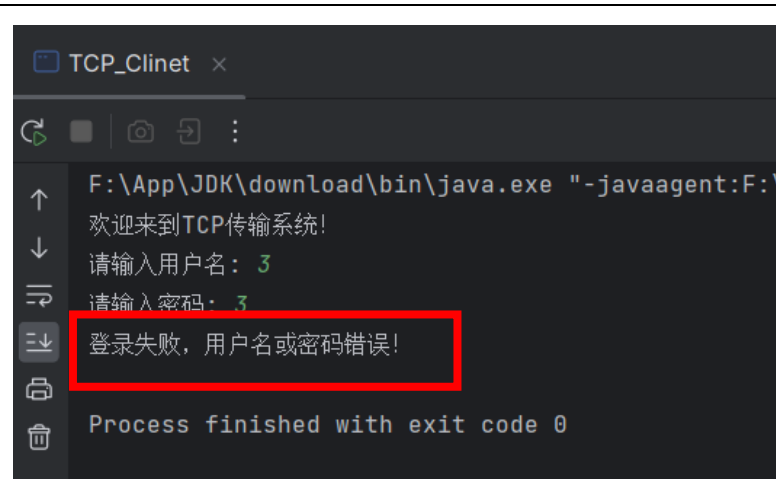
客户端退出:

```
Run com.itheima.TCP_Final.Server x
F:\App\JDK\download\bin\java.exe "-jav
用户 1 登录成功!
用户 bigzzy 登录成功!
show
当前连接的客户端信息:
用户名: 1, IP地址: 127.0.0.1
用户名: bigzzy, IP地址: 127.0.0.1
用户 1 上传了文件: 1.jpg该文件在服务端的名字为: c160c78e5a57409488413f0170e9338d.jpg
用户 bigzzy 上传了文件: 2.jpg该文件在服务端的名字为: e13bf7696e7f4c23b33e65132eb7e64e.jpg
1已退出系统

TCP_Cliet x
欢迎来到TCP传输系统!
请输入用户名: 1
请输入密码: 1
登录成功! 欢迎您, 1
请输入要传输的文件路径(输入quit结束):
F:\App\IdeaC\code\ComputerNet\Lab_02\Clietdir\1.jpg
上传成功!
请输入要传输的文件路径(输入quit结束):
quit
Process finished with exit code 0
```

在提示输入文件名时输入“quit”，那么就会退出客户端，并在服务端控制台显示该用户下线。

客户端登陆失败:



```
TCP_Clinet x
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\JDK\download\bin\javaagent.jar"
欢迎来到TCP传输系统!
请输入用户名: 3
请输入密码: 3
登录失败，用户名或密码错误!
Process finished with exit code 0
```

若用户输入的数据不在用户列表中，则该客户端连接直接关闭。

UDP:

● 主要数据结构:

1. **Map:** Map 是 Java 中用于存储键值对的接口，用于快速查找和存储数据。在这个代码中，您使用了多个 Map 来存储不同类型的信息。
userDatabase: 用于存储用户名和密码，以进行用户身份验证。
connectedClients: 用于存储已连接的客户端的用户名和其 IP 地址。
userMessageStatus: 用于存储用户的消息状态，指示用户是否已发送消息。

● 主要算法描述（传输消息的系统）

服务端:

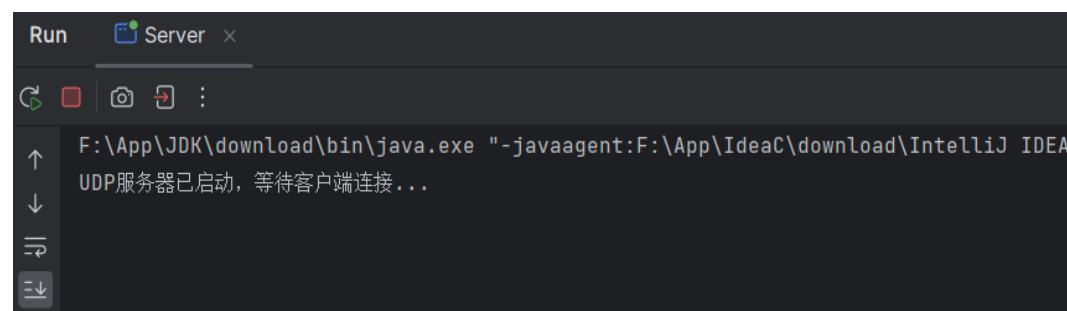
1. 初始化:
 - 启动服务器并指定监听 10086 端口。
 - 构建用户列表。
 - 启动一个线程监听控制台输入。
2. 处理客户端请求:
 - 接收用户端传来的用户名及密码进行验证
 - 向用户回传登陆结果。
 - 接收客户端传来的消息并显示于控制台。
3. 监听控制台输入:
 - **Show:** 显示所有发送过消息的客户端。

