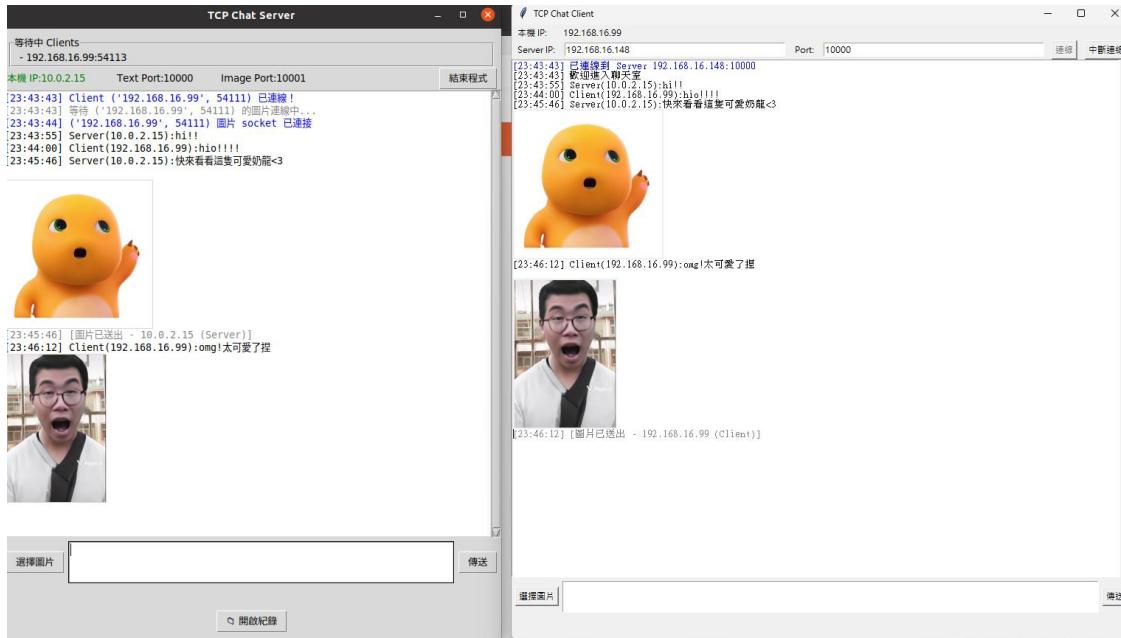


# 進階電網作業 - TCP 聊天室程式說明文件

S1154005-李育林(配分 100%)



## 一、程式功能說明

本程式為一個基於 TCP socket 開發的圖文聊天室系統，採用 Tkinter 實作 GUI 並搭配 Pillow 函式庫呈現圖片，分為 Server 與 Client 端，基本功能包含：

| 功能項目                     | 功能說明                                    |
|--------------------------|---|
| GUI 控制介面                 | GUI 介面分為連線資訊區、訊息顯示區與訊息輸入控制區，支援視窗大自適應調整。 |
| Server 與 Client 一對一連線聊天室 | Server 端接受與單一 Client 連線進行聊天，並同步顯示圖文內容   |
| Server/Client 互相圖文傳輸     | 支援傳送與接收文字訊息與圖片檔案，且可支援同時傳輸文字與圖片          |
| Server/Client 中斷連線       | 連線成功後雙方皆可在聊天過程中隨時中斷連線，並會以訊息提示           |
| 訊息著色                     | 錯誤 / 系統 / 一般訊息以不同顏色區分                   |

## 額外進階功能包含

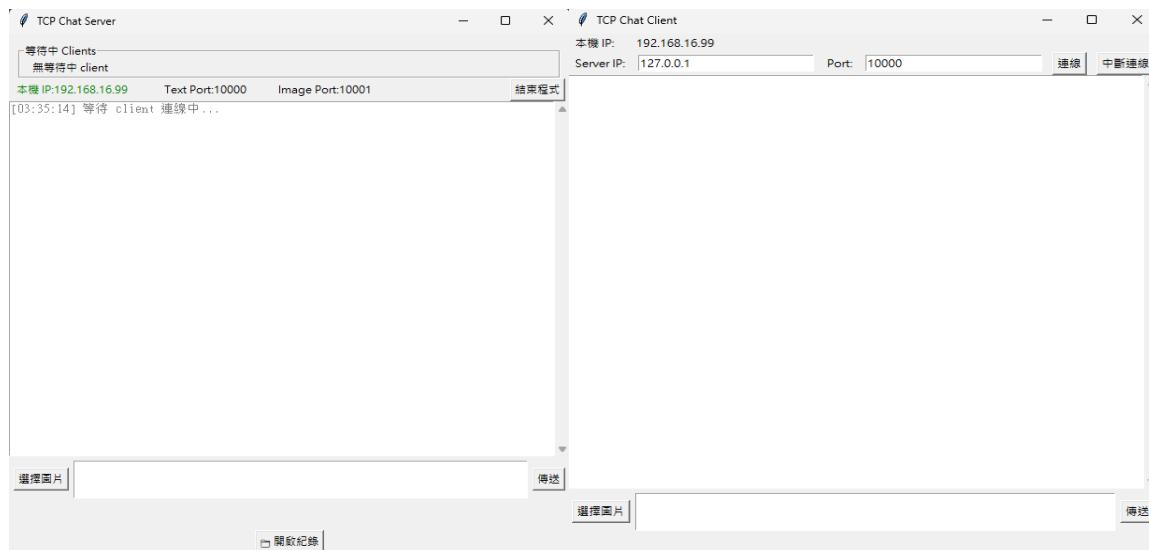
| 功能項目                    | 功能說明   |
|-------------------------|--|
| Server 多個 Client 連線等待機制 | Server 可同時接受多位 Client 嘗試連線，多 Client 嘗試連線會進入排隊，僅允許一人同時與 Server 溝通                     |
| Client 排隊等待機制           | 若當前已有 Client 連線，後續連線會進入佇列，自動獲得序號與等待提示。且 Client 若中途斷線，會自動從佇列中移除；前一位 Client 離線後下一位自動接入 |
| 時間戳記                    | 所有訊息顯示與紀錄皆加上時間標籤   |
| 聊天紀錄保存                  | 聊天訊息自動儲存至 chat_logs 資料夾，Server 端可一鍵開啟資料夾   |
| 圖片預覽                    | 使用者點擊聊天窗內的圖片可放大檢視  |

# 基本程式功能說明

## ✧ GUI 控制介面

Server 和 Client 可透過 GUI 介面進行連線、查看聊天訊息以及送出圖片/文字訊息，畫面分為「等待列表」(Server 端 Only)、「連線資訊」、「訊息框」、「輸入區」，並支援圖片預覽與視窗大小自調整。

**實作方式:** 使用 tkinter 分別建立上中下三層 Frame 並使用函式內的各個控制組件建立出聊天訊息框及控制按鈕的介面



▲Server 端之 GUI 介面

▲Client 端之 GUI 介面

## 相關程式碼(Server 端):

```
# Server GUI 画面建立
def setup_gui(self):
    self.window = tk.Tk()
    self.window.title("TCP Chat Server")
    self.window.geometry("600x600")

    self.window.grid_rowconfigure(2, weight=1)
    self.window.grid_columnconfigure(0, weight=1)

    # 最上方顯示waiting frame
    self.waiting_frame = tk.LabelFrame(self.window, parameter, sticky="s")
    self.waiting_frame.grid(row=0, column=0, sticky="ew", padx=10, pady=(10, 0))
    self.waiting_label = tk.Label(self.waiting_frame, text="無等待中 client")
    self.waiting_label.pack(anchor="w", padx=10)

    # top_frame: 連線資訊與控制
    top_frame = tk.Frame(self.window)
    top_frame.grid(row=1, column=0, sticky="ew")

    for i in range(7):
        top_frame.grid_columnconfigure(i, weight=1)

        tk.Label(top_frame, text=f"本機 IP:{self.local_ip}", fg="green").grid(row=0, column=0, sticky="w", padx=5)
        tk.Label(top_frame, text=f"Text Port:{self.TEXT_PORT}").grid(row=0, column=1, sticky="w")
        tk.Label(top_frame, text=f"Image Port:{self.IMAGE_PORT}").grid(row=0, column=2, sticky="w")
        tk.Button(top_frame, text="結束程式", command=self.close_server).grid(row=0, column=6, sticky="e", padx=5)
```

```

# middle_frame: 訊息框
middle_frame = tk.Frame(self.window)
middle_frame.grid(row=2, column=0, sticky="nsew")
middle_frame.grid_rowconfigure(0, weight=1)
middle_frame.grid_columnconfigure(0, weight=1)

self.log_text = ScrolledText(middle_frame, width=50, height=20)
self.log_text.grid(row=0, column=0, sticky="nsew")

# bottom_frame: 傳送訊息
bottom_frame = tk.Frame(self.window)
bottom_frame.grid(row=3, column=0, sticky="ew", pady=5)
bottom_frame.grid_columnconfigure(1, weight=1)

tk.Button(bottom_frame, text="選擇圖片", command=self.select_image).grid(row=0, column=0, padx=5)
self.input_text = tk.Text(bottom_frame, height=3)
self.input_text.grid(row=0, column=1, sticky="ew")
tk.Button(bottom_frame, text="傳送", command=self.send_message).grid(row=0, column=2, padx=5)

self.img_label = tk.Label(self.window)
self.img_label.grid(row=4, column=0, pady=5)

# 開啟連天記錄存檔的資料夾
tk.Button(self.window, text="📁 開啟紀錄", command=self.open_log_folder).grid(row=5, column=0, pady=(0, 10))

```

## 相關程式碼(Client 端):

```

# Client GUI畫面建立
def setup_gui(self):
    self.window = tk.Tk()
    self.window.title("TCP Chat Client")
    self.window.geometry("600x600")

    self.window.grid_rowconfigure(1, weight=1)
    self.window.grid_columnconfigure(0, weight=1)

    # 上層：顯示本機 IP，輸入 Server IP 與 Port
    top_frame = tk.Frame(self.window)
    top_frame.grid(row=0, column=0, sticky="ew")
    top_frame.grid_columnconfigure(1, weight=1)
    top_frame.grid_columnconfigure(3, weight=1)

    tk.Label(top_frame, text="本機 IP:").grid(row=0, column=0, sticky="w", padx=5)
    tk.Label(top_frame, text=self.local_ip).grid(row=0, column=1, sticky="w")

    tk.Label(top_frame, text="Server IP:").grid(row=1, column=0, sticky="w", padx=5)
    self.server_ip_entry = tk.Entry(top_frame)
    self.server_ip_entry.insert(0, "127.0.0.1")
    self.server_ip_entry.grid(row=1, column=1, sticky="ew", padx=5)

    tk.Label(top_frame, text="Port:").grid(row=1, column=2, sticky="w", padx=5)
    self.server_port_entry = tk.Entry(top_frame)
    self.server_port_entry.insert(0, "10000")
    self.server_port_entry.grid(row=1, column=3, sticky="ew", padx=5)

    self.connect_button = tk.Button(top_frame, text="連線", command=self.connect)
    self.connect_button.grid(row=1, column=4, padx=5)
    tk.Button(top_frame, text="中斷連線", command=self.disconnect).grid(row=1, column=5, padx=5)

    self.server_ip = '' # 清除舊值
    self.server_text_port = 10000

# 中間訊息視窗
self.log_text = ScrolledText(self.window, width=50, height=20)
self.log_text.grid(row=1, column=0, sticky="nsew")

# 下層輸入與傳送
bottom_frame = tk.Frame(self.window)
bottom_frame.grid(row=2, column=0, sticky="ew", pady=5)
bottom_frame.grid_columnconfigure(1, weight=1)

tk.Button(bottom_frame, text="選擇圖片", command=self.select_image).grid(row=0, column=0, padx=5)
self.input_text = tk.Text(bottom_frame, height=3)
self.input_text.grid(row=0, column=1, sticky="ew")
tk.Button(bottom_frame, text="傳送", command=self.send_message).grid(row=0, column=2, padx=5)

self.img_label = tk.Label(self.window)
self.img_label.grid(row=3, column=0, pady=5)

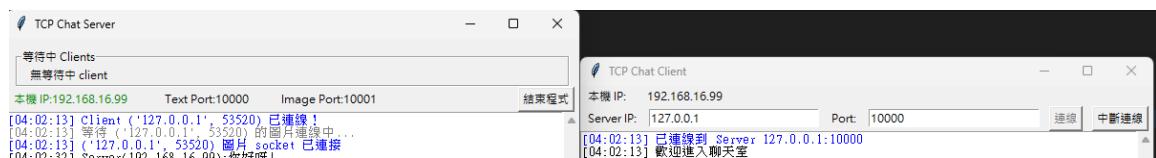
```

## ✧ Server與Client一對一連線聊天室

連線聊天過程，Server 端維持一對一聊天連線模型。Client 按下連線按鈕和 Server 建立連線後，雙方可即時傳送圖文訊息。所有訊息將即時顯示於聊天框 ( **ScrolledText** ) 中，並附上發送者身份與時間戳記。文字訊息直接插入，圖片訊息則以縮圖顯示。圖文可合併顯示，避免重複傳送者標頭，維持聊天紀錄整潔可讀。

