IS301 课程大作业　聊天程序

项 目 文 档

刘子涵 518021910690

1 概述

本项目基于 Python 语言实现了网络聊天程序的服务端和客户端，实现的功能包括：**多用户一对一私聊、群组聊天、文件传输、使用表情包**。

- **运行环境：**
  - 服务端：阿里云服务器 Ubuntu 20.04 / 个人电脑 Windows 10 Home
  - 客户端：个人电脑 Windows 10 Home
- **编译工具：**
  - 阿里云服务器 Ubuntu 20.04：Python 3.8.5 + PyInstaller 4.1
  - 个人电脑：Python 3.7.3 + PyInstaller 3.6
  - PyInstaller 打包为 exe 命令

```
// ChatClient-win
pyinstaller -D -w -i ./package/icon_client.ico ChatClient.py
--add-data .\package\tkdnd2.8;tkdnd --add-data .\emoji;emoji
```

```
// ChatServer-win/linux
pyinstaller -D -i ./package/icon_server.ico ChatServer.py
```

- **程序文件列表：**

```
.
├── bin
│   ├── ChatClient-win_x86_64
│   │   ├── ChatClient.exe // 客户端可执行程序（win）
│   │   └── ......
│   ├── ChatServer-linux_x86_64
│   │   ├── ChatServer // 服务端可执行程序（linux）
│   │   └── ......
│   └── ChatServer-win_x86_64
│       ├── ChatServer.exe // 服务端可执行程序（win）
│       └── ......
└── src
    ├── ChatClient.py  // 客户端源程序
    ├── ChatServer.py  // 服务端源程序
    ├── emoji  // 表情包图片
    └── package
        ├── tkdnd2.8 // Tk 拖拽功能拓展包
        └── TkDnD.py // 使用 tkdnd 构造的中间脚本
```

2 算法实现

本项目基于 Python socket 库建立客户端和服务器之间的可靠 TCP 连接，而后通信双方利用该连接实现网络数据流传输。

客户端和服务器之间的交互过程为：

- 服务端程序的主线程侦听指定端口（如 6666），客户端利用服务器的 IP 和相应端口号主动连接服务器，服务端程序 fork 出子线程用于处理与该客户端的连接，并接收后续该客户端发来的数据。

- TCP 连接建立后，客户端处理用户请求，并将生成的数据通过建立好的连接发送至服务器，服务端子线程接收到客户端发来的数据后，将其放入**消息队列**中等待转发。这一过程为保障线程安全，多线程操作数据修改不会混乱，使用**可重入锁**，只有获得锁的线程才能将消息放入队列。

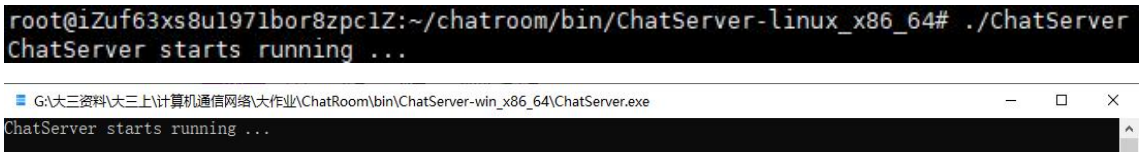
- 处理转发数据的工作由一个**独立的子线程**来完成，它将消息队列中的数据取出，转发至所有客户端。

- 客户端收到服务端发来的数据后进行处理，处理结果响应在图形界面上。注意，传输的数据为二进制比特流，其中的内容主要分为两种形式，一种是字符串，字符串中有多种标志位（如$;$）用于标识各类信息；另一种是在线用户列表，这里需要将列表序列化为 JSON 格式后才能进行传输。

另外，项目中比较繁琐的是客户端 GUI 的设计与实现，需要反复调节各组件的位置、大小、颜色等。这里 GUI 的实现利用的是 Python tkinter 库及其拓展库。值得一提的是，在文件传输功能的实现中，参考了当下主流聊天程序的功能——可以将需要传输的文件由桌面拖到界面框中。这一步的实现依赖拓展工具包 TkDND，并编写一个 TkDnD 类充当调用的接口（TkDnD.py）。