客户端:

1. 根据提示输入用户名与密码（服务端进行验证并回传信息）
2. 发送消息。

● 用户手册

■ UDP 系统启动:



```
Run Server x
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\IdeaC\download\IntelliJ IDEA
UDP服务器已启动，等待客户端连接...
```

■ 启动多个用户端登录并输入用户名与密码:



```
Run Server x
F:\App\JDK\download\bin\java.exe "-javaagent:
UDP服务器已启动，等待客户端连接...

Run UDP x
F:\App\JDK\download\bin\java.exe "-javaagent:
欢迎来到UDP传输系统!
请输入用户名: 1
请输入密码: 1
登录成功!
请输入要发送的消息 (输入 'quit' 结束):

Run UDP x
F:\App\JDK\download\bin\java.exe "-javaagent:
欢迎来到UDP传输系统!
请输入用户名: 2
请输入密码: 2
登录成功!
请输入要发送的消息 (输入 'quit' 结束):
```

■ 显示发送用户信息:

```
Run Server x UDP x UDP x
F:\App\JDK\download\bin\java.exe "-javaagent:
UDP服务器已启动, 等待客户端连接...
show
发送过信息的客户端:
无
F:\App\JDK\download\bin\java.exe "-javaagent:
欢迎来到UDP传输系统!
请输入用户名: 1
请输入密码: 1
登录成功!
请输入要发送的消息 (输入 'quit' 结束):
F:\App\JDK\download\bin\java.exe "-javaagent:
欢迎来到UDP传输系统!
请输入用户名: 2
请输入密码: 2
登录成功!
请输入要发送的消息 (输入 'quit' 结束):
```

此时由于用户 1 与用户 2 皆未发送过消息, 所以显示“无”。

■ 用户 1 向服务端发送消息:

```
Run Server x UDP x UDP x
F:\App\JDK\download\bin\java.exe "-javaagent:
UDP服务器已启动, 等待客户端连接...
1 发来消息: 我爱上海大学!
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\
欢迎来到UDP传输系统!
请输入用户名: 1
请输入密码: 1
登录成功!
请输入要发送的消息 (输入 'quit' 结束): 我爱上海大学!
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\
欢迎来到UDP传输系统!
请输入用户名: 2
请输入密码: 2
登录成功!
请输入要发送的消息 (输入 'quit' 结束):
```

用户 1 发送了消息, 服务端进行了显示。此时在服务端输入 show 会得到如下结果:

```
Run Server x UDP x UDP x
F:\App\JDK\download\bin\java.exe "-javaagent:
UDP服务器已启动, 等待客户端连接...
1 发来消息: 我爱上海大学!
show
发送过信息的客户端:
用户名: 1, IP地址: /127.0.0.1
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\
欢迎来到UDP传输系统!
请输入用户名: 1
请输入密码: 1
登录成功!
请输入要发送的消息 (输入 'quit' 结束): 我爱上海大学!
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\
欢迎来到UDP传输系统!
请输入用户名: 2
请输入密码: 2
登录成功!
请输入要发送的消息 (输入 'quit' 结束):
```

