目录

[一、 实验要求 2](#_Toc156227374)

[二、协议原理 2](#_Toc156227375)

[2.1 TCP协议 2](#_Toc156227376)

[2.2 UDP协议 4](#_Toc156227377)

[2.3 TCP协议与UDP协议的关系 6](#_Toc156227378)

[2.4 Socket网络编程 8](#_Toc156227379)

[三、程序功能与流程 10](#_Toc156227380)

[3.1 程序功能设计与程序流程 10](#_Toc156227381)

[3.2 程序流程 10](#_Toc156227382)

[四、分析程序代码 10](#_Toc156227383)

[4.1 导入相应库并启动主函数 11](#_Toc156227384)

[4.2 初始化Socket 12](#_Toc156227385)

[4.3 初始化GUI 13](#_Toc156227386)

[4.4 发送消息 16](#_Toc156227387)

[4.5 接受消息 17](#_Toc156227388)

[4.6 护眼模式与白黑模式 19](#_Toc156227389)

[4.7 下载数据到数据库 20](#_Toc156227390)

[五、总结 21](#_Toc156227391)

[六、参考文献 22](#_Toc156227392)

实验报告

## 实验要求

1. 基本要求：设计 Server 和 Client，实现多用户同时聊天通信以及私聊等。2. 设计语言：Python、C/C++。3. 套接字程序设计。4. 技术难点：用户注册、聊天记录保存到数据库中，图形化界面、个性化特色。

## 二、协议原理

## 2.1 TCP协议

TCP（Transmission Control Protocol）是一种面向连接的、可靠的传输层协议，用于在计算机网络中传输数据。它是互联网协议套件（TCP/IP）中最常用的协议之一，被广泛用于应用层协议（如HTTP、FTP、SMTP等）的可靠数据传输。

下面是对TCP协议及其特点的详细解释：

1. 面向连接：在进行数据传输之前，发送方和接收方需要通过三次握手建立一个连接。连接的建立包括发送方向接收方发送一个连接请求（SYN），接收方向发送方发送一个连接确认（SYN-ACK），以及发送方向接收方发送一个连接确认（ACK）。连接的建立确保了双方都愿意进行数据传输，并为后续的数据传输提供了必要的准备。

2. 可靠性：TCP通过使用序列号、确认应答和重传等机制来确保可靠的数据传输。每个TCP报文段都包含一个序列号，接收方通过确认应答来告知发送方已经接收到哪些数据。如果发送方没有接收到确认应答或者接收到了超时的确认应答，它将会重新发送未确认的数据。这种机制确保了数据的可靠性，即使在网络丢包或出现错误的情况下也能够进行恢复。

3. 流量控制：TCP使用滑动窗口机制来进行流量控制。接收方通过告知发送方自己的缓冲区大小来限制发送方发送的数据量。发送方根据接收方提供的窗口大小来控制发送的数据量，以确保接收方能够及时处理接收到的数据。

4. 拥塞控制：TCP使用拥塞控制算法来避免网络拥塞。发送方通过动态调整发送的数据量来适应网络的状况。当网络拥塞时，发送方减少发送的数据量，以降低网络负载。当网络状况改善时，发送方逐渐增加发送的数据量，以提高数据传输的效率。

5. 有序性：TCP保证数据的有序传输。接收方根据接收到的TCP报文段的序列号对它们进行重新排序，以确保数据按照发送方发送时的顺序进行传输。

6. 双工通信：TCP支持全双工通信，即发送方和接收方可以同时进行数据的发送和接收。这使得双方能够同时进行数据交换，提高了通信的效率。

总结起来，TCP是一种可靠的面向连接的传输协议，通过建立连接、使用序列号和确认应答、流量控制和拥塞控制等机制，提供了可靠的数据传输、有序性、流量控制和拥塞控制等功能。它是互联网上应用层协议进行可靠数据传输的基础。

上述讲到了TCP协议及其特点，此外，TCP协议在互联网中有广泛的应用场景，以下是一些常见的应用场景：

1. 网页浏览：TCP作为HTTP协议的可靠传输层协议，用于在浏览器和服务器之间传输网页内容。当您在浏览器中输入网址并请求网页时，TCP协议负责将网页内容分成多个TCP报文段，并通过互联网将它们从服务器传输到您的浏览器。

2. 文件传输：TCP协议也被用于文件传输协议，如FTP（File Transfer Protocol）。通过TCP的可靠性和流量控制，FTP可以确保文件在客户端和服务器之间的可靠传输。

3. 电子邮件：TCP协议用于电子邮件的传输。当您发送电子邮件时，您的电子邮件客户端使用TCP将邮件传输到邮件服务器，而接收方的电子邮件客户端使用TCP从邮件服务器接收邮件。

4. 远程登录：TCP协议被用于远程登录协议，如Telnet和SSH。这些协议允许用户通过网络远程登录到远程计算机，并与其进行交互。TCP协议提供了可靠的数据传输和双向通信的支持，确保用户与远程计算机之间的交互是稳定和可靠的。

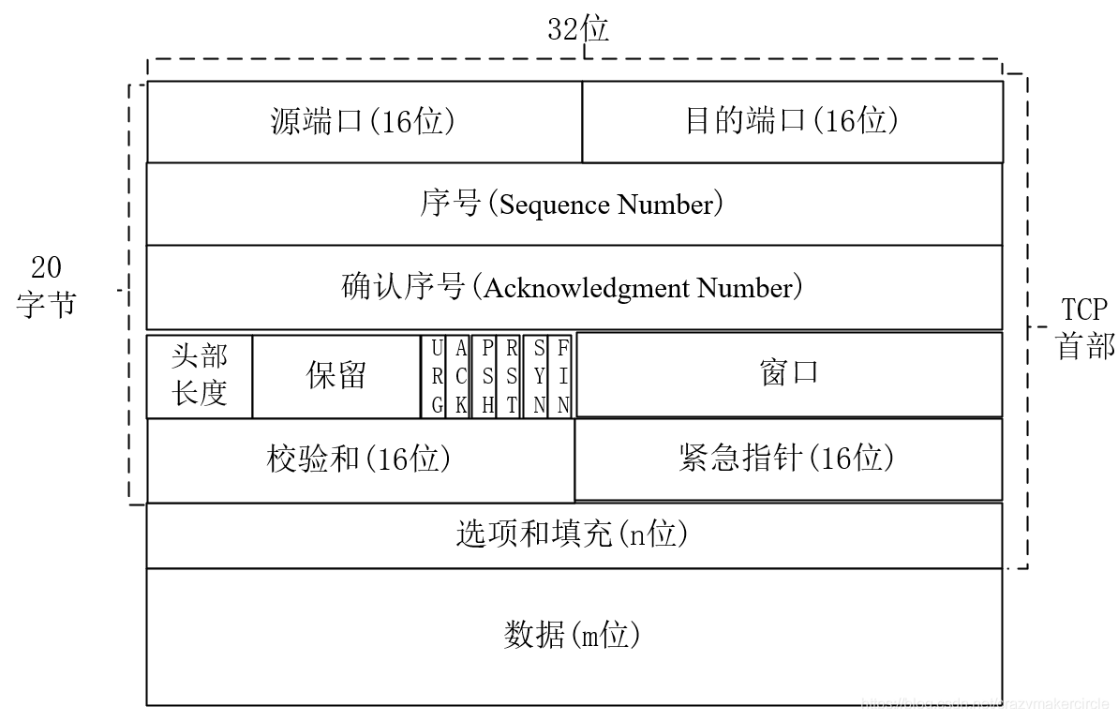
5. 数据库访问：TCP协议常被用于数据库访问，如MySQL和PostgreSQL。通过TCP协议，客户端可以与数据库服务器建立连接，并进行查询、更新和管理数据库中的数据。