### 實作方式：

Server 開啟時會等待 Client 來建立連線，當 Client 成功連上 Server 即開始一對一聊天程式。每則訊息會自動加上 **Server(IP)** 或 **Client(IP)** 開頭 + 時間戳記，例：  
[14:23:10] Server(192.168.1.10):。接收/傳送圖片時使用 **log\_image()** 建立圖片元件，插入於訊息框。接收/傳送文字時使用 **log()** 同步插入於訊息框。



▲雙方建立連線成功後會跳出提示訊息，可以開始進行聊天



▲圖片/文字送出後，會同步顯示於雙方訊息框內，達到一對一同步對話

## 相關程式碼:

```
threading.Thread(target=self.start_text_server, daemon=True).start() # 初始化socket監聽
# 當client連入時，建立與client端的文字訊息傳輸連線
def start_text_server(self):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((self.HOST, self.TEXT_PORT))
    server_socket.listen(5)
    self.log("等待 client 連線中...\n", tag="system")
while True:
    conn, addr = server_socket.accept()
    threading.Thread(target=handle_client, args=(conn, addr), daemon=True).start()

def handle_client(conn, addr):
    if self.text_conn is None:
        self.text_conn = conn
        self.client_addr = addr
        self.log_text.delete("0.0", tk.END) # 新連線清空聊天紀錄
        self.log(f"Client {addr} 已連線!\n", tag="info")
        self.text_conn.sendall(len("歡迎進入聊天室\n".encode())).to_bytes(4, 'big') + "歡迎進入聊天室\n".encode()
        threading.Thread(target=receive_text, daemon=True).start()
        threading.Thread(target=self.start_image_server, args=(addr,), daemon=True).start()
    else:
        identifier = f"{addr[0]}:{addr[1]}"
        self.waiting_clients.put((conn, addr))
        self.waiting_addrs.append(identifier)
        self.update_waiting_label()
        pos = self.waiting_clients.qsize()
        try:
            msg = f"您是第 {pos} 位等待中，請稍候...\n".encode()
            conn.sendall((len(msg).to_bytes(4, 'big') + msg))
        except:
            conn.close()
```

▲Server 端在開啟時就會建立 thread 來監聽 Client 端的連線，當有連線時就會去做 `handle_client()` 的連線機制處理

```
# client連線按鈕操作(嘗試和輸入欄位位址之Server IP和Port連線)
def connect(self):
    self.server_ip = self.server_ip_entry.get().strip()
    try:
        self.server_text_port = int(self.server_port_entry.get())
    except:
        self.log("[錯誤] 請輸入有效的 Port 編號\n", tag="error")
    return
    try:
        self.connect_button.config(state="disabled")
        self.text_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.text_socket.connect((self.server_ip, self.server_text_port))

        self.log(f"已連線到 Server {self.server_ip}:{self.server_text_port}\n", tag="info")
        threading.Thread(target=receive_text, daemon=True).start()
    except Exception as e:
        self.log(f"[錯誤] 無法連線到 Server: {e}\n", tag="error")
```

▲Client 端按下連線按鈕後會去嘗試 connect 輸入欄內的 server IP 及 port

```
# 於聊天框內顯示訊息，透過tag區分顏色
def log(self, msg, tag=None):
    # 記錄時間
    now = datetime.now().strftime("[%H:%M:%S]")
    msg = f"[{now}] {msg}"

    if tag and tag not in self.log_text.tag_names():
        if tag == "error":
            self.log_text.tag_configure(tag, foreground="red")
        elif tag == "info":
            self.log_text.tag_configure(tag, foreground="blue")
        elif tag == "system":
            self.log_text.tag_configure(tag, foreground="gray")
        else:
            self.log_text.tag_configure(tag, foreground="black")

    if tag:
        self.log_text.insert(tk.END, msg, tag)
    else:
        self.log_text.insert(tk.END, msg)
    self.log_text.see(tk.END)

    # server會保存文字聊天紀錄
    try:
        with open(self.log_file_path, "a", encoding="utf-8") as f:
            f.write(msg)
    except Exception as e:
        print(f"[log 寫入失敗]: {e}")

# 圖片顯示前處理
def display_image(self, img_bytes, sender="Client"):
    if self.received_text:
        self.log(self.received_text)
        self.received_text = ""
        sender = None
    self.received_image_pending = False
    try:
        img = Image.open(io.BytesIO(img_bytes))
        img.thumbnail((200, 200))
        photo = ImageTk.PhotoImage(img)
        self.log_image(sender, photo, img_bytes)
    except Exception as e:
        self.log(f"[錯誤] 圖片顯示失敗: {e}\n", tag="error")

# 在訊息框內顯示圖片
def log_image(self, sender, photo, original_bytes):
    # 記錄時間
    now = datetime.now().strftime("[%H:%M:%S]")
    if sender == "":
        sender = None
    self.log_text.insert(tk.END, f"[{now}] {sender}:\n")
    if sender:
        self.log_text.insert(tk.END, f"\n")
    img_widget = tk.Label(self.log_text, image=photo, cursor="hand2")
    img_widget.image = photo
    img_widget.bind("<Button-1>", lambda e: self.show_full_image(original_bytes))
    self.log_text.window_create(tk.END, window=img_widget)
    self.log_text.insert(tk.END, "\n")
    self.image_refs.append(photo)
    self.log_text.see(tk.END)
```

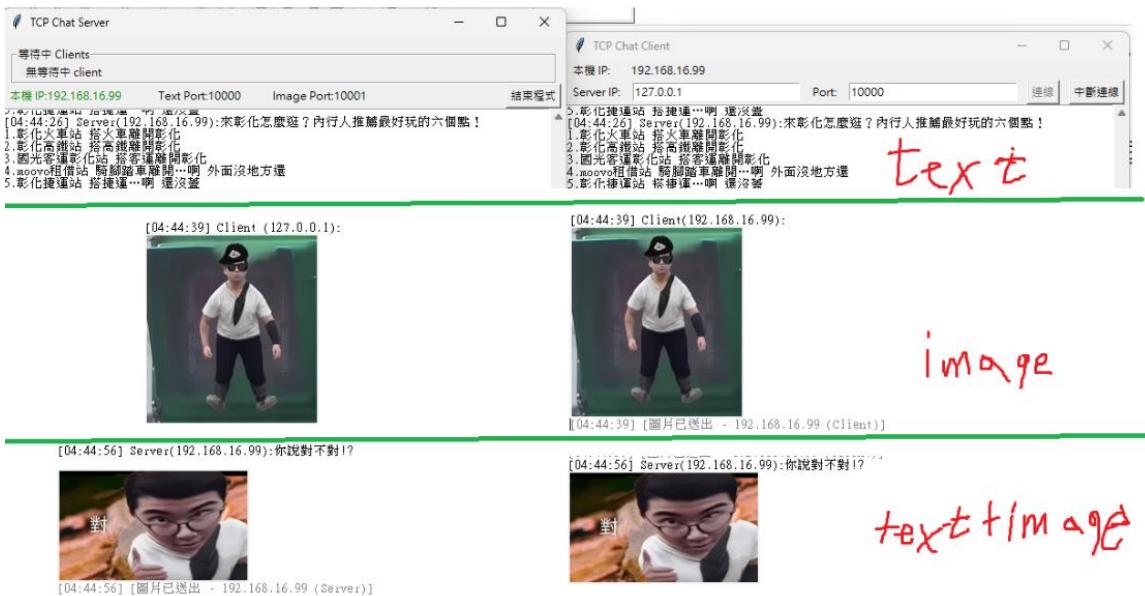
▲圖片/文字訊息於訊息框內的顯示相關函式

## ✧ Server/Client互相圖文傳輸

使用者可選擇打字輸入訊息傳輸，也可選擇按下「選擇圖片」按鈕來選擇傳送圖片。

支援雙向傳送純文字與圖片，且可同時傳送圖文。

**實作方式:** 按下「傳送」按鈕後，透過 `send_message()` 判斷是否有文字或圖片輸入，將文字以 byte 長度+內容封包方式傳送(有效防止訊息內容過長無法傳輸)，圖片以二進位方式傳送，接收端的 `receive_text()` / `receive_image()` 分別以固定 header 長度 (4 bytes) 首先讀取接下來訊息/圖片的 byte 長度，接著持續讀直到超過 byte 長度。圖片傳送後會以 `display_image()` 顯示並插入訊息框。



▲使用者可以選擇發送純文字/純圖片/圖片+文字，並同步顯示於雙方訊息框內

相關程式碼:

```
# 從本地資料夾選取要傳送的圖片
def select_image(self):
    filepath = filedialog.askopenfilename(title="選擇圖片",
                                           filetypes=[("Image files", "*.png *.jpg *.jpeg *.gif *.bmp")])
    if filepath:
        with open(filepath, "rb") as f:
            self.selected_image = f.read()
        img = Image.open(io.BytesIO(self.selected_image)) # 透過BytesIO將讀入的圖片轉成pillow可處理的形式
        img.thumbnail((200, 200))
        photo = ImageTk.PhotoImage(img)
        self.img_label.config(image=photo)
        self.img_label.image = photo
```

▲傳送圖片選擇程式碼

```

# 處理送出訊息(圖片/文字)
def send_message(self):
    msg = self.input_text.get("1.0", tk.END).strip()
    sent_text = False
    sent_image = False

    # 根據目前狀況(是否有文字輸入/圖片選擇)送出訊息
    if self.text_conn:
        if msg:
            full_msg = f"Server({self.local_ip}):{msg}\n"
            try:
                encoded_msg = full_msg.encode()
                self.text_conn.sendall(len(encoded_msg).to_bytes(4, 'big') + encoded_msg)
                sent_text = True
            except:
                self.log("[錯誤] 傳送失敗\n", tag="error")
                self.input_text.delete("1.0", tk.END)
    # 處理圖片傳送
    if self.image_conn and self.selected_image:
        try:
            self.image_conn.sendall(len(self.selected_image).to_bytes(4, 'big') \
                                   + self.selected_image) # 送出圖片大小byte+實際圖片byte的TCP封包
            sent_image = True
        except Exception as e:
            self.log(f"[錯誤] 圖片傳送失敗: {e}\n", tag="error")

    # 最後才來處理訊息框顯示，圖文任何一者成功就log+顯示
    if sent_text or sent_image:
        timestamp = datetime.now().strftime("[%H:%M:%S]")
        sender = f"{timestamp} Server({self.local_ip}):"
        self.log_text.insert(tk.END, sender + (" " + msg + "\n" if sent_text else "") + "")
        if sent_image:
            self.display_image(self.selected_image, sender="")
            self.log(f"[圖片已送出 - {self.local_ip} (Server)]\n", tag="system")
            self.log_text.see(tk.END)
    # 送出後重置已選擇圖片
    self.selected_image = None
    self.img_label.config(image='')
    self.img_label.image = None

```

▲按下傳送訊息時，會執行 `send_message()`，接著根據輸入框內容進行訊息傳輸

```

# 文字訊息接收處理
def receive_text(self):
    self.received_text = ""
    self.received_image_pending = False
    while True:
        # 流程:
        # 1. 每段文字訊息都會先傳送第一段內容表示接下來訊息的長度
        # 2. 持續接收訊息直到超過長度
        # 不論多長的訊息都能進行傳輸
        try:
            length_data = self.text_conn.recv(4)
            if not length_data:
                break
            length = int.from_bytes(length_data, 'big')
            data = b''
            while len(data) < length:
                chunk = self.text_conn.recv(length - len(data))
                if not chunk:
                    break
                data += chunk
            if not data:
                break
            message = data.decode()
            self.received_text = message
            self.received_image_pending = True

            # 延遲顯示直到圖片來，避免當圖文同時接收時訊息框連跳兩次傳送者標頭
            def flush_text():
                if self.received_text:
                    self.log(self.received_text)
                    self.received_text = ""
                    self.received_image_pending = False
            self.window.after(300, flush_text)
        except:
            break
    self.log("(目前連線之Client已離線)\n", tag="system")
    self.text_conn = None
    self.image_conn = None

```

```

# 圖片訊息接收處理
def receive_image(self):
    while True:
        length_data = self.image_conn.recv(4)
        if not length_data:
            break
        length = int.from_bytes(length_data, 'big')
        img_data = b''
        while len(img_data) < length:
            chunk = self.image_conn.recv(length - len(img_data))
            if not chunk:
                break
            img_data += chunk
        self.display_image(img_data, sender=f"Client({self.client_addr[0]})")
    except:
        break