3 程序测试截图及说明

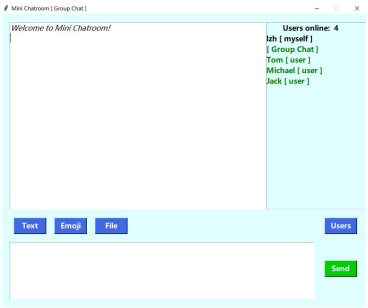
① 运行服务端程序，侦听指定端口 6666。

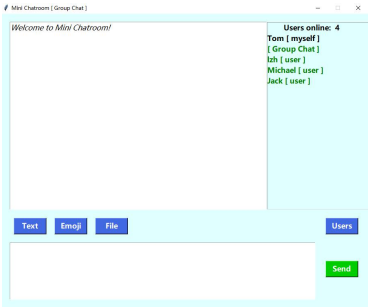


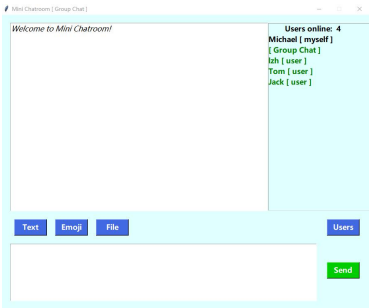
注：以上第一张图为在阿里云 Ubuntu 服务器运行（需要配置安全组开放 6666 端口），第二张图为在个人 Win10 电脑上运行。由于两者测试效果相同，后续仅展示前者的截图。

② 运行客户端程序，输入服务器 IP、端口号、用户名并登录。

③ 在个人电脑上开启四个客户端程序进程用于测试。







点击按钮 Users 可以展开和折叠右侧的在线用户栏，可以看到栏中显示当前在线用户数为4，以及所有用户的用户名，其中绿色的项目表示可以聊天的对象，点击 [Group Chat] 即群聊，点击某一用户则私聊。

左侧的三个按钮中 Text 表示发送文本，Emoji 表示发送表情包，File 表示发送文件。默认显示的界面是发送文本。以下先以文本为例，展示多用户一对一私聊和群聊功能。