6. 实时通信：TCP协议也被用于实时通信应用，如即时通讯（Instant Messaging）和语音通话。通过TCP的可靠性和流量控制，这些应用可以确保消息和音频数据在用户之间的实时传输。

需要注意的是，尽管TCP协议在许多应用中被广泛使用，但对于某些特定的应用，如实时视频流或大规模数据传输等，可能会选择使用UDP协议，因为UDP具有更低的延迟和更高的传输速度，但可靠性较差。因此，在选择使用TCP还是UDP时，需要根据应用的需求来做出权衡。

上述了解了TCP协议、特点还有它适合的场景，为了更加详细的了解他，需要掌握它的报文格式：

TCP（Transmission Control Protocol）报文格式如下所示：



下面是对各个字段的详细解释：

1. 源端口号（Source Port）：16位字段，用于标识发送方的应用程序或服务的端口号。

2. 目标端口号（Destination Port）：16位字段，用于标识接收方的应用程序或服务的端口号。

3. 序列号（Sequence Number）：32位字段，用于对TCP数据流中的每个字节进行编号。序列号用来保证数据的顺序性。

4. 确认号（Acknowledgment Number）：32位字段，用于确认已经收到的数据的序列号。确认号表示期望接收的下一个字节的序列号。

5. 数据偏移（Data Offset）：4位字段，表示TCP报文头部的长度，以4字节为单位。TCP报文头长度最小为20字节。

6. 保留（Reserved）：6位字段，保留用于将来的扩展。

7. URG标志（URG）：1位字段，表示紧急指针字段是否有效。

8. ACK标志（ACK）：1位字段，表示确认号字段是否有效。

9. PSH标志（PSH）：1位字段，表示接收方是否应该尽快将数据交给应用层。

10. RST标志（RST）：1位字段，表示中断连接。

11. SYN标志（SYN）：1位字段，用于建立连接时进行同步。

12. FIN标志（FIN）：1位字段，表示发送方已经完成数据的发送，准备关闭连接。

13. 窗口大小（Window Size）：16位字段，表示接收方可接收的字节数。

14. 校验和（Checksum）：16位字段，用于检验TCP报文的完整性。

15. 紧急指针（Urgent Pointer）：16位字段，用于指示紧急数据的字节偏移量。

16. 选项（Options）：可选字段，用于在TCP报文中添加一些可选的功能或参数。

17. 数据（Data）：可选字段，用于携带应用层的数据。

TCP报文格式中的各个字段共同构成了TCP协议数据传输的头部部分，提供了必要的控制和管理信息，以实现可靠的、有序的数据传输。

## 2.2 UDP协议

UDP（User Datagram Protocol）是一种面向无连接的传输层协议，与TCP一样属于互联网协议套件（TCP/IP）中的一部分。与TCP不同，UDP不提供可靠性、流量控制、拥塞控制和有序性等特性，但具有较低的延迟和较高的传输速度。

对此，我们详细的对UDP协议进行解释，以下是对UDP协议及其特点的详细解释：

1. 面向无连接：UDP不需要像TCP那样在通信之前建立连接。发送方将数据打包成UDP数据报，直接发送给接收方，而不需要进行握手和建立连接的过程。因此，UDP具有较低的开销和较快的传输速度。

2. 不可靠性：UDP协议不提供数据传输的可靠性保证。一旦发送方将数据发送出去，就无法得知接收方是否成功接收到数据。UDP不使用确认应答和重传机制，因此在网络丢包或错误的情况下，数据可能会丢失或出现错误。

3. 简单性：UDP协议相对简单，头部开销较小。UDP数据报的头部只有8字节，而TCP头部则有20字节的开销。由于不需要维护连接状态、序列号、确认应答等机制，UDP协议的实现较为简单，也减少了网络开销。

4. 快速传输：由于UDP不需要进行连接的建立和维护，以及不需要进行可靠性保证的机制，它具有较低的延迟和较高的传输速度。这使得UDP在实时应用场景，如语音通话、视频流和实时游戏等方面得到广泛应用。

5. 广播和多播：UDP支持广播和多播功能。通过UDP协议，发送方可以将数据报发送给同一网络中的多个接收方。这在实现多播通信和实时流媒体等应用中非常有用。

6. 适用场景：UDP适用于那些对可靠性要求较低但对实时性要求较高的应用，如实时音视频传输、DNS（域名系统）、SNMP（简单网络管理协议）等。它也常被用于一些需要快速传输的简单请求-响应应用，如NTP（网络时间协议）。

需要注意的是，由于UDP的不可靠性和无连接性，它不适合用于那些对数据完整性和可靠性要求较高的应用，例如文件传输和可靠的数据传输。在这些情况下，更适合选择TCP协议。

上述讲到了TCP协议及其特点，此外，UDP协议在现实场景也有适用的应用场景，如以下几种：

1. 实时音视频传输：UDP协议因其较低的延迟和较高的传输速度而在实时音视频传输中得到广泛应用。例如，视频会议、实时直播和在线游戏等应用通常使用UDP来传输实时的音视频数据。在这些场景下，实时性比可靠性更为重要，因此选择UDP可以减少传输延迟，提供更好的用户体验。

2. DNS（域名系统）：UDP协议常用于域名系统（DNS）的查询和响应过程。DNS用于将域名（例如www.example.com）转换为IP地址。由于DNS查询通常是短暂的、简单的请求-响应过程，并且需要快速的响应时间，所以UDP协议非常适合用于DNS通信。