```

▲處理接收到封包的 `receive_text()` 及 `receive_image()`

## ✧ Server/Client中斷連線

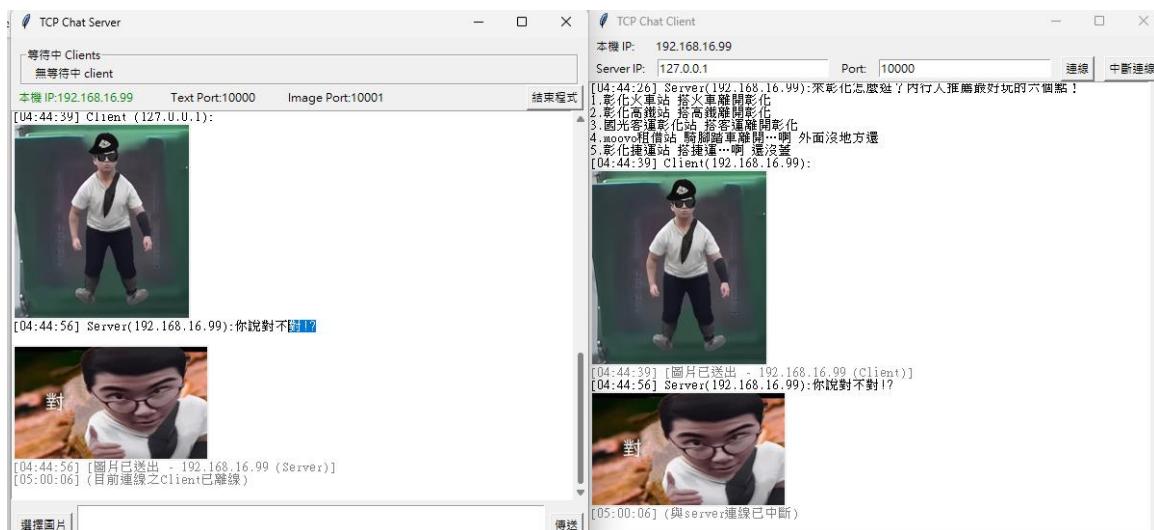
Server 端和 Client 端皆可手動中斷連線。且 Server 端偵測連線斷開後會釋放該連線並接續和排隊 Client 建立連線。當連線中斷時，會在雙方聊天室都透過系統提醒告知。

**實作方式:** Client 使用 `disconnect()` 關閉 `text_socket` 和 `image_socket`。

Server 端接收失敗或 socket 之 `recv()` 為空即中止對應連線並釋放

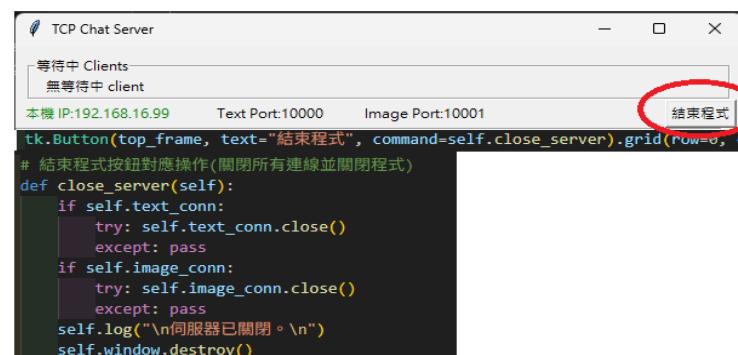
`text_conn/image_conn`。斷線後由 `queue_monitor()` 自動將佇列下一位接入。

(後續會介紹的連線機制)



▲當雙方連線中斷時，會產生系統提醒訊息通知，並且可再次連線

## 相關程式碼:



```
tk.Button(top_frame, text="結束程式", command=self.close_server).grid(row=0, column=6, sticky="e", padx=5)

# 結束程式按鈕對應操作(關閉所有連線並關閉程式)
def close_server(self):
    if self.text_conn:
        try: self.text_conn.close()
        except: pass
    if self.image_conn:
        try: self.image_conn.close()
        except: pass
    self.log("\n伺服器已關閉.\n")
    self.window.destroy()
```

TCP Chat Client

本機 IP: 192.168.16.99  
Server IP: 127.0.0.1 Port: 10000 連線 中斷連線

3.國光客運彰化站 捷客運離開彰化  
4.moovo租借站 腳踏車離開…喲 外面沒地方還  
5.彰化捷運站 捷運…啊 還沒蓋  
[04:44:39] Client(192.168.16.99):

```

tk.Button(top_frame, text="中斷連線", command=self.disconnect).grid(row=1, column=5, padx=5)
# 中斷連線按鈕對應操作(中斷目前client對server連線)
def disconnect(self):
    if self.text_socket:
        try: self.text_socket.close()
        except: pass
        self.text_socket = None
    if self.image_socket:
        try: self.image_socket.close()
        except: pass
        self.image_socket = None
    self.connect_button.config(state="normal")

```

---

```

# 文字訊息接收處理
def receive_text(self):
    self.received_text = ""
    self.received_image_pending = False
    while True:
        # 流程:
        # 1. 每段文字訊息都會先傳送第一段內容表示接下來訊息的長度
        # 2. 持續接收訊息直到超過長度
        # 不論多長的訊息都能進行傳輸
        try:
            length_data = self.text_conn.recv(4)
            if not length_data:
                break
            length = int.from_bytes(length_data, 'big')
            data = b''
            while len(data) < length:
                chunk = self.text_conn.recv(length-len(data))
                if not chunk:
                    break
                data += chunk
            if not data:
                break
            message = data.decode()
            self.received_text = message
            self.received_image_pending = True

```

---

```

# 延遲顯示直到圖片來，避免當圖文同時接收時訊息框連跳兩次傳送者標頭
def flush_text():
    if self.received_text:
        self.log(self.received_text)
        self.received_text = ""
        self.received_image_pending = False
    self.window.after(300, flush_text)
except:
    break

```

```

self.log("(目前連線之Client已離線)\n", tag="system")
self.text_conn = None
self.image_conn = None

```

▲任意一方因為各種原因導致連線中斷時，會讓接收訊息的 thread 跳出迴圈，可以藉此來判斷連線是否中斷

## ✧ 訊息著色

不同類型訊息（錯誤、系統、使用者）以不同顏色顯示，便於識別。

**實作方式：**自訂 log 函式 `log(msg, tag)` 中，根據 tag 設定文字顏色使用 `tag_configure()` 設定 red, blue, gray 等 tag 對應色，例：錯誤資訊設為 "error"，系統提示為 "system"

```
[05:11:57] [錯誤] 無法連線到 Server: [WinError 10061] 無法連線，因為目標電腦拒絕連線。  
[05:12:03] 已連線到 Server 127.0.0.1:10000  
[05:12:03] 歡迎進入聊天室  
[05:12:07] (與server連線已中斷)
```

▲不同類型訊息有不同顏色，便於辨認，畫面顯示資訊更清楚。

相關程式碼：

```
def log(self, msg, tag=None):
    # 紀錄時間
    now = datetime.now().strftime("[%H:%M:%S]")
    msg = f"{now} {msg}"

    if tag and tag not in self.log_text.tag_names():
        if tag == "error":
            self.log_text.tag_configure(tag, foreground="red")
        elif tag == "info":
            self.log_text.tag_configure(tag, foreground="blue")
        elif tag == "system":
            self.log_text.tag_configure(tag, foreground="gray")
        else:
            self.log_text.tag_configure(tag, foreground="black")

    if tag:
        self.log_text.insert(tk.END, msg, tag)
    else:
        self.log_text.insert(tk.END, msg)
    self.log_text.see(tk.END)
```

# 進階程式功能說明

## ✧ Server多個Client連線等待機制

Server 可同時接受多位 Client 嘗試連線，但僅允許其中一人與 Server 通訊，其餘進入等待佇列，依序接入。若已有 client 連線，則透過 `queue.Queue()` 將新 client 加入排隊，每個新連線會先進入 `waiting_clients` 的 list 中，只有當 Server 端的 `text_conn` 為 `None` 時(代表目前沒有Client連線狀態)，才會接著啟用該排隊 client連線

**實作方式**：Server端在處理Client連入時，會分成兩個部分，1. 直接建立連線

監聽訊息傳輸 2. 已經有Client連線時，暫時放入Queue內等待，分別對應

`handle_client()`和`queue_monitor()`兩個函式：

`handle_client()`：負責處理嘗試連入的Client端，若已經有連線->不進一步建立後續訊息接收，直到目前連線之Client中斷連線。

`queue_monitor()`：負責將嘗試的 Client 連線加入 queue，並管理整個排隊佇列。



▲當有多個 Client 端嘗試連線時，後面的 Client 會被放入等待佇列，直到目前的 Client 連線結束才會接著依序跟等待佇列中的 Client 建立聊天連線。

## 相關程式碼：

```
# 建立並等待與client的TCP連線
# 分成兩個Thread來處理列隊等待中的client
# 1. handle_client(): 處理當client嘗試連線時的排隊情況，並在client連入或排隊等到時進一步進到後續接收訊息階段
# 2. queue_monitor(): 實際管理排隊並進一步放入client連線
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((self.HOST, self.TEXT_PORT))
server_socket.listen(5)
self.log("等待 client 連線中...\n", tag="system")

def handle_client(conn, addr):
    if self.text_conn is None:
        self.text_conn = conn
        self.client_addr = addr
        self.log_text.delete("0.0", tk.END) # 新連線清空聊天紀錄
        self.log(f"Client {addr} 已連線！\n", tag="info")
        self.text_conn.sendall(len("歡迎進入聊天室\n".encode()).to_bytes(4, 'big') + "歡迎進入聊天室\n".encode())
        threading.Thread(target=self.receive_text, daemon=True).start()
        threading.Thread(target=self.start_image_server, args=(addr,), daemon=True).start()
    else:
        identifier = f"[addr[0]]:{addr[1]}"
        self.waiting_clients.put((conn, addr))
        self.waiting_addrs.append(identifier)
        self.update_waiting_label()
        pos = self.waiting_clients.qsize()
        try:
            msg = f"您是第 {pos} 位等待中，請稍候...\n".encode()
            conn.sendall((len(msg).to_bytes(4, 'big') + msg))
        except:
            conn.close()

    return
```

已有連線  
進入等候

▲handle\_client()依據 client 嘗試連線的兩種情況(1. 已經有連線建立, 2 尚無連線建立) · 來決定目前該 client 之連線應該先放入佇列或是開始開放訊息接收/傳輸

```
def queue_monitor():
    import time
    while True:
        if self.text_conn is None and not self.waiting_clients.empty():
            try:
                #self.log_text.delete("0.0", tk.END) # 新連線清空聊天紀錄
                conn, addr = self.waiting_clients.get(timeout=0.1)
                identifier = f"[addr[0]]:{addr[1]}"
                if identifier in self.waiting_addrs:
                    self.waiting_addrs.remove(identifier)
                    self.window.after(0, self.update_waiting_label)
                    self.window.after(0, lambda: handle_client(conn, addr))
            except queue.Empty:
                pass
            time.sleep(0.2)

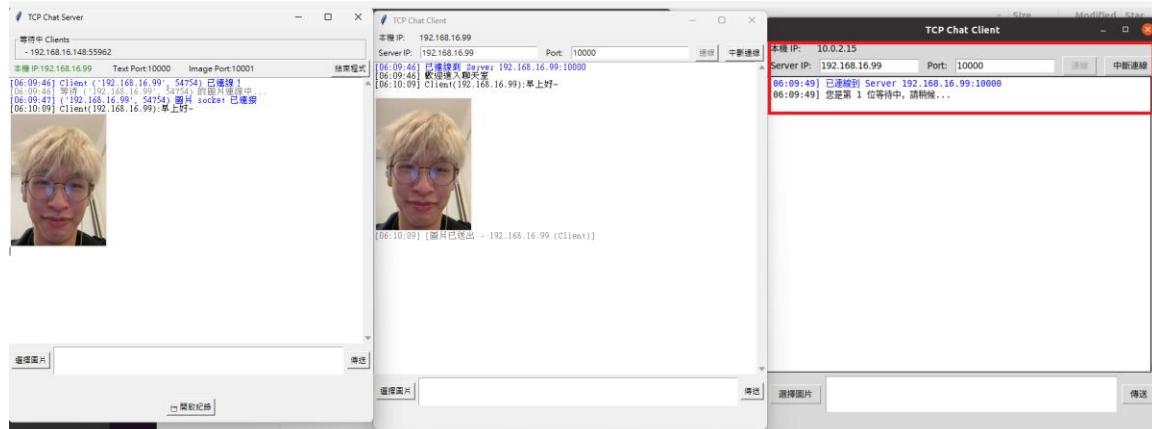
    threading.Thread(target=queue_monitor, daemon=True).start()
    while True:
        conn, addr = server_socket.accept()
        threading.Thread(target=handle_client, args=(conn, addr), daemon=True).start()
```