可以看到, 用户 1 发送过消息, 所以被显示出来, 用户 2 未发送消息, 故不显示。

■ 用户 2 向服务端发送消息:

```
Run Server x UDP x UDP x
F:\App\JDK\download\bin\java.exe "-javaagent:
UDP服务器已启动, 等待客户端连接...
1 发来消息: 我爱上海大学!
show
发送过信息的客户端:
用户名: 1, IP地址: /127.0.0.1
2 发来消息: 我爱计算机网络!
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\
欢迎来到UDP传输系统!
请输入用户名: 1
请输入密码: 1
登录成功!
请输入要发送的消息 (输入 'quit' 结束): 我爱上海大学!
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\
欢迎来到UDP传输系统!
请输入用户名: 2
请输入密码: 2
登录成功!
请输入要发送的消息 (输入 'quit' 结束): 我爱计算机网络!
```

此时, 用户 2 也发送来消息, 服务端控制台进行了显示。此时输入 show, 得到如下结果:

```
Run Server x UDP x UDP x
F:\App\JDK\download\bin\java.exe "-javaagent:
UDP服务器已启动, 等待客户端连接...
1 发来消息: 我爱上海大学!
show
发送过信息的客户端:
用户名: 1, IP地址: /127.0.0.1
2 发来消息: 我爱计算机网络!
show
发送过信息的客户端:
用户名: 1, IP地址: /127.0.0.1
用户名: 2, IP地址: /127.0.0.1
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\
欢迎来到UDP传输系统!
请输入用户名: 1
请输入密码: 1
登录成功!
请输入要发送的消息 (输入 'quit' 结束): 我爱上海大学!
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\
欢迎来到UDP传输系统!
请输入用户名: 2
请输入密码: 2
登录成功!
请输入要发送的消息 (输入 'quit' 结束):
```

这表明用户 2 也发送过消息了, 用户 1 与用户 2 都被记录了下来。

■ 退出客户端:

```
Run Server x UDP x
F:\App\JDK\download\bin\java.exe "-:
UDP服务器已启动, 等待客户端连接...
1 发来消息: 我爱上海大学!
show
发送过信息的客户端:
用户名: 1, IP地址: /127.0.0.1
2 发来消息: 我爱计算机网络!
show
发送过信息的客户端:
用户名: 2, IP地址: /127.0.0.1
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\
欢迎来到UDP传输系统!
请输入用户名: 1
请输入密码: 1
登录成功!
请输入要发送的消息 (输入 'quit' 结束): 我爱上海大学!
请输入要发送的消息 (输入 'quit' 结束): quit
客户端即将退出。
Process finished with exit code 0
F:\App\JDK\download\bin\java.exe "-javaagent:F:\App\
欢迎来到UDP传输系统!
请输入用户名: 2
请输入密码: 2
登录成功!
请输入要发送的消息 (输入 'quit' 结束): 我爱计算机网络!
请输入要发送的消息 (输入 'quit' 结束):
```

此时用户 1 输入了 quit, 退出了客户端。

■ 客户端信息验证失败:

```
Run Server x UDP x
F:\App\JDK\download\bin\java.exe "-:
UDP服务器已启动, 等待客户端连接...
F:\App\JDK\download\bin\java.
欢迎来到UDP传输系统!
请输入用户名: 3
请输入密码: 3
登录失败, 请重新登录。
请输入用户名:
```