3. 实时游戏：UDP协议在实时游戏中广泛使用。游戏中的实时性非常重要，玩家的操作和反馈需要以最小的延迟进行传输。UDP的低延迟和较高的传输速度使其成为实时游戏中常用的协议，用于传输玩家的位置、动作和游戏状态等数据。

4. SNMP（简单网络管理协议）：SNMP是一种用于网络设备管理的协议，UDP是SNMP的传输层协议。SNMP通常用于获取和设置网络设备的状态、监控网络性能和进行故障排除。由于SNMP的简单请求-响应模式和需要快速的响应速度，UDP协议非常适合用于SNMP通信。

5. 实时传感器数据：UDP协议也适用于实时传感器数据的传输。例如，监测设备、传感器网络和物联网应用中的传感器数据通常需要实时传输，并且对于数据的完整性要求相对较低。UDP的低延迟和简单性使其成为这些应用中常用的协议。

需要注意的是，UDP协议的不可靠性意味着在数据传输过程中可能会出现丢包、错误和重复等问题。因此，在选择UDP作为传输协议时，应权衡好实时性和可靠性之间的需求，并采取额外的机制来处理数据的完整性和错误处理。

相比与TCP协议，UDP协议的报文格式就简单的多，如下表所示：

UDP（User Datagram Protocol）报文格式如下所示：

+-----------------+-----------------+

| 源端口号 | 目标端口号 |

+-----------------+-----------------+

| 长度 | 校验和 |

+-----------------+-----------------+

| 数据（可选） |

+-----------------+-----------------+

下面是对各个字段的详细解释：

1. 源端口号（Source Port）：16位字段，用于标识发送方的应用程序或服务的端口号。

2. 目标端口号（Destination Port）：16位字段，用于标识接收方的应用程序或服务的端口号。

3. 长度（Length）：16位字段，表示UDP报文的总长度（包括报文头部和数据），以字节为单位。

4. 校验和（Checksum）：16位字段，用于检验UDP报文的完整性。校验和计算包括UDP报文头部和数据部分。

5. 数据（Data）：可选字段，用于携带应用层的数据。UDP报文中的数据部分没有固定长度，长度可以根据应用需求而变化。

UDP报文格式相对于TCP报文格式简单，它没有序列号、确认号、窗口大小等与可靠性相关的字段。UDP是一种无连接的传输协议，不提供数据的可靠性保证和顺序保证。UDP报文只包含了必要的字段，以及应用层的数据，因此UDP报文的开销相对较小。

UDP报文的简洁性和较小的开销使得它适用于对实时性要求较高、对可靠性要求较低的应用场景，如实时音视频传输、实时游戏、DNS查询等。UDP在这些应用中可以提供较低的延迟和较高的传输速度，但可能会牺牲数据的可靠性和顺序性。在使用UDP进行数据传输时，应用程序需要自行处理数据的完整性、丢失和重复等问题。

## 2.3 TCP协议与UDP协议的关系

TCP协议（Transmission Control Protocol）和UDP协议（User Datagram Protocol）是互联网协议套件（TCP/IP）中的两个主要的传输层协议。它们在网络通信中扮演不同的角色，但彼此之间有一些关系。

1. 同属于传输层：TCP和UDP都属于TCP/IP协议套件中的传输层协议。传输层负责在网络中的两个节点之间提供端到端的通信服务。

2. 互补的功能：TCP和UDP在功能上是互补的。TCP提供面向连接的、可靠的数据传输，确保数据的完整性、顺序性和可靠性。UDP则是面向无连接的、不可靠的数据传输，适用于实时性要求较高、对可靠性要求较低的应用场景。

3. 共享网络基础设施：TCP和UDP共享同一网络基础设施。它们都使用IP协议进行数据包的路由和传输。无论是TCP还是UDP，它们的数据都会被分割成数据包，在网络中进行传输。

4. 端口的共享：TCP和UDP使用端口号来标识不同的应用程序或服务。通过端口号，网络中的数据包可以被正确地交付给相应的应用程序。TCP和UDP使用相同的16位端口号空间，但同一个端口号可以被TCP和UDP同时使用，因为它们的连接方式和数据报结构不同。

5. TCP-over-UDP：有时候，UDP协议可以用作TCP-over-UDP的底层传输。这是一种在UDP上实现类似TCP的可靠性机制的技术。通过在UDP上构建可靠性机制，可以在一些特殊的网络环境中实现TCP类似的功能，同时兼顾UDP的低延迟和高吞吐量的优势。

综上所述，TCP和UDP在网络通信中有不同的特点和功能，可以根据具体的应用需求选择使用。TCP提供可靠性和顺序性，适用于对数据完整性和可靠性要求较高的场景。UDP提供较低的延迟和较高的传输速度，适用于实时性要求较高、对可靠性要求较低的场景。

上述我们分析了TCP协议与UDP协议在同属传输层，功能，网络基础设施，端口以及在底层应用上关系，为了更加深入的阐述TCP与UDP之间的关系，我们讲阐述他们之间的区别与对比，如下所示。

TCP协议（Transmission Control Protocol）和UDP协议（User Datagram Protocol）是互联网协议套件（TCP/IP）中的两个重要的传输层协议。它们在功能、特点和适用场景上有以下几个主要区别：

1. 连接性：

- TCP：TCP是面向连接的协议，它通过建立连接、维护状态和进行握手来确保可靠的数据传输。连接的建立和关闭过程需要时间，并且在通信期间保持连接状态。

- UDP：UDP是面向无连接的协议，它不需要事先建立连接，也不维护连接状态。每个UDP数据报都是独立的单元，发送方将数据报发送给接收方，而不需要握手和确认。

2. 可靠性：

- TCP：TCP提供可靠的数据传输，它使用确认应答机制和重传机制来确保数据的可靠性。如果数据包丢失或损坏，TCP会自动重传，以保证数据的完整性和顺序性。

- UDP：UDP不提供可靠性保证。它不使用确认应答和重传机制，因此在网络出现丢包或错误的情况下，数据可能会丢失或乱序。

3. 延迟和传输速度：

- TCP：由于TCP需要建立连接和维护状态，以及提供可靠性保证，它的传输速度相对较慢。TCP还会根据网络状态自动调整传输速率，以避免网络拥塞。

- UDP：由于UDP不需要建立连接和维护状态，以及提供可靠性保证，它的传输速度相对较快。UDP的延迟较低，适合实时性要求较高的应用。

4. 数据完整性：

- TCP：TCP协议确保数据的完整性，它使用序列号和校验和等机制来保证数据的正确性。如果数据包损坏或丢失，TCP会自动重传，以确保数据的完整性。

- UDP：UDP不提供数据的完整性保证。接收方无法知道是否接收到完整和正确的数据，因此在需要数据完整性的应用中，需要额外的机制来验证和保证数据的正确性。

5. 适用场景：

- TCP：TCP适用于对数据可靠性和顺序性要求较高的应用，如网页浏览、文件传输、电子邮件、远程登录等。TCP适合于大规模数据传输和长时间的通信。

- UDP：UDP适用于对实时性和传输速度要求较高，而对数据可靠性要求较低的应用，如实时音视频传输、实时游戏、DNS查询、SNMP等。UDP适合于短时间的通信和小型数据传输。