▲queue\_monitor()用於持續管理監控 Client 連線等待佇列的最新情況

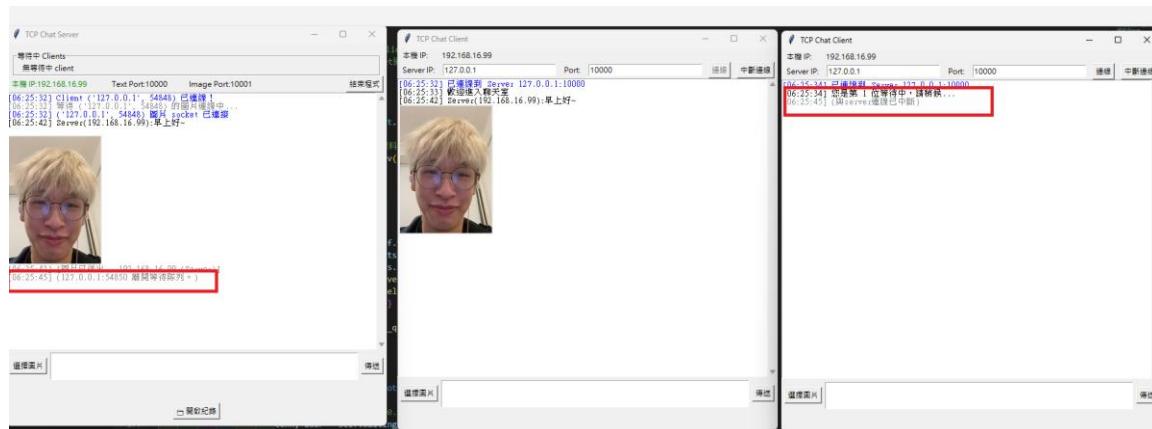
## ✧ Client 排隊等待機制

Client 若非第一位進入，會自動進入等待佇列，並收到「您是第 N 位等待中...」提示。若有 client 離線，佇列自動遞補。

**實作方式：**Server 使用 `waiting_addrs` 儲存正在等待的 IP:port。透過 `conn.sendall()` 回傳排隊通知訊息給 client。若 client 中途斷線，`monitor_queue_socket()` 會每隔兩秒用 `select.select()` 偵測 connection socket 是否還存在，若不存在代表該Client與Server之連線以中斷，將該Client之連線從queue佇列中移除。



▲當 Client 需要進入排隊佇列時，會收到來自 Server 的通知訊息



▲當排隊中的 Client 斷線時，Server 端會跳出系統通知，並且將 Client 移除等待佇列

## 相關程式碼:

```
# monitor_queue_socket(): 監控client端是否在排隊時中斷連線
# 原理是偵測與該client連線的socket通道是否有中斷，若中斷即代表離開等待連入server佇列
import select
def monitor_queue_socket():
    try:
        while True:
            rlist, _, _ = select.select([conn], [], [], 0.5)
            if rlist:
                # 確保socket有資料可讀才來check
                peek = conn.recv(1, socket.MSG_PEEK)
                if not peek:
                    break
    except:
        pass

    # 一旦連線中斷就移除
    if (conn, addr) in list(self.waiting_clients.queue):
        with self.waiting_clients.mutex:
            self.waiting_clients.queue.remove((conn, addr))
            self.waiting_addrs.remove(identifier)
            self.update_waiting_label()
            self.log(f"{identifier} 離開等待隊列。)\n", tag="system")

    threading.Thread(target=monitor_queue_socket, daemon=True).start()

# 更新server端顯示的client等待佇列
def update_waiting_label(self):
    if self.waiting_addrs:
        text = "\n".join(f"- {addr}" for addr in self.waiting_addrs)
    else:
        text = "無等待中 client"
    self.waiting_label.config(text=text)
```

▲Server 端會持續監控等待佇列中的 Client 是否斷線，並且同步更新 GUI 排隊顯示

## ✧ 時間戳記

每筆訊息都會加上 [HH:MM:SS] 格式的時間戳記，顯示在訊息最前方，提升紀錄一致性與可追蹤性。

**實作方式：**使用 `datetime.now().strftime("[%H:%M:%S]")` 產生時間字串，

將其包含在 `log()` 函式中，每筆顯示或儲存訊息時自動加上。



## 相關程式碼：

```
# 於聊天框內顯示訊息，透過tag區分顏色
def log(self, msg, tag=None):
    # 紀錄時間
    now = datetime.now().strftime("[%H:%M:%S]")
    msg = f'{now} {msg}'

    if tag and tag not in self.log_text.tag_names():
        if tag == "error":
            self.log_text.tag_configure(tag, foreground="red")
        elif tag == "info":
            self.log_text.tag_configure(tag, foreground="blue")
        elif tag == "system":
            self.log_text.tag_configure(tag, foreground="gray")
        else:
            self.log_text.tag_configure(tag, foreground="black")

    if tag:
        self.log_text.insert(tk.END, msg, tag)
    else:
        self.log_text.insert(tk.END, msg)
    self.log_text.see(tk.END)
```

```
# 在訊息框內顯示圖片
def log_image(self, sender, photo, original_bytes):
    # 紀錄時間
    now = datetime.now().strftime("[%H:%M:%S]")
    if sender == "":
        sender = None
    self.log_text.insert(tk.END, f"\n{now} {sender}:\n")
    img_widget = tk.Label(self.log_text, image=photo, cursor="hand2")
    img_widget.image = photo
    img_widget.bind("<Button-1>", lambda e: self.show_full_image(original_bytes))
    self.log_text.window_create(tk.END, window=img_widget)
    self.log_text.insert(tk.END, "\n")
    self.image_refs.append(photo)
    self.log_text.see(tk.END)
```

## ✧ 聊天紀錄保存

Server 端會將聊天訊息紀錄自動儲存成 .txt 檔，並存在 /chat\_logs/ 資料夾，可一鍵開啟查看。

**實作方式:** Server 端啟動時建立檔案路徑與資料夾

(os.makedirs("chat\_logs", exist\_ok=True))，在 log() 每筆訊息輸入時會同時寫入紀錄檔案內，點擊按鈕「開啟紀錄」會呼叫 subprocess 開啟資料夾。



## 相關程式碼:

```
# 文字記錄保存相關參數，檔案名稱設定為目前時間
LOG_DIR = "chat_logs"
os.makedirs(LOG_DIR, exist_ok=True)
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
self.log_file_path = os.path.join(LOG_DIR, f"chat_log_{timestamp}.txt")
```

▲ Server 端啟動時會根據目前時間日期，建立聊天紀錄檔案

```
# 於聊天框內顯示訊息，透過tag區分顏色
def log(self, msg, tag=None):
    # 記錄時間
    now = datetime.now().strftime("[%H:%M:%S]")
    msg = f"[{now}] {msg}"

    if tag and tag not in self.log_text.tag_names():
        if tag == "error":
            self.log_text.tag_configure(tag, foreground="red")
        elif tag == "info":
            self.log_text.tag_configure(tag, foreground="blue")
        elif tag == "system":
            self.log_text.tag_configure(tag, foreground="gray")
        else:
            self.log_text.tag_configure(tag, foreground="black")

    if tag:
        self.log_text.insert(tk.END, msg, tag)
    else:
        self.log_text.insert(tk.END, msg)
    self.log_text.see(tk.END)

# server會保存文字聊天紀錄
try:
    with open(self.log_file_path, "a", encoding="utf-8") as f:
        f.write(msg)
except Exception as e:
    print(f"[log 寫入失敗]: {e}")

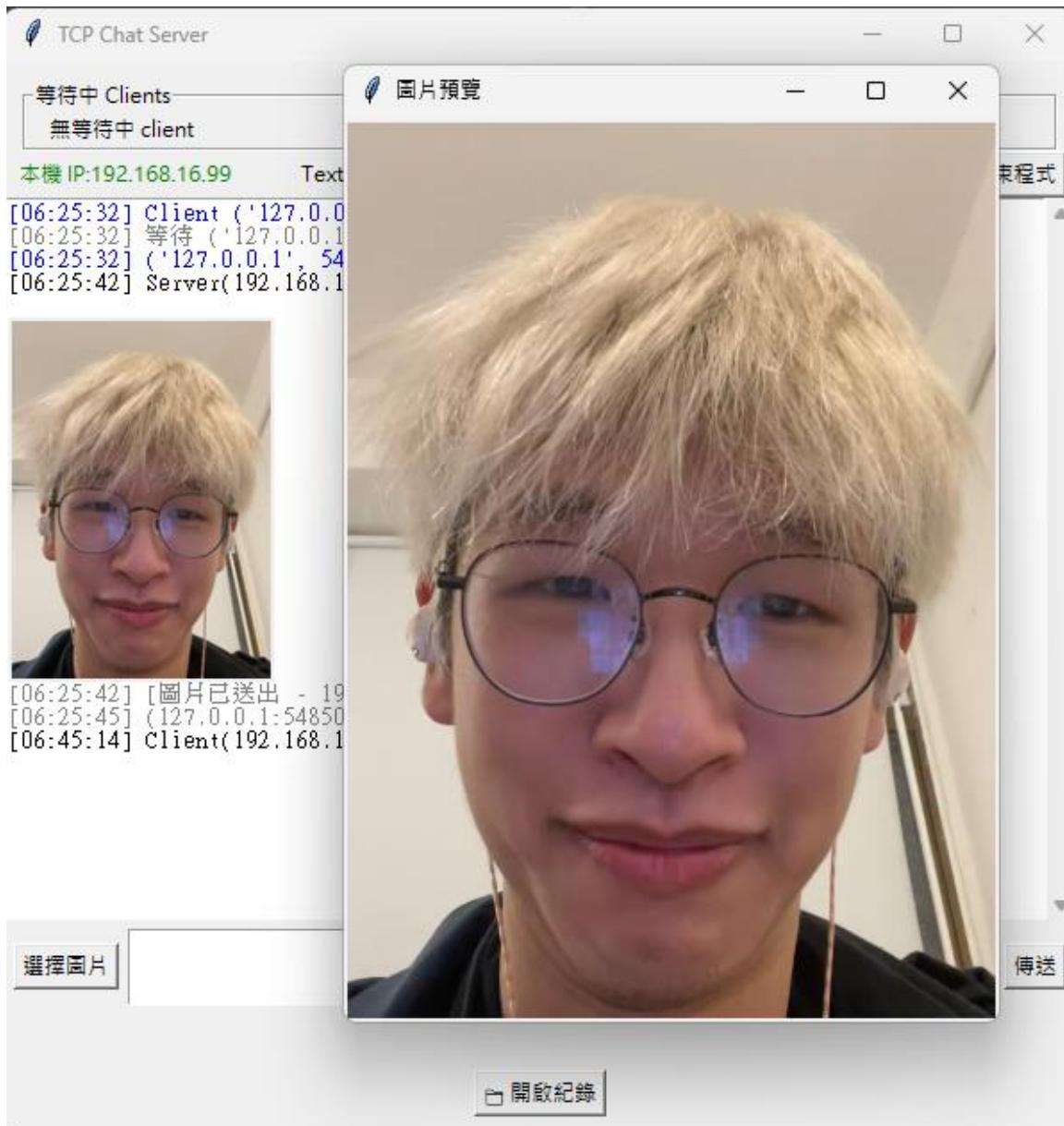
# 開啟聊天紀錄檔案夾
def open_log_folder(self):
    log_path = os.path.abspath("chat_logs")
    try:
        # 根據作業系統選擇開啟檔案資料夾的執行指令
        if os.name == "nt": # Windows
            subprocess.Popen(["explorer", log_path])
        elif os.name == "posix": # macOS / Linux
            subprocess.Popen(["xdg-open", log_path])
    except Exception as e:
        self.log(f"[錯誤] 無法開啟資料夾: {e}", tag="error")
```

## ✧ 圖片閱覽

圖片會縮圖顯示於聊天視窗，使用者可點擊圖片開啟新視窗放大檢視原圖。

**實作方式：**使用PIL函式庫內的`Image.open()` 搭配 `thumbnail()` 產生預覽圖。

接著在將圖片放入訊息框時將其綁定`<Button-1>`左鍵單擊事件，接著設計一個新的彈跳視窗顯示原圖。



▲點擊訊息框內的圖片，可達到放大原圖查看的效果

## 相關程式碼：

```
# 點擊訊息框內的圖片可放大檢視
def show_full_image(self, img_bytes):
    try:
        img = Image.open(io.BytesIO(img_bytes))
        top = Toplevel(self.window)
        top.title("圖片預覽")
        width, height = img.size
        top.geometry(f"{width}x{height}")

        # 使用canvas+toplevel模塊來額外彈出視窗顯示原圖片
        photo = ImageTk.PhotoImage(img)
        canvas = Canvas(top, width=width, height=height)
        canvas.pack()
        canvas.create_image(0, 0, anchor=tk.NW, image=photo)
        canvas.image = photo
    except:
        messagebox.showerror("錯誤", "無法開啟圖片")
```