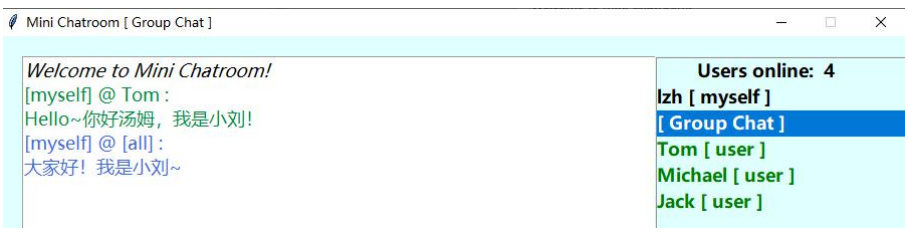
④ 多用户一对一私聊

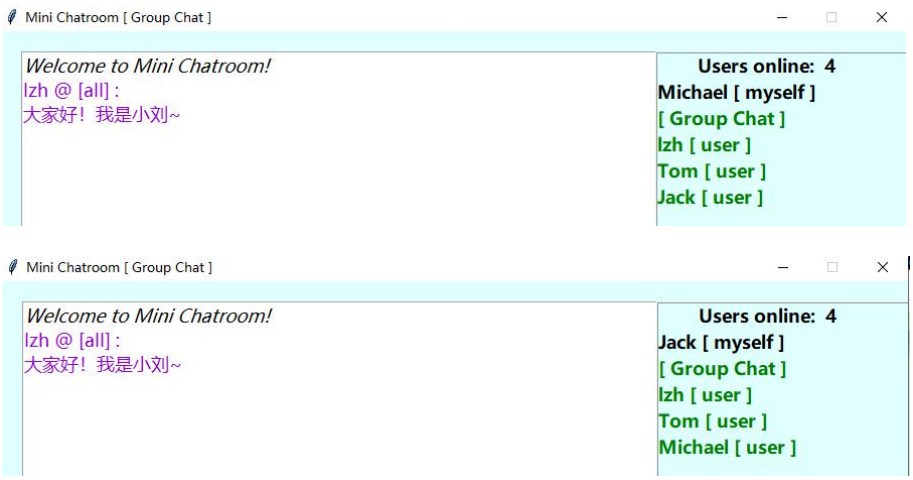
测试：用户 lzh 向 Tom 发送私聊文本消息，其余两用户接收不到任何消息。

从截图可以看出，私聊时发送方显示消息为绿色，接收方显示消息为红色，其余用户无法接收到该消息。

⑤ 群组聊天

测试：用户 lzh 向所有人发送文本消息

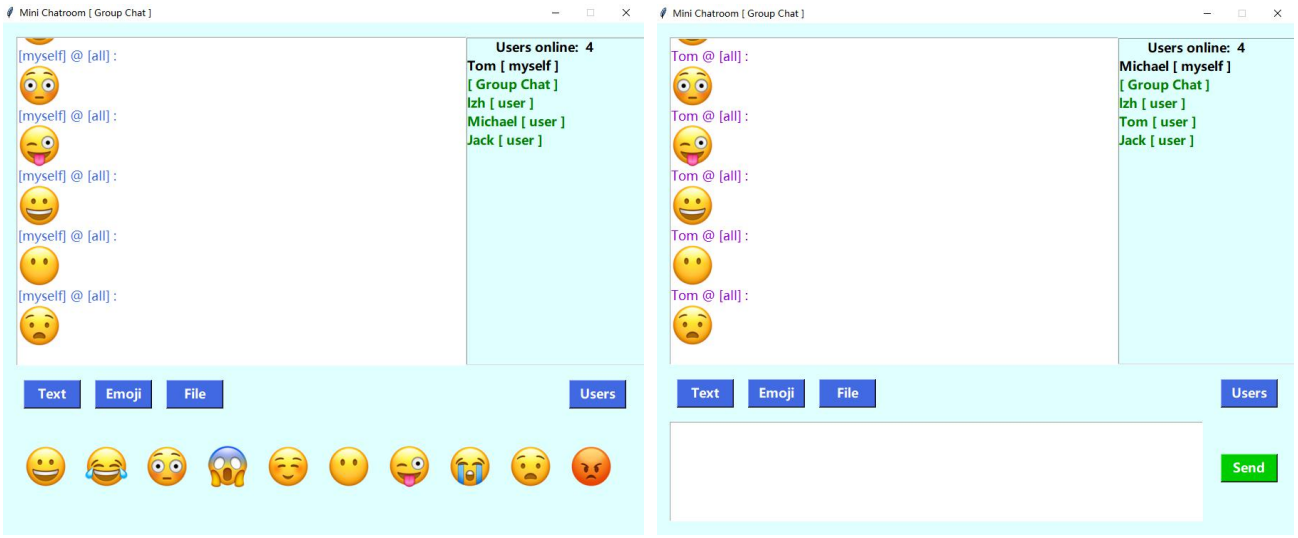




从截图可以看出，群聊时发送方显示消息为蓝色，接收方显示消息为紫色，除发送方外其余用户均能接收到该消息。

⑥ 发送表情包

测试：用户 Tom 群聊发送表情包