需要根据具体的应用需求来选择使用TCP或UDP。在某些情况下，也可以结合使用两者，例如实时音视频传输中使用UDP进行数据传输，同时使用TCP进行控制信令和可靠性保证。

## 2.4 Socket网络编程

Socket网络编程是一种在计算机网络中进行通信的编程接口。它提供了一种在不同主机之间进行数据传输的方式，使得应用程序能够通过网络进行通信。

Socket网络编程基于套接字（Socket）的概念。套接字是一种抽象的通信端点，可以通过网络与其他主机上的套接字进行连接，进行数据的发送和接收。

Socket网络编程涉及两个主要方面：服务器端和客户端。

服务器端：

1. 创建套接字：服务器端首先需要创建一个套接字，用于监听和接受客户端的连接请求。可以使用`socket()`函数创建套接字，并指定通信协议（如TCP或UDP）和地址族（如IPv4或IPv6）。

2. 绑定地址：服务器端需要将套接字与一个特定的地址（IP地址和端口号）绑定，以便客户端能够连接到服务器。使用`bind()`函数将套接字与地址绑定。

3. 监听连接：服务器端调用`listen()`函数开始监听连接请求，等待客户端的连接。

4. 接受连接：一旦有客户端连接请求到达，服务器端使用`accept()`函数接受连接，并创建一个新的套接字用于与客户端进行通信。

5. 数据交换：通过新创建的套接字，服务器端可以与客户端进行数据的收发操作。可以使用`send()`函数发送数据给客户端，使用`recv()`函数接收来自客户端的数据。

客户端：

1. 创建套接字：客户端需要创建一个套接字，用于与服务器端建立连接。同样，可以使用`socket()`函数创建套接字，并指定通信协议和地址族。

2. 连接服务器：客户端使用`connect()`函数向服务器端发起连接请求，并指定服务器的地址（IP地址和端口号）。

3. 数据交换：一旦连接建立成功，客户端可以使用套接字进行与服务器端的数据交换。可以使用`send()`函数发送数据给服务器端，使用`recv()`函数接收来自服务器端的数据。

Socket网络编程可以使用不同的协议，如TCP（面向连接）和UDP（无连接），以满足不同的应用需求。对于TCP套接字，数据的传输是可靠的、有序的，而UDP套接字提供了无连接的、不可靠的数据传输。

通过Socket网络编程，开发者可以实现各种网络应用，如Web服务器、聊天应用程序、文件传输等，实现不同主机之间的通信和数据交换。

以下是一个简单的使用UDP进行Socket编程的Python示例：

```python

import socket

# 服务器端

def server():

# 创建UDP套接字

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# 绑定地址和端口

server\_address = ('localhost', 8888)

server\_socket.bind(server\_address)

while True:

# 接收数据

data, address = server\_socket.recvfrom(1024)

print(f"Received data from {address}: {data.decode()}")

# 发送响应

response = "Hello, client!"

server\_socket.sendto(response.encode(), address)

# 客户端

def client():

# 创建UDP套接字

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# 服务器地址和端口

server\_address = ('localhost', 8888)

while True:

# 发送数据

message = input("Enter message: ")

client\_socket.sendto(message.encode(), server\_address)

# 接收响应

response, \_ = client\_socket.recvfrom(1024)

print("Received response:", response.decode())

# 在不同的终端或计算机上运行服务器端和客户端函数

```

上述示例中，服务器端通过创建UDP套接字，并绑定地址和端口。然后进入一个无限循环，不断接收来自客户端的数据，并发送响应。客户端同样创建UDP套接字，并指定服务器的地址和端口。客户端通过输入消息并发送给服务器端，然后接收服务器端的响应。

通过运行服务器端和客户端函数，可以在本地主机上进行UDP通信。当客户端发送消息时，服务器端接收并打印消息内容，然后发送响应给客户端，客户端接收并打印响应内容。

这只是一个简单的UDP Socket编程示例，实际上可以根据具体需求进行更复杂的功能扩展。

## 三、程序功能与流程

## 3.1 程序功能设计与程序流程

1. UDP实现聊天通信功能：

- 服务器端接收到用户的消息后，将其广播给所有在线用户，实现多用户同时聊天通信。

- 用户可以在聊天过程中随时退出，服务器端需要处理用户退出的情况。

2.保存聊天记录：使用sqlite数据库将用户的聊天记录保存下来。

3. 图形化界面和个性化特色：

- 可以使用图形化界面设计，提供更友好的用户交互体验。

- 可以添加个性化特色，如护眼模式、消息样式自定义等。

## 3.2 程序流程

1. 服务器端启动，创建套接字并绑定地址和端口。

2. 服务器端等待客户端连接请求。

3. 客户端启动，创建套接字并连接到服务器端。

4. 客户端进行用户注册或登录操作。

5. 客户端输入消息后，通过套接字将消息发送给服务器端。

6. 服务器端接收到消息后，根据消息类型进行处理：

7. 客户端接收服务器端发送的消息，并显示在聊天界面中。

8. 用户可以随时退出聊天，客户端发送退出消息给服务器端，并关闭套接字。

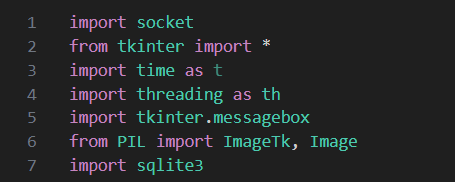
9. 服务器端接收到退出消息后，将用户从在线列表中移除，更新用户状态。

10. 服务器端继续等待其他客户端的连接请求，循环执行步骤2-10。

## 四、分析程序代码

分析程序代码，以服务端为例，客户端类似

## 4.1 导入相应库并启动主函数



图一：需要的库

socket 库：socket 是 Python 中用于网络编程的标准库。它提供了一组函数和类，用于创建网络套接字，进行网络通信和处理网络数据。通过 socket 库，你可以在 Python 中实现各种网络应用，如创建客户端和服务器，进行网络通信和传输数据等。