## 二、程式碼註解與功能對應說明

Server 端程式函式功能對照表

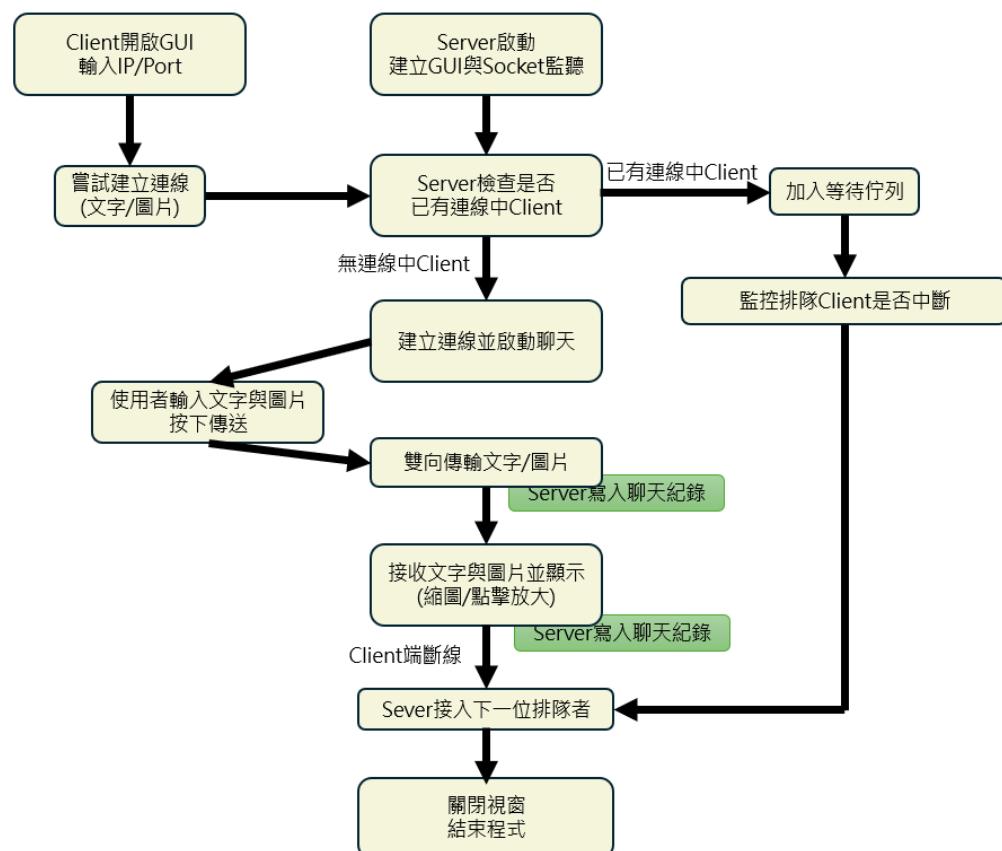
| 函式名稱                 | 功能說明                  |
|----------------------|-----------------------|
| setup_gui()          | 建立主視窗與畫面佈局            |
| start_text_server()  | 文字連線 socket 管理與排隊等待邏輯 |
| start_image_server() | 圖片 socket 傳輸與接收       |
| receive_text()       | 處理 client 傳來的文字訊息     |
| receive_image()      | 處理圖片封包並傳至視窗           |
| send_message()       | Server 傳送文字與圖片        |
| display_image()      | 聊天框內插入縮圖              |
| log()                | 時間戳+訊息顯示與儲存           |
| log_image()          | 設定圖片點擊放大效果            |
| open_log_folder()    | 開啟聊天紀錄資料夾             |
| close_server()       | 關閉連線與視窗               |

## Client 端程式函式功能對照表

| 函式名稱            | 功能說明            |
|-----------------|-----------------|
| setup_gui       | 畫面與控制介面配置       |
| connect         | 建立與 Server 的連線  |
| disconnect      | 中斷連線與 UI 狀態重設   |
| receive_text    | 接收並顯示 Server 文字 |
| receive_image   | 接收圖片並顯示         |
| select_image    | 選擇圖片並預覽         |
| send_message    | 將文字/圖片送出        |
| display_image   | 插入圖片元件          |
| log             | 加時間戳與顏色標示       |
| show_full_image | 圖片放大檢視          |

## 三、操作流程與使用方式

### 整體程式運作流程



- 【Server】啟動後等待 client 連線，接收訊息與圖片，可傳送訊息與圖片並查看等待隊列。
- 【Client】輸入 IP/port 連線，可打字與傳圖，並於對話框預覽圖片與文字。中斷後可重新連線。

整體流程如下：

#### 1. Server 啟動並監聽連線

Server 啟動時會顯示 GUI，並啟動文字與圖片的 socket 監聽埠，準備接受連線。

#### 2. Client 啟動階段

使用者啟動 Client 端程式後，進入圖形化操作介面（GUI），輸入 Server 的 IP 位置與連接埠號（Port）。

#### 3. Client 嘗試建立連線

一旦輸入完畢，Client 會同時向 Server 發起兩條連線請求：一條用於文字傳輸，一條用於圖片傳輸。

#### 4. 檢查是否已有其他 Client 在線

Server 接收到新的連線請求後，會判斷當前是否已有 Client 正在與其連線中。

#### 5. 若已有 Client 在線，則加入等待佇列

若系統已有人連線中，新進來的 Client 會被加入「等待佇列」，Server 同時在 GUI 上顯示等待中的 Client 資訊，並傳送「請稍候」訊息給排隊者。

#### 6. Server 監控排隊者是否中斷連線

每位在等待佇列中的 Client 都會被持續監控，若使用者中途關閉程式或連線中斷，將會自動從佇列移除。

#### 7. 若無其他 Client 在線，則建立連線並啟動聊天

若目前無其他 Client 使用中，Server 會直接與該 Client 建立聊天連線，並啟動專屬的訊息與圖片處理執行緒。

#### 8. 系統雙向傳輸文字與圖片

無論是 Server 或 Client，都可互傳訊息與圖片。訊息帶有時間戳記，圖片會顯示為縮圖，並可點擊放大。

## 9. Server 寫入聊天紀錄

每當訊息送出或收到，Server 都會自動將訊息記錄存入 chat\_logs 目錄下的檔案中，方便日後查閱。

## 10. 任一方斷線時

若聊天過程中 Client 關閉視窗或發生斷線，Server 會立即中止當前會話。若 Server 關閉式窗或發生斷線，也會立即中止 Client 端當前會話。

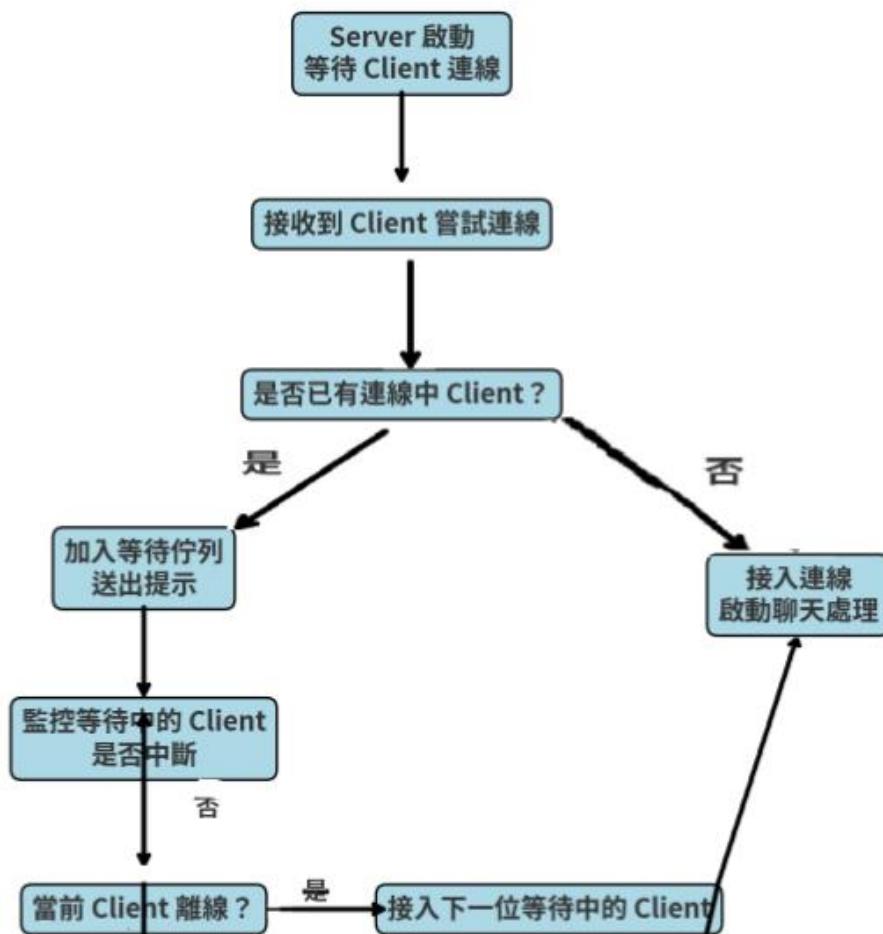
## 11. Server 自動接入下一位排隊者

當前 Client 中斷連線，若等待佇列中還有其他 Client，Server 會立即接入下一位 Client，並重新進入聊天流程。

## 12. 關閉程式結束作業

Server 或 Client 結束視窗後，程式會自動關閉所有 socket 並安全終止。

## 一對一聊天排隊機制運作流程



整個 Server 的連線流程採取「一對一即時聊天」模型，並加入排隊等待邏輯來支援多位 Client 嘗試同時連線的情況。具體流程如下：

1. Server 啟動後，會持續監聽來自 Client 的連線請求。
2. 當接收到某一 Client 嘗試連線時，Server 首先會檢查是否已有一位 Client 正在通訊中。
3. 若尚未有任何 Client 連線中，該 Client 會直接接入並建立通訊連線，此時 Server 會啟動兩個對應的處理執行緒，分別負責文字與圖片的接收與顯示。
4. 若已有其他 Client 正在連線中，則新的 Client 會被加入等待佇列中，Server 會透過 socket 傳送一段提示文字：「您是第 N 位等待中，請稍候...」，並於 GUI 上的等待區塊顯示該 Client 的 IP 與 Port。
5. Server 對於每個排隊中的 Client 都會啟動一個輕量監控程序，使用 select.select() 檢查其是否中途斷線，若該 Client 中途離線，系統會自動將其從佇列中移除，並更新等待區畫面。
6. 同時，Server 會持續偵測目前通訊中的 Client 是否已斷線，若偵測到離線或連線中斷，便會從等待佇列中自動提取下一位 Client 來建立新連線，無須重新啟動 Server。
7. 接入下一位 Client 後，Server 會清空訊息框並重設狀態，再次進入一對一聊天模式，如此持續輪替進行。

#### 四、心得與問題解決

這次作業讓我深入體驗了 TCP socket 網路通訊、GUI 介面設計 以及 多線程排程與同步控制 的實作過程。從一開始作業基本要求的圖文傳輸「Client–Server 一對一聊天室」，到後來加入「等待佇列管理」、「聊天記錄儲存」與「圖文預覽」等功能，整體系統在穩定性、使用性與可視性上都有大幅提升。

過程中，我不僅強化了對 socket 通訊機制的掌握，也實際應用了 Tkinter 介面元件的佈局與互動控制技巧，在寫作業的過程中，不斷的去思考要完成這個功能我可

能會需要那些函式需要開幾個 Thread，還有更多的是我要怎麼去寫才可以讓整體程式架構在後續 debug 或加入新功能時能夠更快速方便，也學會了讓程式具備完整的「使用者互動性」與「非同步處理能力」。

特別是 GUI 同步與連線監控這部分，讓我體認到網路應用不僅需要正確的邏輯，還需顧及使用者體驗與錯誤處理彈性，在測試過程中會一直因為覺得 GUI 介面的哪些部分使用起來不順，或是呈現上看起來很怪，而一直不斷去重新修改程式。能從 0 開始逐步打造出一個具有一定水準的應用，這對我而言是一項極具成就感的經驗。

而在過程中我也遇到了許多不少的問題：

| 問題描述             | 解決方式                                     |
|------------------|--|
| 多 Client 同時連線衝突  | 使用等待佇列 queue.Queue 管理，僅允許一位 Client 接入    |
| 同時傳送圖文會重複顯示發送者資訊 | 設計 received_image_pending 狀態標記合併顯示訊息     |
| Client 排隊時中斷無法偵測 | 使用 select.select() 實現非阻塞 socket 監控中斷     |
| GUI 卡頓與流程不同步     | 使用 threading.Thread 與 after() 控制 UI 異步更新 |
| 訊息過長掉包 / 黏包問題    | 加入訊息長度標頭 (4 bytes) 解決封包界線問題              |

## 1. 多 Client 同時連線衝突

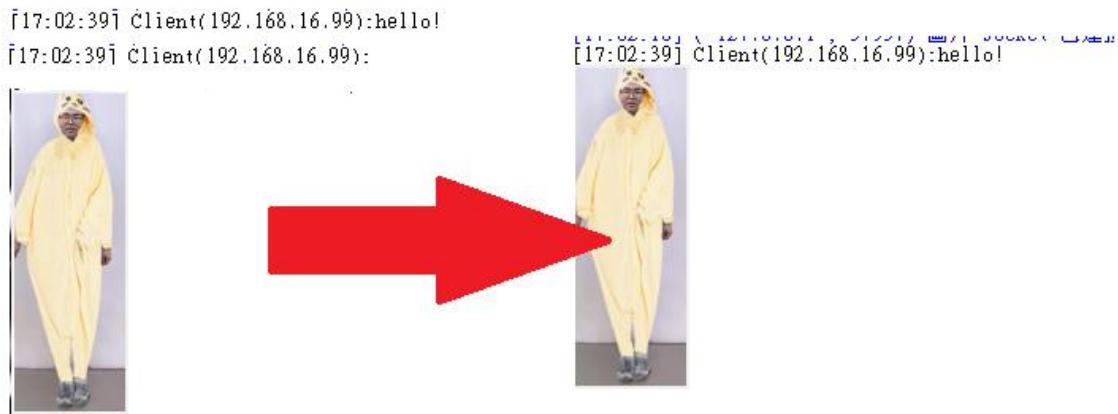
當有兩位 Client 幾乎同時嘗試連線時，由於 Server 僅設計一個文字/圖片通道，導致資料競爭，Socket 被覆蓋或搶占。

