How to write a chat program (two clients chat with each other) with UDP?

由于大多数客户端都使用NAT连接，没有公共IP地址，因此无法直接接受来自互联网未经请求的连接。
因此，可以利用一个公网服务器作为中继来转发消息，两个客服端都只与该服务器通信，服务器负责将消息转发给正确的接收方。

工作流程

1. **启动与注册**：

   - 服务器在公网 IP 地址和端口上启动并持续监听。
   - 客户端 A 启动。它向服务器发送一条“注册”消息。
   - 服务器收到后，记录下客户端 A 的公网 IP 地址和端口号（可以从收到的数据包中获取）。
   - 客户端 B 启动，并执行同样的操作。现在，服务器知道了 A 和 B 的地址。

2. **消息发送与中继**：

   - 客户端 A 的用户输入一条消息，想要发送给客户端 B。
   - 客户端 A 将这条消息打包（比如，格式化为 `SENDTO:B: Hello!`），然后发送给**服务器**。
   - 服务器接收到来自 A 的数据包，解析内容，找到接收方 B。
   - 服务器在它的注册列表中查找 B 的 IP 地址和端口号。
   - 服务器将消息 `Hello!` **转发**给客户端 B 的地址。

3. **消息接收**：

   - 客户端 B 一直在监听来自网络的 UDP 数据包。
   - 它收到了服务器转发过来的消息，并展示出来。

4. **双向通信**：

   - 当 B 回复 A 时，流程完全一样，只是方向相反：`B -> 服务器 -> A`。

组件设计

为了实现该聊天程序，需要设计三个核心组件：服务器、客户端和通信协议。

1. 服务器 (Server)

服务器作为系统中枢，负责接收所有客户端的连接和消息，并将其转发给正确的目标客户端。

核心功能:

- **监听端口**：在一个固定的公网 IP 地址和端口上监听传入的 UDP 数据包。
- **客户端注册**：可以维护一个在线客户端列表。当客户端首次连接时，服务器记录其唯一标识符（如用户名）及其网络地址（IP 和端口）。
  - 可以使用哈希表（或字典、Map）来存储客户端信息，键为客户端的唯一标识符，值为其 IP 地址和端口号。例如：

```
clients = {
    "Annie": {"ip": "123.45.67.89", "port": 12345},
    "Bob":   {"ip": "98.76.54.32", "port": 54321}
}
```

- **消息中继**：接收来自一个客户端的消息，解析出目标接收方，然后从注册列表中查找目标客户端的地址，并将消息转发过去。
- **心跳机制**：UDP 是无连接的，服务器无法感知客户端是否下线。因此，可以实现一个心跳机制来检测客户端是否仍然在线。
  - **服务器端**：为每个客户端维护一个最后活跃时间戳。服务器会定期检查，如果某个客户端长时间（例如超过60秒）没有发送任何消息（包括心跳包），就将其从在线列表中移除。
  - **客户端**：客户端需要定期（例如每30秒）向服务器发送一个心跳包，以证明自己仍在线。

2. 客户端 (Client)

客户端是用户交互的界面，负责发送和接收消息。

核心功能:

- **注册**：启动时，向服务器发送一条注册消息，告诉服务器“我上线了”。

- **发送心跳**：在后台定期向服务器发送心跳包，以维持在线状态。
- **发送消息**：
  - 从用户那里获取输入（通过命令行或图形界面）。
  - 将用户输入的消息按照预定义的协议格式打包（例如，指明接收者）。
  - 将打包后的消息发送到服务器的地址。
- **接收消息**：
  - 在后台持续监听来自服务器的数据包。
  - 收到数据包后，解析并向用户显示消息内容。

并发处理：客户端需要能够同时处理用户输入（发送消息）和网络输入（接收消息）。这可以利用多线程或异步 I/O 来实现。

- 一个线程/任务负责监听和处理用户输入。
- 另一个线程/任务负责监听网络端口并接收来自服务器的消息。
- **处理服务器通知**：解析并响应来自服务器的特殊通知，例如，当它因为超时被移除后，服务器会发回错误通知，客户端收到后自动重新注册。

3. 通信协议 (Protocol)

为了让客户端和服务器能够相互理解消息含义，定义一套简单的文本协议。

比如:

- **注册消息 (C -> S)**：客户端启动后发送给服务器。
  `REGISTER: <Username>`
  e.g.: `REGISTER: Bob`
- **心跳消息 (C -> S)**：客户端定期发送以维持在线状态。
  `HEARTBEAT: <Username>`
  e.g.: `HEARTBEAT: Bob`
- **发送消息 (C -> S)**：客户端发送聊天消息给服务器。
  `SENDTO:<RecipientUsername>: <Message>`
  e.g.: `SENDTO:Annie: Annie are you ok?`
- **转发消息 (S -> C)**：服务器将消息转发给目标客户端。
  `FROM:<SenderUsername>: <Message>`
  e.g.: `FROM:Bob: Annie are you ok?`
- **错误/通知消息 (S -> C)**：服务器发送通知或错误信息给客户端。
  `INFO: <Message>`

e.g.: `INFO: User Annie isn't online because she's been hit by a smooth criminal.`

e.g.: `INFO: You are not registered, please register again.`

实际上的协议可能需要更多的信息，同时需要更多的设计，比如消息队列处理等等。