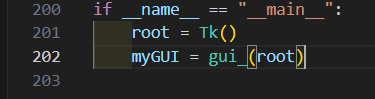
tkinter 库：tkinter 是 Python 的标准 GUI（图形用户界面）库，它提供了一组用于创建图形界面的函数和类。使用 tkinter，你可以在 Python 中创建窗口、按钮、文本框、标签等各种 GUI 元素，并通过事件处理机制响应用户的输入和操作。tkinter 是跨平台的，可以在不同的操作系统上运行，包括 Windows、macOS 和 Linux。

threading 库：threading 是 Python 中用于多线程编程的标准库。它提供了一组函数和类，用于创建和管理线程。通过 threading 库，你可以在 Python 中实现多线程程序，从而可以同时执行多个任务，提高程序的并发性和响应性。

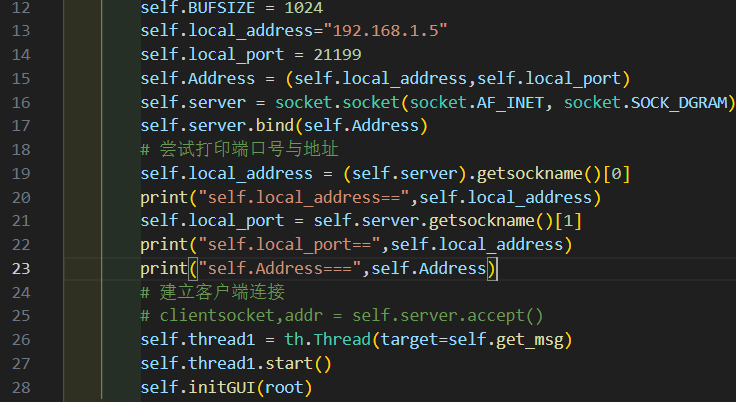
PIL 库：PIL（Python Imaging Library）是一个用于图像处理的强大库。它提供了各种功能，包括图像的加载、保存、调整尺寸、裁剪、旋转、滤镜应用等。PIL 库还可以与 tkinter 库结合使用，用于在 GUI 程序中显示和操作图像。

sqlite3 库：sqlite3 是 Python 的标准库，用于操作 SQLite 数据库。SQLite 是一个轻量级的嵌入式数据库，不需要独立的服务器进程，可以直接在应用程序中使用。sqlite3 库提供了一组函数和类，用于连接、执行 SQL 查询、插入和更新数据等操作。通过 sqlite3 库，你可以在 Python 中使用 SQLite 数据库来存储和管理数据。

这些库在 Python 中广泛使用，每个库都有其独特的功能和用途。它们可以帮助你实现网络通信、创建图形界面、处理图像以及进行数据库操作等任务。



## 4.2 初始化Socket



上述代码片段展示了一个类的初始化方法，其中创建了一个服务器的套接字并绑定到指定的本地地址和端口。同时，还创建了一个线程并启动它，用于处理接收到的消息，并调用了一个名为 `initGUI` 的方法来初始化用户界面。

具体解释如下：

- `self.BUFSIZE = 1024`：定义了接收缓冲区的大小为 1024 字节。

- `self.local\_address = "192.168.1.5"`：指定了服务器绑定的本地 IP 地址。

- `self.local\_port = 21199`：指定了服务器绑定的本地端口号。

- `self.Address = (self.local\_address, self.local\_port)`：创建了一个元组 `Address`，包含要绑定的本地地址和端口。

- `self.server = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)`：创建了一个 UDP 套接字对象 `server`，使用 IPv4 地址族（`socket.AF\_INET`）和数据报传输方式（`socket.SOCK\_DGRAM`）。

- `self.server.bind(self.Address)`：将服务器套接字绑定到指定的本地地址和端口。

- `self.thread1 = th.Thread(target=self.get\_msg)`：创建了一个线程对象 `thread1`，目标函数为 `get\_msg`。

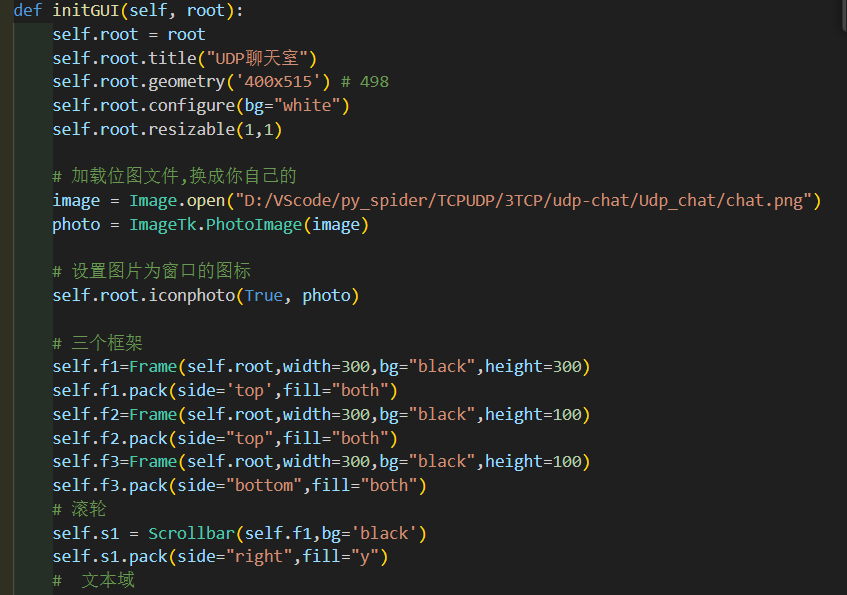
- `self.thread1.start()`：启动线程 `thread1`，开始执行 `get\_msg` 方法。

- `self.initGUI(root)`：调用 `initGUI` 方法来初始化用户界面，传递了 `root` 作为参数。

这段代码的作用是创建一个服务器，绑定到指定的本地地址和端口，并启动一个线程来处理接收到的消息。同时，通过调用 `initGUI` 方法，初始化用户界面。

请注意，代码中还有一行被注释掉的代码 `# clientsocket,addr = self.server.accept()`，该行代码表明了建立客户端连接的部分。然而，由于这是一个 UDP 服务器，它使用数据报传输方式，因此不需要使用 `accept()` 方法来等待客户端连接，而是直接通过 `get\_msg` 方法处理接收到的消息。

## 4.3 初始化GUI



上述代码片段展示了一些与用户界面相关的操作。具体解释如下：

- `image = Image.open("D:/VScode/py\_spider/TCPUDP/3TCP/udp-chat/Udp\_chat/chat.png")`：使用 `Image.open()` 方法加载位图文件，文件路径为 `D:/VScode/py\_spider/TCPUDP/3TCP/udp-chat/Udp\_chat/chat.png`。这行代码假设你有一张名为 `chat.png` 的位图文件，并将其加载到 `image` 变量中。