解決方式：建立排隊機制，透過 `queue.Queue()` 架構，並設計持續偵測連線的 `queue_monitor()` 及 `monitor_queue_socket()` 等相關函式，讓所有嘗試連線的

Client 被放入等待併列。Server 每次僅接入一位 Client，等到上一位斷線後自動輪入下一位，避免衝突。

## 2. 同時傳送圖文會重複顯示發送者資訊

在一開始寫的時候，當使用者同時傳送一段文字與圖片，Server 的訊息框會兩次出現「Client(IP):」的訊息列，顯得冗餘。



解決方式：在 Server 中設計 `received_image_pending` 狀態旗標。如果接下來馬上要接收圖片，文字顯示先延後，圖片接收完畢後再整合圖文內容統一顯示一次。文字輸出會延後 300ms，此時若有接收到圖片則表時圖文同時傳送，文字顯示交給 `display_image()`，若接收到的是純文字則 300ms 後會自動顯示出文字訊息。

```
self.received_text = message
self.received_image_pending = True

# 延遲顯示直到圖片來，避免當圖文同時接收時訊息框連跳兩次傳送者標頭
def flush_text():
    if self.received_text:
        self.log(self.received_text)
        self.received_text = ""
        self.received_image_pending = False
    self.window.after(300, flush_text)
except:
    break
```

```
# 圖片顯示前處理
def display_image(self, img_bytes, sender="Client"):
    if self.received_text:
        self.log(self.received_text)
        self.received_text = ""
        sender = None
    self.received_image_pending = False
    try:
        img = Image.open(io.BytesIO(img_bytes))
        img.thumbnail((200, 200))
        photo = ImageTk.PhotoImage(img)
        self.log_image(sender, photo, img_bytes)
    except Exception as e:
```

### 3. Client 排隊時中斷無法偵測

Client 若在排隊過程中關閉視窗，原本會一直停留在等待佇列中，造成之後無法排入新 Client。

解決方式：利用 `select.select()` 非阻塞 socket 偵測技術，每隔一小段時間檢查是否還有封包可 peek，若無則代表 Server 與等待排隊之 Client 已斷線，自動移除斷線 Client。

```
# monitor_queue_socket(): 監控client端是否在排隊時中斷連線
# 原理是偵測與該client連線的socket通道是否有中斷，若中斷即代表離開等待連入server佇列
import select

def monitor_queue_socket():
    try:
        while True:
            rlist, _, _ = select.select([conn], [], [], 0.5)
            if rlist:
                # 確保socket有資料可讀才來check
                peek = conn.recv(1, socket.MSG_PEEK)
                if not peek:
                    break
    except:
        pass

    # 一旦連線中斷就移除
    if (conn, addr) in list(self.waiting_clients.queue):
        with self.waiting_clients.mutex:
            self.waiting_clients.queue.remove((conn, addr))
            self.waiting_addrs.remove(identifier)
            self.update_waiting_label()
            self.log(f"{identifier} 離開等待隊列。)\n", tag="system")
```

### 4. GUI 卡頓與流程不同步

在不斷擴大程式，以及開發排隊機制的過程中，我發現可能因為 Thread 越開越多的情況下，導致整個程式變得非常卡頓，需要立刻進行優化調整。

解決方式：重新調整設計排隊機制的程式架構，將所有接收端與排程邏輯放入獨立 Thread，避免阻塞主執行緒，搭配 `window.after()` 方法控制 GUI 資料更新，使 UI 流暢不卡頓。

```

def queue_monitor():
    import time
    while True:
        if self.text_conn is None and not self.waiting_clients.empty():
            try:
                #self.log_text.delete("0.0", tk.END) # 新連線清空聊天紀錄
                conn, addr = self.waiting_clients.get(timeout=0.1)
                identifier = f"{addr[0]}:{addr[1]}"
                if identifier in self.waiting_addrs:
                    self.waiting_addrs.remove(identifier)
                self.window.after(0, self.update_waiting_label)
                self.window.after(0, lambda: handle_client(conn, addr))
            except queue.Empty:
                pass
            time.sleep(0.2)
    def handle_client(conn, addr):
        if self.text_conn is None:
            self.text_conn = conn
            self.client_addr = addr
            self.log_text.delete("0.0", tk.END) # 新連線清空聊天紀錄
            self.log(f"Client {addr} 已連線！\n", tag="info")
            self.text_conn.sendall(len("歡迎進入聊天室\n").encode()).to_bytes(4, 'big') + "歡迎進入聊天室\n".encode()
            threading.Thread(target=self.receive_text, daemon=True).start()
            threading.Thread(target=self.start_image_server, args=(addr,), daemon=True).start()
    def queue_monitor():
        import time
        while True:
            if self.text_conn is None and not self.waiting_clients.empty():
                try:
                    #self.log_text.delete("0.0", tk.END) # 新連線清空聊天紀錄
                    conn, addr = self.waiting_clients.get(timeout=0.1)
                    identifier = f"{addr[0]}:{addr[1]}"
                    if identifier in self.waiting_addrs:
                        self.waiting_addrs.remove(identifier)
                    self.window.after(0, self.update_waiting_label)
                    self.window.after(0, lambda: handle_client(conn, addr))
                except queue.Empty:
                    pass
                time.sleep(0.2)
        threading.Thread(target=queue_monitor, daemon=True).start()

```

## 5. 訊息過長掉包 / 黏包問題

傳送長訊息或連續訊息時，容易因 黏包或太常導致訊息不完整或多筆混在一起。

解決方式：設計傳送訊息前先送出固定 4 Bytes 的長度標頭，接收端先讀長度再收資料，確保每筆資料封包完整分離。

```

# 流程：
# 1. 每段文字訊息都會先傳送第一段內容表示接下來訊息的長度
# 2. 持續接收訊息直到超過長度
# 不論多長的訊息都能進行傳輸
try:
    length_data = self.text_conn.recv(4)
    if not length_data:
        break
    length = int.from_bytes(length_data, 'big')
    data = b''
    while len(data) < length:
        chunk = self.text_conn.recv(length - len(data))
        if not chunk:
            break
        data += chunk
    if not data:
        break
    message = data.decode()
    self.received_text = message
    self.received_image_pending = True

```

在這次作業遇到的這些問題有不少都是額外花了許多時間在重新去設計架構和思考解決辦法，甚至有些內容是我之前不會的，也因此查了許多額外資料，不過最後都有成功解決，也學到了更多額外的實作技巧，非常的值得。

總的來說，這次作業涵蓋了 網路通訊、資料封包、介面設計、非同步處理與錯誤管理等多層面技能，透過每個功能逐步實作與改進，我對整體系統開發流程有了更全面的理解。未來若要擴充為多人聊天室、加密通訊、或整合資料庫儲存，我也更有信心從這次的基礎出發，持續優化與開發！

## 五、程式碼

### chat\_ftps.py:

```
1. import socket
2. import threading
3. import tkinter as tk
4. from tkinter.scrolledtext import ScrolledText
5. from tkinter import messagebox, filedialog, Toplevel, Canvas
6. from PIL import Image, ImageTk
7. import io
8. import select
9. import queue
10. import subprocess
11. import os
12. from datetime import datetime
13.
14.
15. class ChatServer:
16.     def __init__(self, host='0.0.0.0', text_port=10000, image_port=10001):
17.         # 初始化 chat server 的設定
18.         self.HOST = host
19.         self.TEXT_PORT = text_port
20.         self.IMAGE_PORT = image_port
21.         self.text_conn: socket.socket = None # 文字傳輸的連線
22.         self.image_conn: socket.socket = None # 圖片傳輸的連線
23.         self.client_addr = None
24.         self.local_ip = self.get_local_ip()
25.         self.image_refs = [] # 保留紀錄圖片傳輸紀錄
26.         self.selected_image = None # 暫存目前選取要傳送的圖片
27.
28.         self.waiting_clients = queue.Queue()
29.         self.waiting_addrs = [] # 紀錄等待連線中的 client IP
30.
31.         # 文字記錄保存相關參數，檔案名稱設定為目前時間
32.         LOG_DIR = "chat_logs"
33.         os.makedirs(LOG_DIR, exist_ok=True)
34.         timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
35.         self.log_file_path = os.path.join(LOG_DIR, f"chat_log_{timestamp}.txt")
36.
```

```

37.         self.setup_gui() # 初始化界面
38.         threading.Thread(target=self.start_text_server, daemon=True).start() # 初始化
39.         socket 監聽
40.         # 自動抓取本地 IP 位址
41.         def get_local_ip(self):
42.             # 透過對內部網路建立一次連線來得到本地 ip 位址
43.             # socket 會自動偵測本機的網路介面，並綁定適當的 IP 連接，透過這個原理可以不用設定本地 IP 就
44.             # 得到本地位址
45.             s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
46.             try:
47.                 s.connect(('10.255.255.255', 1))
48.                 ip = s.getsockname()[0]
49.             except:
50.                 ip = '127.0.0.1'
51.             finally:
52.                 s.close()
53.             return ip
54.         # Server GUI 畫面建立
55.         def setup_gui(self):
56.             self.window = tk.Tk()
57.             self.window.title("TCP Chat Server")
58.             self.window.geometry("600x600")
59.
60.             self.window.grid_rowconfigure(2, weight=1)
61.             self.window.grid_columnconfigure(0, weight=1)
62.
63.             # 最上方顯示 waiting frame
64.             self.waiting_frame = tk.LabelFrame(self.window, text="等待中 Clients")
65.             self.waiting_frame.grid(row=0, column=0, sticky="ew", padx=(10, 0))
66.             self.waiting_label = tk.Label(self.waiting_frame, text="無等待中 client")
67.             self.waiting_label.pack(anchor="w", padx=10)
68.
69.             # top_frame: 連線資訊與控制
70.             top_frame = tk.Frame(self.window)
71.             top_frame.grid(row=1, column=0, sticky="ew")
72.
73.             for i in range(7):
74.                 top_frame.grid_columnconfigure(i, weight=1)
75.
76.                 tk.Label(top_frame, text=f"本機 IP:{self.local_ip}", fg="green").grid(row=0,
77. column=0, sticky="w", padx=5)
78.                 tk.Label(top_frame, text=f"Text Port:{self.TEXT_PORT}").grid(row=0, column=1,
79. sticky="w")
80.                 tk.Label(top_frame, text=f"Image Port:{self.IMAGE_PORT}").grid(row=0, column=2,
81. sticky="w")
82.                 tk.Button(top_frame, text="結束程式", command=self.close_server).grid(row=0,
83. column=6, sticky="e", padx=5)
84.
85.             # middle_frame: 訊息框
86.             middle_frame = tk.Frame(self.window)
87.             middle_frame.grid(row=2, column=0, sticky="nsew")
88.             middle_frame.grid_rowconfigure(0, weight=1)
89.             middle_frame.grid_columnconfigure(0, weight=1)
90.
91.             self.log_text = ScrolledText(middle_frame, width=50, height=20)
92.             self.log_text.grid(row=0, column=0, sticky="nsew")

```

```

92.         bottom_frame.grid(row=3, column=0, sticky="ew", pady=5)
93.         bottom_frame.grid_columnconfigure(1, weight=1)
94.
95.         tk.Button(bottom_frame, text="選擇圖片", command=self.select_image).grid(row=0,
96.             column=0, padx=5)
96.         self.input_text = tk.Text(bottom_frame, height=3)
97.         self.input_text.grid(row=0, column=1, sticky="ew")
98.         tk.Button(bottom_frame, text="傳送", command=self.send_message).grid(row=0,
99.             column=2, padx=5)
100.
100.        self.img_label: tk.Label = tk.Label(self.window)
101.        self.img_label.grid(row=4, column=0, pady=5)
102.
103.        # 開啟連天記錄存檔的資料夾
104.        tk.Button(self.window, text="開啟紀錄",
105.            command=self.open_log_folder).grid(row=5, column=0, pady=(0, 10))
105.
106.        # 更新 server 端顯示的 client 等待佇列
107.        def update_waiting_label(self):
108.            if self.waiting_addrs:
109.                text = "\n".join(f"- {addr}" for addr in self.waiting_addrs)
110.            else:
111.                text = "無等待中 client"
112.            self.waiting_label.config(text=text)
113.
114.        # 當 client 連入時，建立與 client 端的文字訊息傳輸連線
115.        def start_text_server(self):
116.            # 建立並等待與 client 的 TCP 連線
117.            # 分成兩個 Thread 來處理列隊等待中的 client
118.            # 1. handle_client(): 處理當 client 嘗試連線時的排隊情況，並在 client 連入或排隊等到時進一步進到後續接收訊息階段
119.            # 2. queue_monitor(): 實際管理排隊並進一步放入 client 連線
120.            server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
121.            server_socket.bind((self.HOST, self.TEXT_PORT))
122.            server_socket.listen(5)
123.            self.log("等待 client 連線中...\n", tag="system")
124.
125.            def handle_client(conn, addr):
126.                if self.text_conn is None:
127.                    self.text_conn = conn
128.                    self.client_addr = addr
129.                    self.log_text.delete("0.0", tk.END) # 新連線清空聊天紀錄
130.                    self.log(f"Client {addr} 已連線! \n", tag="info")
131.                    self.text_conn.sendall(len("歡迎進入聊天室\n").encode().to_bytes(4, 'big'))
132.                    + "歡迎進入聊天室\n".encode())
132.                    threading.Thread(target=self.receive_text, daemon=True).start()
133.                    threading.Thread(target=self.start_image_server, args=(addr,), daemon=True).start()
134.                else:
135.                    identifier = f"{addr[0]}:{addr[1]}"
136.                    self.waiting_clients.put((conn, addr))
137.                    self.waiting_addrs.append(identifier)
138.                    self.update_waiting_label()
139.                    pos = self.waiting_clients.qsize()
140.                    try:
141.                        msg = f"您是第 {pos} 位等待中，請稍候...\n".encode()
142.                        conn.sendall((len(msg).to_bytes(4, 'big') + msg))
143.                    except:
144.                        conn.close()
145.