可以看到, 用户 3 不在用户列表中, 服务端回传登录失败信息, 要求重新输入。

TCP 系统代码:

```
//Server 服务端
package com.itheima.TCP_Final;

import com.itheima.TCP_Final.MyRunnable;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.*;
import java.util.concurrent.*;

public class Server {

    private static final Map<String, String> userDatabase = new HashMap<>();
    private static final Map<String, String> connectedClients = new HashMap<>();
    static List<Socket> connectedSockets = new ArrayList<>();

    public static void main(String[] args) throws IOException {
        ThreadPoolExecutor pool = new ThreadPoolExecutor(
            4,
            16,
            60,
            TimeUnit.SECONDS,
            new ArrayBlockingQueue<>(2),
            Executors.defaultThreadFactory(),
            new ThreadPoolExecutor.AbortPolicy()
        );

        userDatabase.put("bigzzy", "21120971");
        userDatabase.put("maoyuxian", "21120956");
        userDatabase.put("chennann", "21120992");
        userDatabase.put("1", "1");
        userDatabase.put("2", "2");

        ServerSocket ss = new ServerSocket(10088);

        // 控制台输入处理线程
        new Thread() -> {
            BufferedReader consoleReader = new
            BufferedReader(new
            InputStreamReader(System.in));
            while (true) {
                try {
```

```

String command = consoleReader.readLine();
if ("show".equalsIgnoreCase(command)) {
    // 执行显示连接信息的操作
    showConnectedClients();
} else if (command.startsWith("kick")) {
    // 格式为 "kick+用户名"
    String[] parts = command.split(" ");
    if (parts.length == 2) {
        String usernameToKick = parts[1];
        if (usernameToKick != null && !usernameToKick.isEmpty()) {
            disconnectUser(usernameToKick);
        } else {
            System.out.println("无效的用户名。");
        }
    } else {
        System.out.println("无效的踢出命令格式。请使用：kick+用户名");
    }

} else if ("exit".equalsIgnoreCase(command)) {
    System.out.println("服务端即将关闭...");
    // 关闭服务器前关闭所有连接
    for (Socket clientSocket : connectedSockets) {
        try {
            BufferedWriter back = new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream()));
            back.write("与主机的连接已断开");
            back.newLine();
            back.flush();

            // 延迟一段时间，以确保消息发送出去
            Thread.sleep(1000); // 1 秒延迟
        } catch (IOException | InterruptedException e) {
            break;
        }
    }
    pool.shutdown();
    ss.close(); // 关闭服务器 Socket
    System.exit(0); // 退出服务器程序
} catch (IOException e) {
    break;
}
}

```



```

    }).start();

    while (true) {
        //等待客户来连接
        Socket socket = ss.accept();
        connectedSockets.add(socket);
        //开启一条线程
        //一个用户对应一条线程
        pool.submit(new MyRunnable(socket, userDatabase, connectedClients));
    }
}

public static void showConnectedClients() {
    if (connectedClients.isEmpty()) {
        System.out.println("当前连接的客户端信息：无");
    } else {
        System.out.println("当前连接的客户端信息：");
        for (Map.Entry<String, String> entry : connectedClients.entrySet()) {
            System.out.println("用户名：" + entry.getKey() + ", IP 地址：" +
entry.getValue());
        }
    }
}

private static String getUsernameFromSocket(Socket socket) {
    // 根据 Socket 获取对应的用户名
    String clientAddress = socket.getInetAddress().getHostAddress();
    for (Map.Entry<String, String> entry : connectedClients.entrySet()) {
        if (entry.getValue().equals(clientAddress)) {
            return entry.getKey();
        }
    }
    return null;
}

public static void disconnectUser(String username) {
    Iterator<Socket> iterator = connectedSockets.iterator();
    while (iterator.hasNext()) {
        Socket clientSocket = iterator.next();
        try {
            BufferedWriter writer = new BufferedWriter(new

```

```

OutputStreamWriter(clientSocket.getOutputStream()));
        String connectedUsername = getUsernameFromSocket(clientSocket);
        //System.out.println(connectedUsername);
        if (connectedUsername != null && connectedUsername.equals(username)) {
            writer.write("您已被管理员踢出");
            writer.newLine();
            writer.flush();
            clientSocket.close();
            iterator.remove(); // 安全地移除元素
            System.out.println("用户 " + username + " 已被断开连接。");
            return;
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
System.out.println("未找到用户 " + username + " 的连接。");
}
}

```

```

//MyRunnable
package com.itheima.TCP_Final;

import java.io.*;
import java.net.Socket;
import java.util.Map;
import java.util.UUID;

import static java.lang.System.out;

public class MyRunnable implements Runnable {

    Socket socket;
    private Map<String, String> userDatabase;
    private Map<String, String> connectedClients;

    public MyRunnable(Socket socket, Map<String, String> userDatabase, Map<String, String>
connectedClients) {
        this.socket = socket;
        this.userDatabase = userDatabase;
        this.connectedClients = connectedClients;
    }

    @Override

