Hướng Dẫn Lập Trình Ứng Dụng Chat Client-Server với Python, Socket và Tkinter

Tác giả: Đặng Kim Thi
Ngày viết: 20/03/2025

Mục Lục

Giới Thiệu và Mục Tiêu

Tôi là Đặng Kim Thi, và trong bài viết này, tôi sẽ hướng dẫn bạn cách lập trình một **ứng dụng chat client-server** đơn giản bằng Python. Ứng dụng này cho phép người dùng gửi và nhận tin nhắn qua mạng bằng giao diện đồ họa (GUI).

Mục tiêu:

- Hiểu cách sử dụng `socket` để thiết lập kết nối mạng giữa server và client.
- Sử dụng `Tkinter` để tạo giao diện người dùng cho client.
- Áp dụng `threading` để xử lý nhận tin nhắn mà không làm treo giao diện.
- Giải thích chi tiết từng dòng code để người mới học lập trình có thể hiểu rõ.

Công Nghệ Sử Dụng

Dưới đây là các công nghệ chính được sử dụng trong ứng dụng này, cùng với giải thích chi tiết:

Socket

- **Là gì?**: `Socket` là một module chuẩn của Python dùng để lập trình mạng. Nó cung cấp giao diện để tạo kết nối giữa các máy tính qua giao thức TCP hoặc UDP.
- **Tại sao dùng?**: Trong ứng dụng này, `socket` được dùng để thiết lập kết nối TCP giữa server và client, đảm bảo dữ liệu được truyền đáng tin cậy và theo thứ tự.
- **Cách hoạt động**: Server lắng nghe kết nối, client kết nối đến server, sau đó hai bên gửi/nhận dữ liệu qua socket.

Tkinter

- **Là gì?**: `Tkinter` là thư viện chuẩn của Python để tạo giao diện đồ họa (GUI). Nó dựa trên toolkit Tcl/Tk.
- **Tại sao dùng?**: Tôi chọn `Tkinter` vì nó đơn giản, dễ học và tích hợp sẵn trong Python, phù hợp để tạo giao diện nhập liệu và hiển thị tin nhắn cho client.
- **Cách hoạt động**: Tạo các thành phần như ô nhập liệu (`Entry`), vùng văn bản (`Text`), và nút (`Button`) để người dùng tương tác.

Threading

- **Là gì?**: `Threading` là module của Python hỗ trợ lập trình đa luồng, cho phép chạy nhiều tác vụ đồng thời.
- **Tại sao dùng?**: Tôi dùng `threading` để nhận tin nhắn từ server trên một luồng riêng, tránh làm treo giao diện Tkinter khi chờ dữ liệu.
- **Cách hoạt động**: Một luồng chạy hàm nhận tin nhắn, trong khi luồng chính chạy giao diện GUI.

Phần 1: Lập Trình Server

Code của Server

```
import socket

# Thiết lập thông tin server
host = '127.0.0.1'  # Địa chỉ IP của server (localhost)
port = 12345        # Cổng mà server sẽ lắng nghe
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Liên kết và lắng nghe kết nối
server_socket.bind((host, port))
server_socket.listen(1)

print("Server đang lắng nghe tại", host, ":", port)

# Chấp nhận kết nối từ client
client_socket, addr = server_socket.accept()
print("Đã kết nối với", addr)

# Vòng lặp để nhận và gửi tin nhắn
while True:
    data = client_socket.recv(1024).decode()
    if not data:
```

```
            break
        print("Client gửi:", data)
        client_socket.send("Tin nhắn đã được nhận!".encode())

    # Đóng kết nối
    client_socket.close()
    server_socket.close()
```

Giải Thích Chi Tiết Code Server

1. `import socket`
    - Nhập module `socket` để xử lý kết nối mạng.
2. `host = '127.0.0.1'`
    - Địa chỉ IP localhost, nơi server chạy.
3. `port = 12345`
    - Cổng server lắng nghe, chọn số trong dải 1024-65535.
4. `server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`
    - Tạo socket TCP với IPv4.
5. `server_socket.bind((host, port))`
    - Liên kết socket với địa chỉ và cổng.
6. `server_socket.listen(1)`
    - Lắng nghe tối đa 1 kết nối.
7. `print("Server đang lắng nghe tại", host, ":", port)`
    - Thông báo server sẵn sàng.
8. `client_socket, addr = server_socket.accept()`
    - Chấp nhận kết nối từ client, trả về socket và địa chỉ client.
9. `print("Đã kết nối với", addr)`
    - Thông báo kết nối thành công.
10. `while True:`
    - Vòng lặp nhận tin nhắn liên tục.
11. `data = client_socket.recv(1024).decode()`
    - Nhận tối đa 1024 byte dữ liệu, chuyển thành chuỗi.
12. `if not data:`

- Thoát nếu không còn dữ liệu (client ngắt).

13. `break`
    - Thoát vòng lặp.
14. `print("Client gửi:", data)`
    - Hiển thị tin nhắn từ client.
15. `client_socket.send("Tin nhắn đã được nhận!".encode())`
    - Gửi phản hồi về client.
16. `client_socket.close()`
    - Đóng kết nối với client.
17. `server_socket.close()`
    - Đóng socket server.

Cách Chạy Server

- Lưu code vào file `server.py`.
- Chạy lệnh: `python server.py` trong terminal.