```

```

146.             # monitor_queue_socket(): 監控 client 端是否在排隊時中斷連線
147.             # 原理是偵測與該 client 連線的 socket 通道是否有中斷，若中斷即代表離開等待連入
148.             server 佇列
149.             import select
150.             def monitor_queue_socket():
151.                 try:
152.                     while True:
153.                         rlist, _, _ = select.select([conn], [], [], 0.5)
154.                         if rlist:
155.                             # 確保 socket 有資料可讀才來 check
156.                             peek = conn.recv(1, socket.MSG_PEEK)
157.                             if not peek:
158.                                 break
159.                             except:
160.                                 pass
161.             # 一旦連線中斷就移除
162.             if (conn, addr) in list(self.waiting_clients.queue):
163.                 with self.waiting_clients.mutex:
164.                     self.waiting_clients.queue.remove((conn, addr))
165.                     self.waiting_addrs.remove(identifier)
166.                     self.update_waiting_label()
167.                     self.log(f"{identifier} 離開等待隊列。)\n", tag="system")
168.
169.             threading.Thread(target=monitor_queue_socket, daemon=True).start()
170.
171.         def queue_monitor():
172.             import time
173.             while True:
174.                 if self.text_conn is None and not self.waiting_clients.empty():
175.                     try:
176.                         #self.log_text.delete("0.0", tk.END) # 新連線清空聊天紀錄
177.                         conn, addr = self.waiting_clients.get(timeout=0.1)
178.                         identifier = f"{addr[0]}:{addr[1]}"
179.                         if identifier in self.waiting_addrs:
180.                             self.waiting_addrs.remove(identifier)
181.                             self.window.after(0, self.update_waiting_label)
182.                             self.window.after(0, lambda: handle_client(conn, addr))
183.                     except queue.Empty:
184.                         pass
185.                     time.sleep(0.2)
186.
187.             threading.Thread(target=queue_monitor, daemon=True).start()
188.             while True:
189.                 conn, addr = server_socket.accept()
190.                 threading.Thread(target=handle_client, args=(conn, addr), daemon=True).start()
191.
192.             # 當 client 連入時，建立與 client 端的圖片訊息傳輸連線
193.             def start_image_server(self, addr):
194.                 # 建立與 client 的 img socket 連線
195.                 img_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
196.                 img_server.bind((self.HOST, self.IMAGE_PORT))
197.                 img_server.listen(1)
198.                 self.log(f"等待 {addr} 的圖片連線中...\n", tag="system")
199.                 conn, _ = img_server.accept()
200.                 self.image_conn = conn
201.                 self.log(f"{addr} 圖片 socket 已連接\n", tag="info")
202.
203.                 def handle_image_receive(sock):
204.                     while True:
205.                         try:
206.                             length_data = sock.recv(4)
207.                             if not length_data:
208.                                 break

```

```

209.             length = int.from_bytes(length_data, 'big')
210.             img_data = b''
211.             while len(img_data) < length:
212.                 chunk = sock.recv(length - len(img_data))
213.                 if not chunk:
214.                     break
215.                 img_data += chunk
216.             self.display_image(img_data, sender=f"Client ({addr[0]})")
217.         except:
218.             break
219.
220.     threading.Thread(target=handle_image_receive, args=(conn,), daemon=True).start()
221.
222.     # 文字訊息接收處理
223.     def receive_text(self):
224.         self.received_text = ""
225.         self.received_image_pending = False
226.         while True:
227.             # 流程:
228.             # 1. 每段文字訊息都會先傳送第一段內容表示接下來訊息的長度
229.             # 2. 持續接收訊息直到超過長度
230.             # 不論多長的訊息都能進行傳輸
231.             try:
232.                 length_data = self.text_conn.recv(4)
233.                 if not length_data:
234.                     break
235.                 length = int.from_bytes(length_data, 'big')
236.                 data = b''
237.                 while len(data) < length:
238.                     chunk = self.text_conn.recv(length - len(data))
239.                     if not chunk:
240.                         break
241.                     data += chunk
242.                 if not data:
243.                     break
244.                 message = data.decode()
245.                 self.received_text = message
246.                 self.received_image_pending = True
247.
248.             # 延遲顯示直到圖片來，避免當圖文同時接收時訊息框連跳兩次傳送者標頭
249.             def flush_text():
250.                 if self.received_text:
251.                     self.log(self.received_text)
252.                     self.received_text = ""
253.                     self.received_image_pending = False
254.                     self.window.after(300, flush_text)
255.             except:
256.                 break
257.             self.log("(目前連線之 Client 已離線)\n", tag="system")
258.             self.text_conn = None
259.             self.image_conn = None
260.
261.     # 圖片訊息接收處理
262.     def receive_image(self):
263.         while True:
264.             try:
265.                 length_data = self.image_conn.recv(4)
266.                 if not length_data:
267.                     break
268.                 length = int.from_bytes(length_data, 'big')
269.                 img_data = b''
270.                 while len(img_data) < length:
271.                     chunk = self.image_conn.recv(length - len(img_data))
272.                     if not chunk:
273.                         break

```

```

274.                 img_data += chunk
275.                 self.display_image(img_data, sender=f"Client({self.client_addr[0]})")
276.             except:
277.                 break
278.
279.     # 從本地資料夾選取要傳送的圖片
280.     def select_image(self):
281.         filepath = filedialog.askopenfilename(title="選擇圖片",
282.                                             filetypes=[("Image files", "*.*")]
283.                                             if filepath:
284.                                                 with open(filepath, "rb") as f:
285.                                                     self.selected_image = f.read()
286.                                                 img = Image.open(io.BytesIO(self.selected_image)) # 透過 BytesIO 將讀入的圖片轉
287.                                                成 pillow 可處理的形式
288.                                                 img.thumbnail((200, 200))
289.                                                 photo = ImageTk.PhotoImage(img)
290.                                                 self.img_label.config(image=photo)
291.                                                 self.img_label.image = photo
292.
293.     # 處理送出訊息(圖片/文字)
294.     def send_message(self):
295.         msg = self.input_text.get("1.0", tk.END).strip()
296.         sent_text = False
297.         sent_image = False
298.
299.         # 根據目前狀況(是否有文字輸入/圖片選擇)送出訊息
300.         if self.text_conn:
301.             if msg:
302.                 full_msg = f"Server({self.local_ip}):{msg}\n"
303.                 try:
304.                     encoded_msg = full_msg.encode()
305.                     self.text_conn.sendall(len(encoded_msg).to_bytes(4, 'big') +
306.                                           encoded_msg)
307.                     sent_text = True
308.                 except:
309.                     self.log("[錯誤] 傳送失敗\n", tag="error")
310.                     self.input_text.delete("1.0", tk.END)
311.             # 處理圖片傳送
312.             if self.image_conn and self.selected_image:
313.                 try:
314.                     self.image_conn.sendall(len(self.selected_image).to_bytes(4, 'big') \
315.                                           + self.selected_image) # 送出圖片大小 byte+實際圖片
316.                         sent_image = True
317.                 except Exception as e:
318.                     self.log(f"[錯誤] 圖片傳送失敗: {e}\n", tag="error")
319.
320.             # 最後才來處理訊息框顯示，圖文任何一者成功就 log+顯示
321.             if sent_text or sent_image:
322.                 timestamp = datetime.now().strftime("[%H:%M:%S]")
323.                 sender = f"{timestamp} Server({self.local_ip}):"
324.                 self.log_text.insert(tk.END, sender + (" " + msg + "\n" if sent_text else "") +
325.                                     "")
326.                 if sent_image:
327.                     self.display_image(self.selected_image, sender="")
328.                     self.log(f"[圖片已送出 - {self.local_ip} (Server)]\n", tag="system")
329.                     self.log_text.see(tk.END)
330.
331.             # 送出後重置已選擇圖片
332.             self.selected_image = None
333.             self.img_label.config(image='')
334.             self.img_label.image = None

```

```

332.    # 圖片顯示前處理
333.    def display_image(self, img_bytes, sender="Client"):
334.        if self.received_text:
335.            self.log(self.received_text)
336.            self.received_text = ""
337.            sender = None
338.        self.received_image_pending = False
339.        try:
340.            img = Image.open(io.BytesIO(img_bytes))
341.            img.thumbnail((200, 200))
342.            photo = ImageTk.PhotoImage(img)
343.            self.log_image(sender, photo, img_bytes)
344.        except Exception as e:
345.            self.log(f"[錯誤] 圖片顯示失敗: {e}\n", tag="error")
346.
347.    # 在訊息框內顯示圖片
348.    def log_image(self, sender, photo, original_bytes):
349.        # 紀錄時間
350.        now = datetime.now().strftime("[%H:%M:%S]")
351.        if sender == "":
352.            sender = None
353.            self.log_text.insert(tk.END, f"\n")
354.        if sender:
355.            self.log_text.insert(tk.END, f"{now} {sender}:\n")
356.        img_widget = tk.Label(self.log_text, image=photo, cursor="hand2")
357.        img_widget.image = photo
358.        img_widget.bind("<Button-1>", lambda e: self.show_full_image(original_bytes))
359.        self.log_text.window_create(tk.END, window=img_widget)
360.        self.log_text.insert(tk.END, "\n")
361.        self.image_refs.append(photo)
362.        self.log_text.see(tk.END)
363.
364.    # 點擊訊息框內的圖片可放大檢視
365.    def show_full_image(self, img_bytes):
366.        try:
367.            img = Image.open(io.BytesIO(img_bytes))
368.            top = Toplevel(self.window)
369.            top.title("圖片預覽")
370.            width, height = img.size
371.            top.geometry(f"{width}x{height}")
372.
373.            # 使用 canvas+toplevel 模塊來額外彈出視窗顯示原圖片
374.            photo = ImageTk.PhotoImage(img)
375.            canvas = Canvas(top, width=width, height=height)
376.            canvas.pack()
377.            canvas.create_image(0, 0, anchor=tk.NW, image=photo)
378.            canvas.image = photo
379.        except:
380.            messagebox.showerror("錯誤", "無法開啟圖片")
381.
382.    # 於聊天框內顯示訊息，透過 tag 區分顏色
383.    def log(self, msg, tag=None):
384.        # 紀錄時間
385.        now = datetime.now().strftime("[%H:%M:%S]")
386.        msg = f"{now} {msg}"
387.
388.        if tag and tag not in self.log_text.tag_names():
389.            if tag == "error":
390.                self.log_text.tag_configure(tag, foreground="red")
391.            elif tag == "info":
392.                self.log_text.tag_configure(tag, foreground="blue")
393.            elif tag == "system":
394.                self.log_text.tag_configure(tag, foreground="gray")
395.            else:

```

```

396.             self.log_text.tag_configure(tag, foreground="black")
397.
398.         if tag:
399.             self.log_text.insert(tk.END, msg, tag)
400.         else:
401.             self.log_text.insert(tk.END, msg)
402.         self.log_text.see(tk.END)
403.
404.
405.     # server 會保存文字聊天紀錄
406.     try:
407.         with open(self.log_file_path, "a", encoding="utf-8") as f:
408.             f.write(msg)
409.     except Exception as e:
410.         print(f"[log 寫入失敗]: {e}")
411.
412.     # 開啟聊天紀錄檔案資料夾
413.     def open_log_folder(self):
414.         log_path = os.path.abspath("chat_logs")
415.         try:
416.             # 根據作業系統選擇開啟檔案資料夾的執行指令
417.             if os.name == 'nt': # Windows
418.                 subprocess.Popen(f'explorer "{log_path}"')
419.             elif os.name == 'posix': # macOS / Linux
420.                 subprocess.Popen(['xdg-open', log_path])
421.         except Exception as e:
422.             self.log(f"[錯誤] 無法開啟資料夾: {e}", tag="error")
423.
424.     # 結束程式按鈕對應操作(關閉所有連線並關閉程式)
425.     def close_server(self):
426.         if self.text_conn:
427.             try: self.text_conn.close()
428.             except: pass
429.         if self.image_conn:
430.             try: self.image_conn.close()
431.             except: pass
432.         self.log("\n伺服器已關閉。\\n")
433.         self.window.destroy()
434.
435.     def run(self):
436.         self.window.mainloop()
437.
438.if __name__ == '__main__':
439.    ChatServer().run()