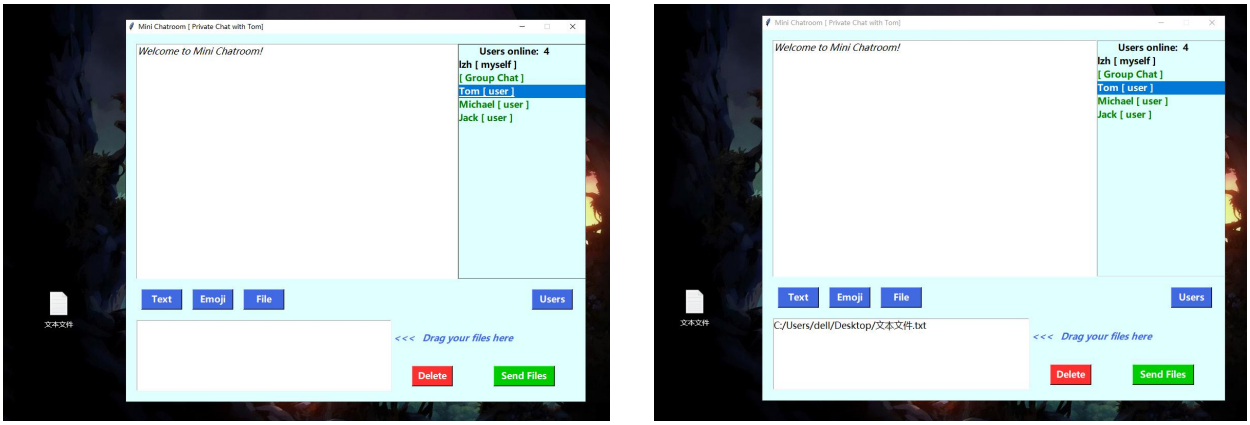


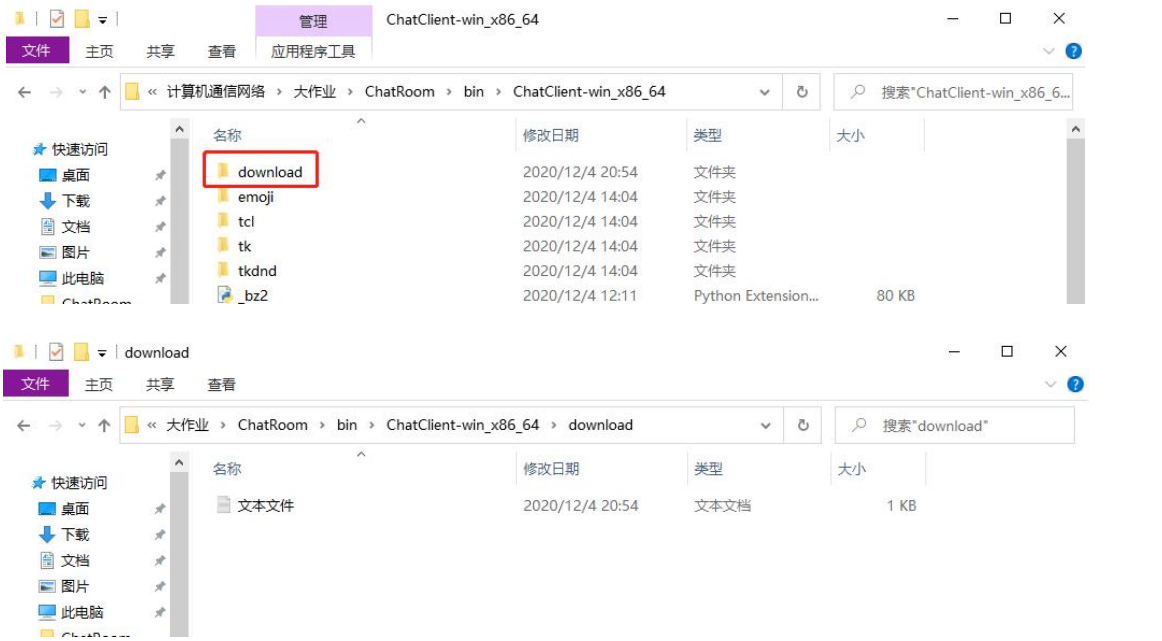
⑦ 文件传输

测试：用户 lzh 给 Tom 私发各种文件

（1）文本文件

测试样例：文本文件.txt（48B）



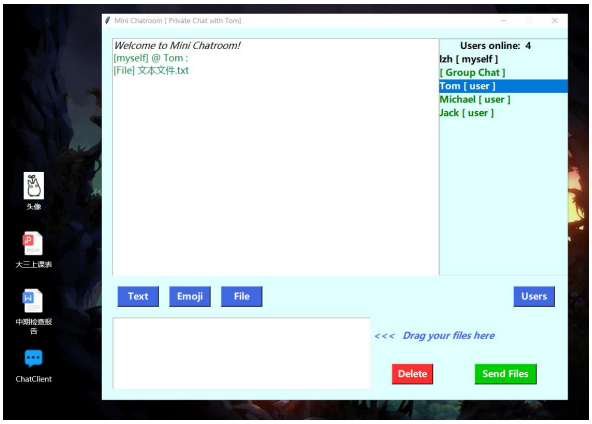


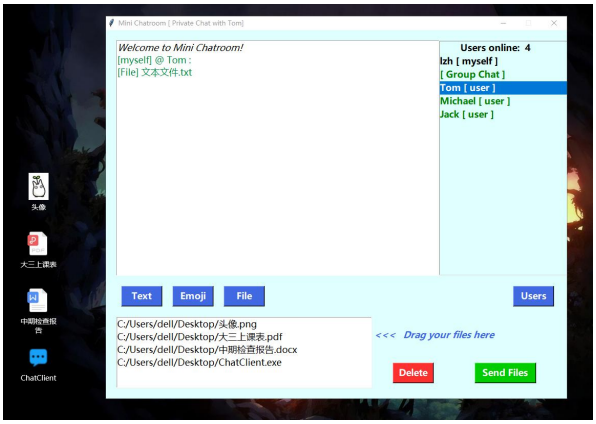
从截图可以看出，传输文件仅需将文件从桌面拖到界面框中，如果放弃发送某一文件，则选中后点击 Delete 按钮即可删除，点击 Send Files 发送文件。