Phần 2: Lập Trình Client với Tkinter

Code của Client

```
import socket
import tkinter as tk
from threading import Thread

# Hàm gửi tin nhắn
def send_message():
    message = entry.get()
    if message:
        client_socket.send(message.encode())
        text_area.insert(tk.END, "Bạn: " + message + "\n")
        entry.delete(0, tk.END)

# Hàm nhận tin nhắn từ server
def receive_messages():
    while True:
        try:
            response = client_socket.recv(1024).decode()
            text_area.insert(tk.END, "Server: " + response + "\n")
        except:
            break
```

```
# Thiết lập socket client
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = '127.0.0.1'
port = 12345
client_socket.connect((host, port))

# Thiết lập giao diện Tkinter
window = tk.Tk()
window.title("Chat Client - Đặng Kim Thi")

text_area = tk.Text(window, height=10, width=50)
text_area.pack(pady=10)

entry = tk.Entry(window, width=40)
entry.pack(pady=5)

send_button = tk.Button(window, text="Gửi", command=send_message)
send_button.pack(pady=5)

# Khởi động luồng nhận tin nhắn
receive_thread = Thread(target=receive_messages)
receive_thread.daemon = True
receive_thread.start()

# Chạy giao diện
window.mainloop()

# Đóng socket khi thoát
client_socket.close()
```

Giải Thích Chi Tiết Code Client

1. **`import socket`**
    - Nhập module mạng.
2. **`import tkinter as tk`**
    - Nhập Tkinter với bí danh `tk`.
3. **`from threading import Thread`**
    - Nhập `Thread` để chạy đa luồng.
4. **`def send_message():`**
    - Hàm gửi tin nhắn khi nhấn nút.
5. **`message = entry.get()`**
    - Lấy nội dung từ ô nhập liệu.
6. **`if message:`**

    - Kiểm tra nội dung không rỗng.
7. `client_socket.send(message.encode())`
    - Gửi tin nhắn qua socket.
8. `text_area.insert(tk.END, "Bạn: " + message + "\n")`
    - Hiển thị tin nhắn trong vùng văn bản.
9. `entry.delete(0, tk.END)`
    - Xóa ô nhập liệu.
10. `def receive_messages():`
    - Hàm nhận tin nhắn từ server.
11. `while True:`
    - Vòng lặp nhận liên tục.
12. `try:`
    - Xử lý lỗi nếu kết nối ngắt.
13. `response = client_socket.recv(1024).decode()`
    - Nhận và giải mã dữ liệu.
14. `text_area.insert(tk.END, "Server: " + response + "\n")`
    - Hiển thị phản hồi từ server.
15. `except:`
    - Thoát nếu có lỗi.
16. `break`
    - Thoát vòng lặp.
17. `client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`
    - Tạo socket client.
18. `host = '127.0.0.1'`
    - Địa chỉ server.
19. `port = 12345`
    - Cổng server.
20. `client_socket.connect((host, port))`

    - Kết nối đến server.
21. `window = tk.Tk()`
    - Tạo cửa sổ GUI.
22. `window.title("Chat Client – Đặng Kim Thi")`
    - Đặt tiêu đề cửa sổ.
23. `text_area = tk.Text(window, height=10, width=50)`
    - Tạo vùng văn bản.
24. `text_area.pack(pady=10)`
    - Sắp xếp vùng văn bản.
25. `entry = tk.Entry(window, width=40)`
    - Tạo ô nhập liệu.
26. `entry.pack(pady=5)`
    - Sắp xếp ô nhập liệu.
27. `send_button = tk.Button(window, text="Gửi", command=send_message)`
    - Tạo nút gửi.
28. `send_button.pack(pady=5)`
    - Sắp xếp nút.
29. `receive_thread = Thread(target=receive_messages)`
    - Tạo luồng nhận tin nhắn.
30. `receive_thread.daemon = True`
    - Đặt luồng thành daemon.
31. `receive_thread.start()`
    - Khởi động luồng.
32. `window.mainloop()`
    - Chạy vòng lặp GUI.
33. `client_socket.close()`
    - Đóng socket khi thoát.

Cách Chạy Client

- Lưu code vào file `client.py`.
- Chạy server trước, sau đó chạy `python client.py`.

Kỹ Thuật Sử Dụng

- **Socket Programming**: Dùng TCP để truyền dữ liệu.
- **Multithreading**: Tách nhận tin nhắn ra luồng riêng.
- **Tkinter GUI**: Tạo giao diện đơn giản.
- **Xử lý dữ liệu**: Chuyển đổi string-byte bằng `encode()`/`decode()`.

Hướng Dẫn Cách Làm

1. **Chuẩn bị**: Cài Python.
2. **Viết Server**: Lưu và chạy `server.py`.
3. **Viết Client**: Lưu và chạy `client.py`.
4. **Thử nghiệm**: Gửi tin nhắn từ client, xem phản hồi.

Kết Luận

Bài viết này, do tôi - Đặng Kim Thi - thực hiện, đã hướng dẫn bạn từng bước xây dựng một ứng dụng chat cơ bản. Hy vọng bạn hiểu rõ cách dùng `socket`, `Tkinter`, và `threading`. Hãy thử mở rộng ứng dụng với các tính năng mới và liên hệ nếu cần hỗ trợ!