```

## chat\_ftpc.py

```

1. import socket
2. import threading
3. import tkinter as tk
4. from tkinter.scrolledtext import ScrolledText
5. from tkinter import simpledialog, messagebox, filedialog, Toplevel, Canvas
6. from PIL import Image, ImageTk
7. import io
8. from datetime import datetime
9.
10. class ChatClient:
11.     def __init__(self):
12.         self.server_ip =
13.         self.server_text_port = 10000
14.         self.server_image_port = 10001
15.         self.text_socket = None
16.         self.image_socket = None

```

```
17. self.local_ip = self.get_local_ip()
18. self.image_refs = []
19. self.selected_image = None
20.
21. self.setup_gui()
22.
23. # 自動抓取本地 IP 位址
24. def get_local_ip(self):
25.     s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
26.     try:
27.         s.connect(('10.255.255.255', 1))
28.         ip = s.getsockname()[0]
29.     except:
30.         ip = '127.0.0.1'
31.     finally:
32.         s.close()
33.     return ip
34.
35. # Client GUI 畫面建立
36. def setup_gui(self):
37.     self.window = tk.Tk()
38.     self.window.title("TCP Chat Client")
39.     self.window.geometry("600x600")
40.
41.     self.window.grid_rowconfigure(1, weight=1)
42.     self.window.grid_columnconfigure(0, weight=1)
43.
44.     # 上層: 顯示本機 IP, 輸入 Server IP 與 Port
45.     top_frame = tk.Frame(self.window)
46.     top_frame.grid(row=0, column=0, sticky="ew")
47.     top_frame.grid_columnconfigure(1, weight=1)
48.     top_frame.grid_columnconfigure(3, weight=1)
49.
50.     tk.Label(top_frame, text="本機 IP:").grid(row=0, column=0, sticky="w", padx=5)
51.     tk.Label(top_frame, text=self.local_ip).grid(row=0, column=1, sticky="w")
52.
53.     tk.Label(top_frame, text="Server IP:").grid(row=1, column=0, sticky="w", padx=5)
54.     self.server_ip_entry = tk.Entry(top_frame)
55.     self.server_ip_entry.insert(0, "127.0.0.1")
56.     self.server_ip_entry.grid(row=1, column=1, sticky="ew", padx=5)
57.
58.     tk.Label(top_frame, text="Port:").grid(row=1, column=2, sticky="w", padx=5)
59.     self.server_port_entry = tk.Entry(top_frame)
60.     self.server_port_entry.insert(0, "10000")
61.     self.server_port_entry.grid(row=1, column=3, sticky="ew", padx=5)
62.
63.     self.connect_button = tk.Button(top_frame, text="連線", command=self.connect)
64.     self.connect_button.grid(row=1, column=4, padx=5)
65.     tk.Button(top_frame, text="中斷連線", command=self.disconnect).grid(row=1, column=5,
66.     padx=5)
66.
67.     self.server_ip = '' # 清除舊值
68.     self.server_text_port = 10000
69.
70.     # 中間訊息視窗
71.     self.log_text = ScrolledText(self.window, width=50, height=20)
72.     self.log_text.grid(row=1, column=0, sticky="nsew")
73.
74.     # 下層輸入與傳送
75.     bottom_frame = tk.Frame(self.window)
76.     bottom_frame.grid(row=2, column=0, sticky="ew", pady=5)
77.     bottom_frame.grid_columnconfigure(1, weight=1)
78.
```

```
79. tk.Button(bottom_frame, text="選擇圖片", command=self.select_image).grid(row=0, column=0,
   padx=5)
80. self.input_text = tk.Text(bottom_frame, height=3)
81. self.input_text.grid(row=0, column=1, sticky="ew")
82. tk.Button(bottom_frame, text="傳送", command=self.send_message).grid(row=0, column=2,
   padx=5)
83.
84. self.img_label = tk.Label(self.window)
85. self.img_label.grid(row=3, column=0, pady=5)
86.
87. # client 連線按鈕操作(嘗試和輸入欄位位址之 Server IP 和 Port 連線)
88. def connect(self):
89.     self.server_ip = self.server_ip_entry.get().strip()
90.     try:
91.         self.server_text_port = int(self.server_port_entry.get())
92.     except:
93.         self.log("[錯誤] 請輸入有效的 Port 編號\n", tag="error")
94.     return
95.     try:
96.         self.connect_button.config(state="disabled")
97.         self.text_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
98.         self.text_socket.connect((self.server_ip, self.server_text_port))
99.
100. self.log(f"已連線到 Server {self.server_ip}:{self.server_text_port}\n", tag="info")
101. threading.Thread(target=self.receive_text, daemon=True).start()
102. except Exception as e:
103.     self.log(f"[錯誤] 無法連線到 Server: {e}\n", tag="error")
104.
105. # 文字訊息接收處理
106. def receive_text(self):
107.     while True:
108.         # 流程:
109.         # 1. 每段文字訊息都會先傳送第一段內容表示接下來訊息的長度
110.         # 2. 持續接收訊息直到超過長度
111.         # 不論多長的訊息都能進行傳輸
112.         try:
113.             length_data = self.text_socket.recv(4)
114.             if not length_data:
115.                 break
116.             length = int.from_bytes(length_data, 'big')
117.             data = b''
118.             while len(data) < length:
119.                 chunk = self.text_socket.recv(length-len(data))
120.                 if not chunk:
121.                     break
122.                 data += chunk
123.             if not data:
124.                 break
125.             message = data.decode()
126.             self.received_text = message
127.             self.received_image_pending = True
128.
129.         # 延遲顯示直到圖片來，避免當圖文同時接收時訊息框連跳兩次傳送者標頭
130.         def flush_text():
131.             if self.received_text:
132.                 self.log(self.received_text)
133.                 self.received_text = ""
134.             self.received_image_pending = False
135.             self.window.after(300, flush_text)
136.
137.         if ("已連線" in message or "歡迎進入聊天室" in message) and not self.image_socket:
138.             try:
139.                 self.image_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
140.self.image_socket.connect((self.server_ip, self.server_image_port))
141.threading.Thread(target=self.receive_image, daemon=True).start()
142.# self.log("圖片通道已建立\n", tag="system")
143.exception as e:
144.self.log(f"[錯誤] 圖片連線失敗: {e}\n", tag="error")
145.exception:
146.break
147.self.connect_button.config(state="normal")
148.self.log("(與 server 連線已中斷)\n", tag="system")
149.

150.# 圖片訊息接收處理
151.def receive_image(self):
152.while True:
153.try:
154.length_data = self.image_socket.recv(4)
155.if not length_data:
156.break
157.length = int.from_bytes(length_data, 'big')
158.img_data = b''
159.while len(img_data) < length:
160.chunk = self.image_socket.recv(length - len(img_data))
161.if not chunk:
162.break
163.img_data += chunk
164.self.display_image(img_data, sender=f"Server ({self.server_ip})")
165.exception:
166.break
167.

168.# 從本地資料夾選取要傳送的圖片
169.def select_image(self):
170.filepath = filedialog.askopenfilename(title="選擇圖片",
171.filetypes=[("Image files", "*.*")]
172.if filepath:
173.with open(filepath, "rb") as f:
174.self.selected_image = f.read()
175.img = Image.open(io.BytesIO(self.selected_image))
176.img.thumbnail((200, 200))
177.photo = ImageTk.PhotoImage(img)
178.self.img_label.config(image=photo)
179.self.img_label.image = photo
180.

181.# 處理送出訊息(圖片/文字)
182.def send_message(self):
183.msg = self.input_text.get("1.0", tk.END).strip()
184.sent_text = False
185.sent_image = False
186.
187.if self.text_socket:
188.if msg:
189.full_msg = f"Client({self.local_ip}):{msg}\n"
190.try:
191.encoded_msg = full_msg.encode()
192.self.text_socket.sendall(len(encoded_msg).to_bytes(4, 'big') + encoded_msg)
193.sent_text = True
194.exception:
195.self.log("[錯誤] 傳送失敗\n", tag="error")
196.self.input_text.delete("1.0", tk.END)
197.
198.if self.image_socket and self.selected_image:
199.try:
200.self.image_socket.sendall(len(self.selected_image).to_bytes(4, 'big') +
self.selected_image)
201.sent_image = True
202.exception as e:
```

```
203.self.log(f"[錯誤] 圖片傳送失敗: {e}\n", tag="error")
204.
205.# 最後才來處理訊息框顯示，圖文任何一者成功就 log+顯示
206.if sent_text or sent_image:
207.timestamp = datetime.now().strftime("[%H:%M:%S]")
208.sender = f"{timestamp} Client({self.local_ip}):"
209.self.log_text.insert(tk.END, sender + (" " + msg + "\n" if sent_text else "") + "")
210.if sent_image:
211.self.display_image(self.selected_image, sender="")
212.self.log(f"[圖片已送出 - {self.local_ip} (Client)]\n", tag="system")
213.self.log_text.see(tk.END)
214.# 送出後重置已選擇圖片
215.self.selected_image = None
216.self.img_label.config(image='')
217.self.img_label.image = None
218.
219.# 圖片顯示前處理
220.def display_image(self, img_bytes, sender="Server"):
221.if self.received_text:
222.self.log(self.received_text)
223.self.received_text = ""
224.sender = None
225.self.received_image_pending = False
226.try:
227.img = Image.open(io.BytesIO(img_bytes))
228.img.thumbnail((200, 200))
229.photo = ImageTk.PhotoImage(img)
230.self.log_image(sender, photo, img_bytes)
231.exception as e:
232.self.log(f"[錯誤] 圖片顯示失敗: {e}\n", tag="error")
233.
234.# 在訊息框內顯示圖片
235.def log_image(self, sender, photo, original_bytes):
236.# 紀錄時間
237.now = datetime.now().strftime("[%H:%M:%S]")
238.if sender == "":
239.sender = None
240.self.log_text.insert(tk.END, f"\n")
241.if sender:
242.self.log_text.insert(tk.END, f"{now} {sender}:\n")
243.# self.log_text.insert(tk.END, f"{now} {sender}:\n")
244.img_widget = tk.Label(self.log_text, image=photo, cursor="hand2")
245.img_widget.image = photo
246.img_widget.bind("<Button-1>", lambda e: self.show_full_image(original_bytes))
247.self.log_text.window_create(tk.END, window=img_widget)
248.self.log_text.insert(tk.END, "\n")
249.self.image_refs.append(photo)
250.self.log_text.see(tk.END)
251.
252.# 點擊訊息框內的圖片可放大檢視
253.def show_full_image(self, img_bytes):
254.try:
255.img = Image.open(io.BytesIO(img_bytes))
256.top = Toplevel(self.window)
257.top.title("圖片預覽")
258.width, height = img.size
259.top.geometry(f"{width}x{height}")
260.
261.photo = ImageTk.PhotoImage(img)
262.canvas = Canvas(top, width=width, height=height)
263.canvas.pack()
264.canvas.create_image(0, 0, anchor=tk.NW, image=photo)
265.canvas.image = photo
266.exception:
```

```
267.messagebox.showerror("錯誤", "無法開啟圖片")
268.
269.# 於聊天框內顯示訊息，透過 tag 區分顏色
270.def log(self, msg, tag=None):
271.# 紀錄時間
272.now = datetime.now().strftime("[%H:%M:%S]")
273.msg = f"{now} {msg}"
274.if tag and tag not in self.log_text.tag_names():
275.if tag == "error":
276.self.log_text.tag_configure(tag, foreground="red")
277.elif tag == "info":
278.self.log_text.tag_configure(tag, foreground="blue")
279.elif tag == "system":
280.self.log_text.tag_configure(tag, foreground="gray")
281.else:
282.self.log_text.tag_configure(tag, foreground="black")
283.
284.if tag:
285.self.log_text.insert(tk.END, msg, tag)
286.else:
287.self.log_text.insert(tk.END, msg)
288.self.log_text.see(tk.END)
289.
290.# 中斷連線按鈕對應操作(中斷目前 client 對 server 連線)
291.def disconnect(self):
292.if self.text_socket:
293.try: self.text_socket.close()
294.except: pass
295.self.text_socket = None
296.if self.image_socket:
297.try: self.image_socket.close()
298.except: pass
299.self.image_socket = None
300.self.connect_button.config(state="normal")
301.
302.def run(self):
303.self.window.mainloop()
304.
305.if __name__ == '__main__':
306.ChatClient().run()
```