- `photo = ImageTk.PhotoImage(image)`：使用 `ImageTk.PhotoImage()` 方法创建一个 `PhotoImage` 对象 `photo`，用于在 Tkinter 窗口中显示图片。这行代码将之前加载的图片 `image` 转换为 Tkinter 可接受的图片对象。

- `self.root.iconphoto(True, photo)`：将窗口的图标设置为之前创建的图片 `photo`。通过调用 `iconphoto()` 方法，将 `photo` 设置为窗口的图标。

- `self.f1 = Frame(self.root, width=300, bg="black", height=300)`：创建了一个名为 `f1` 的框架对象，使用 `Frame()` 构造函数。该框架的父窗口是 `self.root`，宽度为 300 像素，高度为 300 像素，背景颜色为黑色。

- `self.f1.pack(side='top', fill="both")`：将 `f1` 框架放置在父窗口中，位于顶部位置，并填充父窗口的剩余空间。`fill="both"` 参数表示该框架将填充父窗口的水平和垂直方向上的空间。

- 类似地，代码中还创建了 `f2` 和 `f3` 两个框架对象，并设置它们的宽度、高度和背景颜色。然后，通过调用 `pack()` 方法将这些框架放置在父窗口中，其中 `f2` 位于顶部位置，`f3` 位于底部位置。

这段代码的作用是加载一张图片作为窗口的图标，并创建了三个框架用于布局界面。每个框架有自己的宽度、高度和背景颜色，并通过 `pack()` 方法将它们放置在父窗口中。

请注意，代码中的文件路径 `D:/VScode/py\_spider/TCPUDP/3TCP/udp-chat/Udp\_chat/chat.png` 是一个示例路径，你需要根据实际情况修改为你自己的图片文件路径。



上述代码片段展示了在用户界面中创建输入框和按钮的操作。具体解释如下：

- `self.e1 = Entry(self.f2, bg='black', width=40, foreground='white', highlightthickness=2, highlightbackground="grey", highlightcolor='white', insertbackground='white', insertwidth=2)`：创建了一个输入框对象 `e1`，父窗口是 `f2` 框架。该输入框的背景颜色为黑色，宽度为 40 个字符，前景（文本）颜色为白色，使用灰色作为边框高亮颜色，插入符号（光标）的颜色为白色，插入符号的宽度为 2。

- `self.e1.pack(side='left', padx=12, pady=5)`：将输入框 `e1` 放置在父窗口中的左侧位置。`padx=12` 和 `pady=5` 参数表示在水平和垂直方向上添加一些内间距（间距）。

- `self.b1 = Button(self.f2, bg='black', fg='white', text='发送', width=6, height=1, command=self.sendMsg)`：创建了一个按钮对象 `b1`，父窗口是 `f2` 框架。该按钮的背景颜色为黑色，文本颜色为白色，显示的文本为 "发送"，宽度为 6 个字符，高度为 1 行。通过 `command=self.sendMsg`，当按钮被点击时，会调用 `self.sendMsg` 方法。

- `self.b1.pack(side='left', pady=5)`：将按钮 `b1` 放置在父窗口中的左侧位置。`pady=5` 参数表示在垂直方向上添加一些内边距（间距）。

- `self.b2 = Button(self.f3, bg='black', fg='white', text='下载', width=6, height=1, command=self.save\_to\_database)`：创建了一个按钮对象 `b2`，父窗口是 `f3` 框架。该按钮的背景颜色为黑色，文本颜色为白色，显示的文本为 "下载"，宽度为 6 个字符，高度为 1 行。通过 `command=self.save\_to\_database`，当按钮被点击时，会调用 `self.save\_to\_database` 方法。

- `self.b2.pack(side='left', pady=5, padx=6)`：将按钮 `b2` 放置在父窗口中的左侧位置。`pady=5` 和 `padx=6` 参数表示在垂直和水平方向上添加一些内边距（间距）。

这段代码的作用是在用户界面中创建一个输入框和两个按钮。输入框用于输入文本，按钮用于执行相应的操作。通过 `pack()` 方法将它们放置在相应的框架中，并设置一些样式和回调函数。

上述代码片段展示了在用户界面中创建多个按钮的操作。具体解释如下：

- `self.b3 = Button(self.f3, bg='black', fg='white', text='护眼模式', width=6, height=1, command=self.change\_bg\_yellowcolor)`：创建了一个按钮对象 `b3`，父窗口是 `f3` 框架。该按钮的背景颜色为黑色，文本颜色为白色，显示的文本为 "护眼模式"，宽度为 6 个字符，高度为 1 行。通过 `command=self.change\_bg\_yellowcolor`，当按钮被点击时，会调用 `self.change\_bg\_yellowcolor` 方法。

- `self.b3.pack(side='left', pady=5, padx=6)`：将按钮 `b3` 放置在父窗口中的左侧位置。`pady=5` 和 `padx=6` 参数表示在垂直和水平方向上添加一些内边距（间距）。

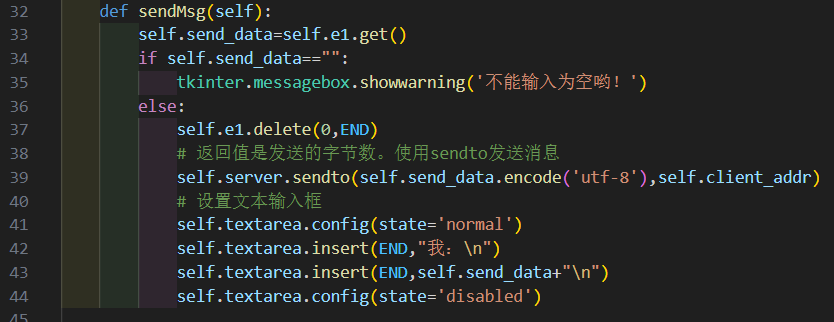
- 类似地，代码中还创建了 `b4` 和 `b5` 两个按钮对象，并设置它们的背景颜色、文本颜色、显示的文本、宽度和高度。并通过 `command` 参数指定按钮点击时的回调函数。

- `self.b4.pack(side='left', pady=5, padx=6)`：将按钮 `b4` 放置在父窗口中的左侧位置。`pady=5` 和 `padx=6` 参数表示在垂直和水平方向上添加一些内边距（间距）。

- `self.b5.pack(side='left', pady=5, padx=6)`：将按钮 `b5` 放置在父窗口中的左侧位置。`pady=5` 和 `padx=6` 参数表示在垂直和水平方向上添加一些内边距（间距）。

这段代码的作用是在用户界面中创建三个按钮。每个按钮都有自己的样式、显示的文本和回调函数。通过 `pack()` 方法将它们放置在 `f3` 框架的左侧位置，并添加一些内边距（间距）。