```

```

public void run() {
    try {
        BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String username = reader.readLine();
        String password = reader.readLine();

        if (authenticate(username, password)) {
            // 获取客户端的 IP 地址
            String clientAddress = socket.getInetAddress().getHostAddress();
            // 将客户端信息添加到 connectedClients 集合中
            connectedClients.put(username, clientAddress);

            BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
            bw.write("登录成功！欢迎您，" + username);
            bw.newLine();
            bw.flush();

            // 打印用户登录信息到服务端控制台
            out.println("用户 " + username + " 登录成功！");

            // 循环接收文件
            while (true) {
                String fileName = reader.readLine();
                if (fileName.equalsIgnoreCase("quit")) {
                    out.println(username+"已退出系统");
                    // 终止条件，收到 "quit" 后停止接收文件
                    break;
                }

                // 提取文件扩展名
                String fileExtension = getFileExtension(fileName);

                // 获取文件长度
                long fileLength = Long.parseLong(reader.readLine());

                // 读取数据并保存到本地中
                BufferedInputStream bis = new
BufferedInputStream(socket.getInputStream());
                String name = UUID.randomUUID().toString().replace("-", "");
                BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream("Lab_02\\Serverdir\\" + name + "." + fileExtension)); // 文件名包含格式

```

```

        long receivedBytes = 0;
        int len;
        byte[] bytes = new byte[1024];
        while ((receivedBytes < fileLength) && ((len = bis.read(bytes)) != -1)) {
            bos.write(bytes, 0, len);
            receivedBytes = receivedBytes + len;
        }

        // 回写数据
        BufferedWriter bww = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
        bww.write("上传成功!");
        bww.newLine();
        bww.flush();
        System.out.println("用户 " + username + " 上传了文件: " +
fileName+"该文件在服务端的名字为: "+name+"."+fileExtension);

    }
} else {
    // 用户验证失败, 回写错误信息
    BufferedWriter bwq = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
    bwq.write("登录失败, 用户名或密码错误!");
    bwq.newLine();
    bwq.flush();
    // 关闭 Socket 连接, 直接退出程序
    socket.close();
}
} catch (IOException e) {
    throw new RuntimeException(e);
} finally {
    // 释放资源
    if (socket != null) {
        try {
            // 获取用户名, 以便在用户下线时从 connectedClients 集合中移除
            String username = getUsernameFromSocket(socket);
            if (username != null) {
                connectedClients.remove(username);
            }
            socket.close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
}

```

```

        }
    }
}

private String getFileExtension(String fileName) {
    int lastDotIndex = fileName.lastIndexOf(".");
    if (lastDotIndex != -1) {
        return fileName.substring(lastDotIndex + 1);
    }
    return "";
}

private boolean authenticate(String username, String password) {
    String storedPassword = userDatabase.get(username);
    return storedPassword != null && storedPassword.equals(password);
}

private String getUsernameFromSocket(Socket socket) {
    // 根据 Socket 获取对应的用户名
    String clientAddress = socket.getInetAddress().getHostAddress();
    for (Map.Entry<String, String> entry : connectedClients.entrySet()) {
        if (entry.getValue().equals(clientAddress)) {
            return entry.getKey();
        }
    }
    return null;
}
}

```

//Client 客户端

```

package com.itheima.TCP_Final;

import java.io.*;
import java.net.Socket;
import java.util.Scanner;

import static java.lang.Thread.sleep;

public class Client {
    public static void main(String[] args) throws IOException, InterruptedException {
        Socket socket = new Socket("127.0.0.1", 10088);
        Scanner scanner = new Scanner(System.in);
        System.out.println("欢迎来到 TCP 传输系统！");
        System.out.print("请输入用户名: ");
        String username = scanner.nextLine();
    }
}

```

```

System.out.print("请输入密码: ");
String password = scanner.nextLine();

// 发送用户名和密码到服务器
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
out.println(username);
out.println(password);

// 读取服务器响应
BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
String response = br.readLine();
System.out.println(response);

if (response.startsWith("登录成功")) {
    Thread readThread = new Thread() -> {
        try {
            BufferedReader brr = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            String line;
            while ((line = brr.readLine()) != null) {
                System.out.println(line);
                if ("与主机的连接已断开".equals(line)) {
                    // 处理连接断开的逻辑，例如关闭客户端程序
                    System.out.println("与主机的连接已断开，客户端即将退出。");
                }

                socket.close();
                System.exit(0);
                break;
            }
            if ("您已被管理员踢出".equals(line)) {
                // 处理连接断开的逻辑，例如关闭客户端程序
                System.out.println("客户端即将退出...");
                socket.close();
                System.exit(0);
            }
        }
    } catch (IOException e) {

    }
};
readThread.start();

while (true) {