成功发送后，发送方和接收方均以相应颜色显示发送文件名。接收方会在 ChatClient.exe 同级目录下建立新目录 download/用于存放接收到的文件，可见 Tom 成功接收到 lzh 发送的 文本文件.txt 。

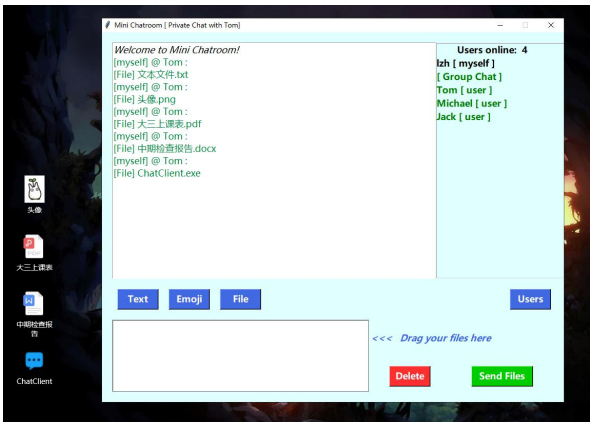
（2）非文本文件

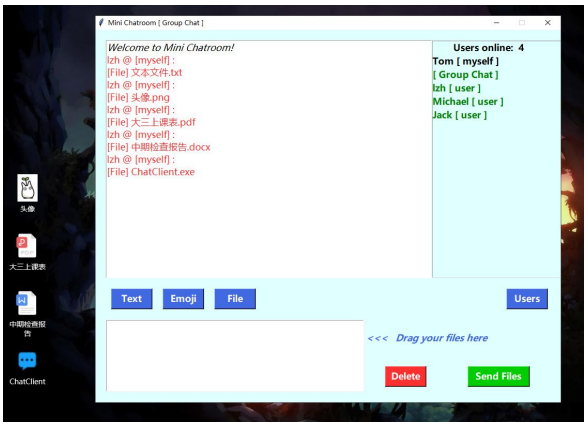
测试样例：

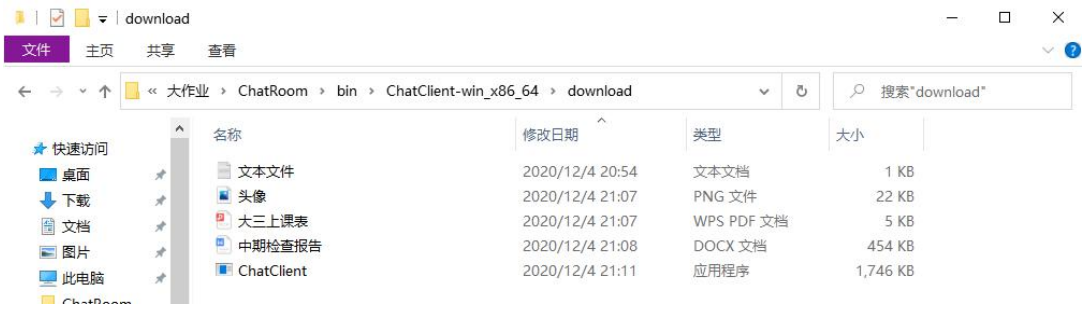
① 头像.png（21KB）
② 大三上课表.pdf（4KB）
③ 中期检查报告.docx（453KB）
④ ChatClient.exe（1.71MB）