## 4.4 发送消息



上述代码片段是一个名为 `sendMsg` 的方法，用于处理发送消息的操作。具体解释如下：

- `self.send\_data = self.e1.get()`：获取输入框 `e1` 中的文本内容，并将其存储在 `self.send\_data` 变量中。

- `if self.send\_data == "": tkinter.messagebox.showwarning('不能输入为空哟！')`：检查 `self.send\_data` 是否为空。如果为空，显示一个警告对话框，提示用户不能输入空消息。

- `else:`：如果 `self.send\_data` 不为空，则执行以下操作。

- `self.e1.delete(0, END)`：清空输入框 `e1` 中的文本内容，将其重置为空。

- `self.server.sendto(self.send\_data.encode('utf-8'), self.client\_addr)`：将 `self.send\_data` 编码为 UTF-8 格式，并使用 `self.server` 对象的 `sendto` 方法发送消息给 `self.client\_addr`。这里假设 `self.server` 是一个网络服务器对象，用于发送数据。

- `self.textarea.config(state='normal')`：将文本域 `textarea` 的状态设置为可编辑状态。

- `self.textarea.insert(END, "我：\n")`：在文本域 `textarea` 的末尾插入一行文本 "我："，表示消息发送者为当前用户。

- `self.textarea.insert(END, self.send\_data+"\n")`：在文本域 `textarea` 的末尾插入用户输入的消息内容，并在末尾添加一个换行符。

- `self.textarea.config(state='disabled')`：将文本域 `textarea` 的状态设置为禁用状态，即不可编辑状态。

这段代码的作用是处理用户发送消息的逻辑。它获取用户输入的消息内容，进行验证，如果不为空，则将消息发送给指定的客户端地址。然后，在界面的文本域中显示发送者信息和发送的消息内容，并将文本域设置为只读状态。

## 4.5 接受消息



上述代码片段是一个名为 `get\_msg` 的方法，用于接收消息并处理。具体解释如下：

- `while True:`：进入一个无限循环，用于持续接收消息。

- `try:`：尝试执行以下操作。

- `self.data, self.client\_addr = self.server.recvfrom(self.BUFSIZE)`：使用 `self.server` 对象的 `recvfrom` 方法接收消息，并将接收到的数据和客户端地址存储在 `self.data` 和 `self.client\_addr` 变量中。

- `self.data = self.data.decode('utf-8')`：将接收到的数据使用 UTF-8 解码，以获得原始的文本消息。

- `if self.client\_addr == self.Address: pass`：检查接收到消息的客户端地址是否等于预设的 `self.Address`。如果是相同的地址，则跳过后续操作，不进行处理。

- `else:`：如果接收到消息的客户端地址不是预设的 `self.Address`，则执行以下操作。

- `if self.data == "client-0001":`：检查接收到的消息内容是否为 "client-0001"。如果是，则执行以下操作。

- `self.send1 = "server-0001"`：设置要发送的消息内容为 "server-0001"。

- `self.server.sendto(self.send1.encode('utf-8'), self.client\_addr)`：将要发送的消息内容编码为 UTF-8 格式，并使用 `self.server` 对象的 `sendto` 方法将消息发送给 `self.client\_addr`。

- `self.textarea.config(state='normal')`：将文本域 `textarea` 的状态设置为可编辑状态。

- `self.textarea.insert(END, self.client\_addr[0])`：在文本域 `textarea` 的末尾插入接收消息的客户端地址的第一个元素，即 IP 地址。

- `self.textarea.insert(END, "已连接\n")`：在文本域 `textarea` 的末尾插入一行文本 "已连接"，表示连接已建立。

- `self.textarea.config(state='disabled')`：将文本域 `textarea` 的状态设置为禁用状态，即不可编辑状态。

- `else:`：如果接收到的消息内容不是 "client-0001"，则执行以下操作。

- `self.data = self.data + "\n"`：在接收到的消息内容末尾添加一个换行符。

- `self.textarea.config(state='normal')`：将文本域 `textarea` 的状态设置为可编辑状态。

- `self.textarea.insert(END, self.client\_addr[0])`：在文本域 `textarea` 的末尾插入接收消息的客户端地址的第一个元素，即 IP 地址。

- `self.textarea.insert(END, "：\n")`：在文本域 `textarea` 的末尾插入一行文本 "："，表示消息发送者。

- `self.textarea.insert(END, self.data)`：在文本域 `textarea` 的末尾插入接收到的消息内容。

- `self.textarea.config(state='disabled')`：将文本域 `textarea` 的状态设置为禁用状态，即不可编辑状态。

- `except Exception as e:`：捕获任何异常，并将异常信息存储在变量 `e` 中。

- `print(e)`：打印异常信息。

这段代码的作用是持续接收消息，并根据接收到的消息内容执行相应的操作。如果接收到的消息是 "client-0001"，则发送一条回复消息，并在界面的文本域中显示已连接的信息。如果接收到的消息不是 "client-0001"，则将消息内容显示在界面的文本域中，同时显示消息发送者的 IP 地址。

## 4.6 护眼模式与白黑模式



上述代码较为简单，以护眼模式为例，

上述代码是一个名为 `change\_bg\_yellowcolor` 的方法，用于切换护眼模式。具体解释如下：

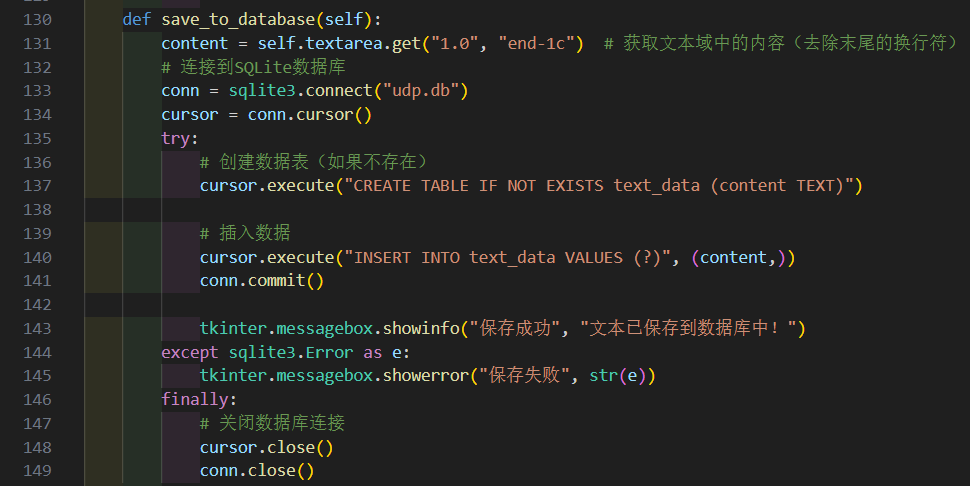
- `self.textarea.configure(bg='#FFFFCC', fg="black")`：使用 `self.textarea` 对象的 `configure` 方法来配置文本域的属性。