```

```

        System.out.println("请输入要传输的文件路径（输入 quit 结束）:");

        String filePath = scanner.nextLine();

        if (filePath.equalsIgnoreCase("quit")) {
            // 发送退出命令
            out.println("quit");
            break;
        }

        File file = new File(filePath);
        if (file.exists() && file.isFile()) {
            // 发送文件名到服务器
            out.println(file.getName());
            // 发送文件长度到服务器
            out.println(file.length());

            BufferedInputStream bis = new BufferedInputStream(new
FileInputStream(file));
            BufferedOutputStream bos = new
BufferedOutputStream(socket.getOutputStream());
            byte[] bytes = new byte[1024];
            int len;
            while ((len = bis.read(bytes)) != -1) {
                bos.write(bytes, 0, len);
            }
            bos.flush();
        } else {
            System.out.println("文件不存在或不是一个有效的文件，请重新输入
文件路径:");
        }
        sleep(1000);//保证监听线程的及时输出
    }
}
socket.close();
}
}

```

UDP 系统代码

```

//Server 服务端
package com.itheima.UDPTTest2;

import java.io.BufferedReader;

```

```

import java.io.IOException;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.HashMap;
import java.util.Map;

public class Server {
    private static final int SERVER_PORT = 10086;

    private static final Map<String, String> userDatabase = new HashMap<>();
    private static final Map<String, InetAddress> connectedClients = new HashMap<>();
    private static final Map<String, Boolean> userMessageStatus = new HashMap<>();

    static {
        // 在用户数据库中添加用户名和密码
        userDatabase.put("1", "1");
        userDatabase.put("2", "2");
    }

    public static void main(String[] args) throws IOException {
        DatagramSocket socket = new DatagramSocket(SERVER_PORT);
        byte[] receiveData = new byte[1024];

        System.out.println("UDP 服务器已启动，等待客户端连接...");

        // 创建控制台输入线程
        Thread consoleInputThread = new Thread(() -> {
            try {
                BufferedReader consoleReader = new BufferedReader(new
InputStreamReader(System.in));
                while (true) {
                    String command = consoleReader.readLine();
                    if ("show".equalsIgnoreCase(command)) {
                        showConnectedClients();
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
        consoleInputThread.start();
    }

```



```

        while (true) {
            DatagramPacket    receivePacket    =    new    DatagramPacket(receiveData,
receiveData.length);
            socket.receive(receivePacket);

            InetAddress clientAddress = receivePacket.getAddress();
            int clientPort = receivePacket.getPort();
            String    receivedMessage    =    new    String(receivePacket.getData(),    0,
receivePacket.getLength());

            if (receivedMessage.startsWith("LOGIN:")) {
                // 处理登录请求
                String    responseMessage    =    handleLoginRequest(receivedMessage,
clientAddress);
                sendResponseMessage(socket, clientAddress, clientPort, responseMessage);
            } else {
                // 启动一个新线程来处理已登录客户端的消息
                String username = getUsername(clientAddress);
                if (username != null) {
                    Thread clientThread = new Thread(() -> handleClientMessage(username,
receivedMessage, socket, clientAddress, clientPort));
                    clientThread.start();
                } else {
                    System.out.println("未登录的客户端发来消息: " + receivedMessage);
                }
            }
        }
    }

    private static String handleLoginRequest(String message, InetAddress clientAddress) {
        String[] parts = message.split(":");
        if (parts.length == 3) {
            String username = parts[1];
            String password = parts[2];
            if (userDatabase.containsKey(username) &&
userDatabase.get(username).equals(password)) {
                // 登录成功
                connectedClients.put(username, clientAddress);
                userMessageStatus.put(username, false); // 初始状态为未发送消息
                //System.out.println(username + " 登录成功。");
                return "LOGIN_SUCCESS";
            } else {
                // 登录失败
                //System.out.println("登录失败，用户名或密码错误。");
            }
        }
    }