- `bg='#FFFFCC'`：设置文本域的背景颜色为淡黄色。`#FFFFCC` 是一个十六进制颜色码，表示淡黄色。

- `fg="black"`：设置文本域的前景颜色为黑色。`"black"` 是预定义的颜色名称，表示黑色。

这段代码的作用是将文本域的背景颜色设置为淡黄色（#FFFFCC），以实现护眼模式。同时，将文本域的前景颜色设置为黑色，以确保文本内容与背景对比明显。

## 4.7 下载数据到数据库



上述代码是一个名为 `save\_to\_database` 的方法，用于将文本域中的内容保存到 SQLite 数据库中。具体解释如下：

- `content = self.textarea.get("1.0", "end-1c")`：使用 `self.textarea` 对象的 `get` 方法获取文本域中的内容。参数 `"1.0"` 表示从第一行第一列开始获取，`"end-1c"` 表示获取到倒数第一个字符前的位置，即去除末尾的换行符。

- `conn = sqlite3.connect("udp.db")`：连接到 SQLite 数据库文件 "udp.db"。如果该文件不存在，则会被创建。

- `cursor = conn.cursor()`：创建一个游标对象，用于执行 SQL 语句。

- `cursor.execute("CREATE TABLE IF NOT EXISTS text\_data (content TEXT)")`：执行 SQL 语句，创建一个名为 "text\_data" 的数据表。如果该表已存在，则不会重新创建。

- `cursor.execute("INSERT INTO text\_data VALUES (?)", (content,))`：执行 SQL 语句，将获取到的文本内容插入到 "text\_data" 数据表中。使用占位符 "?" 来表示要插入的值，`(content,)` 是一个包含单个元素的元组，表示要插入的具体值。

- `conn.commit()`：提交事务，将插入的数据持久化到数据库中。

- `tkinter.messagebox.showinfo("保存成功", "文本已保存到数据库中！")`：显示一个信息对话框，提示用户保存成功的消息。

- `except sqlite3.Error as e:`：捕获 SQLite 数据库操作过程中可能发生的异常，并将异常信息存储在变量 `e` 中。

- `tkinter.messagebox.showerror("保存失败", str(e))`：显示一个错误对话框，提示用户保存失败的消息，并将异常信息作为错误消息显示。

- `finally:`：无论是否发生异常，都会执行以下操作。

- `cursor.close()`：关闭游标。

- `conn.close()`：关闭数据库连接。

这段代码的作用是将文本域中的内容保存到 SQLite 数据库中的 "text\_data" 表中。首先，连接到数据库，然后创建数据表（如果不存在），接着将获取到的文本内容插入到表中。如果保存成功，显示保存成功的消息，否则显示保存失败的消息。

请注意，这段代码使用了 `sqlite3` 模块来操作 SQLite 数据库。确保你已经正确导入了 `sqlite3` 模块，并且已经安装了 SQLite 数据库。

## 五、总结

在本次对话中，我们讨论了如何使用 UDP 协议设计 Server 和 Client，以实现多用户同时聊天通信，并画出 GUI 界面。下面是一个详细总结：

1. UDP 协议：

- UDP 是用户数据报协议，它提供了一种无连接、不可靠的数据传输方式。

- UDP 适用于需要快速传输数据、对可靠性要求不高的场景，如实时通信等。

- UDP 不保证数据的顺序和可靠性，但具有较低的延迟和网络开销。

2. Server 设计：

- Server 端是一个中央服务器，负责接收来自不同 Client 的消息并进行转发。

- Server 端需要使用 socket 编程来建立 UDP 连接，并监听特定的端口。

- Server 端需要维护一个用户列表，以存储已连接的 Client 信息。

- Server 端需要通过广播或多播方式将消息发送给所有在线的用户。

- Server 端需要支持多用户同时聊天的功能。

3. Client 设计：

- Client 端是用户使用的客户端程序，负责与 Server 进行通信。

- Client 端需要使用 socket 编程来建立与 Server 的 UDP 连接。

- Client 端需要提供用户界面(GUI)，以便用户输入消息和查看聊天记录。

- Client 端需要支持发送消息给所有在线用户。

- Client 端需要接收和显示来自 Server 的消息，并将其显示在用户界面上。

4. GUI 界面设计：

- GUI 界面可以使用图形界面库（如 tkinter）来实现。

- GUI 界面应该包含输入框和发送按钮，以便用户输入消息并发送。

- GUI 界面应该有一个消息显示区域，用于显示收到的消息和用户发送的消息。

5. 实现用户聊天通信的关键点：

- Server 端需要使用 socket 编程来处理 UDP 连接请求和消息传递，以实现多用户同时聊天的能力。

- Server 端需要维护一个用户列表，以跟踪在线用户的状态和位置。

- Server 端需要处理来自 Client 的消息，并将其转发给所有在线用户。

- Client 端需要使用 socket 编程来建立与 Server 的 UDP 连接，发送和接收消息。

- Client 端需要处理来自 Server 的消息，并将其显示在 GUI 界面上。

在设计 UDP 协议的多用户聊天系统时，需要注意数据包的顺序和可靠性问题，因为 UDP 不提供这些功能。另外，设计合适的消息格式和协议，以便 Server 和 Client 能够正确解析和处理消息。

## 六、参考文献

1. UDP 协议相关文献：

- RFC 768: User Datagram Protocol (UDP)：https://tools.ietf.org/html/rfc768

- TCP/IP Illustrated, Volume 1: The Protocols (Chapter 8) by W. Richard Stevens：该书第一卷的第8章介绍了 UDP 协议的详细内容。

2. 多用户聊天系统设计文献：

- Distributed Systems: Concepts and Design by George Coulouris, Jean Dollimore, and Tim Kindberg：该书第15章介绍了多用户聊天系统的设计原则和实现方法。

- Building Scalable and High-Performance Java Web Applications Using J2EE Technology by Greg Barish：该书的第7章提供了关于多用户聊天系统设计和实现的详细指南。

3. Python socket 编程文档：

- Python 官方文档：socket 模块：https://docs.python.org/3/library/socket.html

- Real Python Tutorial: Python Socket Programming: https://realpython.com/python-sockets/

4. GUI 设计和编程资源：

- Python 官方文档：tkinter 模块：https://docs.python.org/3/library/tkinter.html

- PyQt 官方文档：https://www.riverbankcomputing.com/static/Docs/PyQt5/

- wxPython 官方文档：https://docs.wxpython.org/