```

```

        return "LOGIN_FAILED";
    }
}
return "LOGIN_REQUEST_FORMAT_ERROR";
}

private static void sendResponseMessage(DatagramSocket socket, InetAddress
clientAddress, int clientPort, String message) throws IOException {
    byte[] responseData = message.getBytes();
    DatagramPacket responsePacket = new DatagramPacket(responseData,
responseData.length, clientAddress, clientPort);
    socket.send(responsePacket);
}

private static String getUsername(InetAddress clientAddress) {
    for (Map.Entry<String, InetAddress> entry : connectedClients.entrySet()) {
        if (entry.getValue().equals(clientAddress)) {
            return entry.getKey();
        }
    }
    return null;
}

private static void handleClientMessage(String username, String message, DatagramSocket
socket, InetAddress clientAddress, int clientPort) {
    // 提取用户名和消息内容
    String[] parts = message.split(":");
    if (parts.length == 2) {
        String usernameFromMessage = parts[0];
        String messageContent = parts[1];

        System.out.println(usernameFromMessage + " 发来消息: " + messageContent);

        // 更新用户的消息状态为已发送消息
        userMessageStatus.put(usernameFromMessage, true);
    }
}

private static void showConnectedClients() {
    System.out.println("发送过信息的客户端:");
    boolean exist = false;
    for (Map.Entry<String, InetAddress> entry : connectedClients.entrySet()) {
        String username = entry.getKey();
        boolean hasSentMessage = userMessageStatus.getOrDefault(username, false);
    }
}

```

```

        if (hasSentMessage) {
            exist = true;
            System.out.println("用户名: " + username + ", IP 地址: " + entry.getValue());
        }
    }
    if (!exist) {
        System.out.println("无");
    }
}
}
}

```

//Client 客户端

```

package com.itheima.UDPTTest2;

import java.io.IOException;
import java.net.*;
import java.util.Scanner;

public class Clinet {
    private static final int SERVER_PORT = 10086;
    private static final String SERVER_ADDRESS = "127.0.0.1"; // 本地主机地址

    public static void main(String[] args) throws IOException {
        DatagramSocket socket = new DatagramSocket();
        Scanner scanner = new Scanner(System.in);

        System.out.println("欢迎来到 UDP 传输系统! ");
        String username = null;

        InetAddress serverHost = InetAddress.getByName(SERVER_ADDRESS);

        while (true) {
            if (username == null) {
                // 如果未登录，则进行登录
                System.out.print("请输入用户名: ");
                String inputUsername = scanner.nextLine();
                System.out.print("请输入密码: ");
                String password = scanner.nextLine();

                // 发送登录请求消息
                String loginMessage = "LOGIN:" + inputUsername + ":" + password;
                byte[] loginData = loginMessage.getBytes();
                DatagramPacket loginPacket = new DatagramPacket(loginData,
loginData.length, serverHost, SERVER_PORT);
                socket.send(loginPacket);
            }
        }
    }
}

```

```

        // 接收服务器响应
        byte[] receiveData = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
        socket.receive(receivePacket);

        String responseMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
        if ("LOGIN_SUCCESS".equals(responseMessage)) {
            System.out.println("登录成功！");
            username = inputUsername; // 登录成功后设置当前用户的用户名
        } else {
            System.out.println("登录失败，请重新登录。");
        }
    } else {
        // 已登录状态，允许发送消息
        System.out.print("请输入要发送的消息（输入 'quit' 结束）：");
        String message = scanner.nextLine();

        if ("quit".equalsIgnoreCase(message)) {
            System.out.println("客户端即将退出。");
            break;
        }

        // 发送消息时包含用户名信息
        String messageWithUsername = username + ":" + message;
        byte[] sendData = messageWithUsername.getBytes();
        DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, serverHost, SERVER_PORT);
        socket.send(sendPacket);
    }
}
socket.close();
}
}

```