在测试过程中，较小的文件如①②能在较短时间内发送，较大的文件如③④需要等待较长时间才能发送。因为程序中设定套接字每发送 1KB 即暂停 0.1s（time.sleep(0.1)），所以文件④这样大小的文件在实际发送中需要至少 3min 左右。但是如果将等待时间大幅调小甚至不等待，数据传输容易发生错误。如何在保证数据传输无误的前提下提高传输效率，是聊天程序今后的一大改进点。

4 遇到的问题及解决方法

① 问题：Tkinter 输入文本框无法输入多行文本，输入的字符只能呈现在一行内。

解决：更换组件，将 tkinter.Entry 更换为 tkinter.Text。

② 问题：Tkinter 点击功能切换按钮（Text/Emoji/File），原功能界面组件使用了 place_forget() 也并未隐藏，多次点击后甚至无法切换功能。

解决：增加状态变量 status，Text/Emoji/File 功能分别对应 0/1/2，当功能切换时，根据当前状态决定动作。

③ 问题：Tkinter 中未点击按钮却自动执行 command 函数。

解决：出现此问题是因为 command 函数带有参数，而 Tkinter 要求按钮触发的控制器函数不能含有参数，若要给函数传递参数，需要使用匿名函数 lambda。

（参考：https://blog.csdn.net/guge907/article/details/23291763）

④ 问题：无法传输非文本文件。

原因：按程序执行逻辑顺序进行调试，发现服务器端程序对所有二进制数据直接 decode，而非文本文件的二进制流直接 decode 会报 UnicodeDecodeError。由于接收函数是在 try-except 块中，程序捕获到异常后直接进入 except 块与客户端断开连接，客户端异常终止，报错“ConnectionResetError: [WinError 10054] 远程主机强迫关闭了一个现有的连接”，该错误较为隐蔽。

解决：服务器端程序改为直接接收并转发二进制数据，消息队列也存放二进制数据。

⑤ 问题：生成客户端可执行程序 ChatClient.exe 后，运行过程中弹出错误提示窗口“Failed to execute script ChatClient”。

解决：取消打包 ChatClient.exe 时的-w 参数，从而显示运行窗口。发现窗口中报错找不到 emoji 文件夹和 tkdnd 文件夹。通过--add-data 参数添加依赖的文件夹即可。

（参考：https://www.cnblogs.com/littlelong/p/11120604.html）

```
// ChatClient-win
pyinstaller -D -w -i ./package/icon_client.ico ChatClient.py --add-data .\package\tkdnd2.8;tkdnd
--add-data .\emoji;emoji
```

5 体会与建议

本次大作业我选择开发聊天程序，因为个人对网络编程和软件开发比较感兴趣。该项目的主要任务是利用 Python 提供的 socket 调用接口实现网络数据通信，实现了多用户私聊、群聊，发送文本、表情、文件等功能。该项目的难点主要在于协调服务器与客户端之间的通信、服务端多线程处理、客户端 GUI 设计等。通过该项目的实践，我对计算机通信网络中的网络层、传输层通信原理加深了理解。

项目开展的全流程可谓困难重重：从前期学习 Python 的 socket 库和 tkinter 库，再到中期引入多线程和线程管理、消息队列、线程锁、JSON 序列化等思想到项目实现中，这些概念之前从未接触过，最后到利用 Tkinter 及其拓展完善客户端 GUI 设计，这也是耗费时间最长的一部分，因为这一步需要反复调节各组件的参数以及调试各种错误。但是，在发现问题、寻求解决方案的过程中，我查阅大量资料，与同学沟通，最终解决问题让我非常有成就感。

此次实验提高了我的编程实战能力，并且第一次接触了 GUI 编程。另外，我对操作系统的知识也有了更多的了解。

总之，通过此次大作业，我获益良多。非常感谢老师提供的帮助和指导！

附录：源程序

① 服务端源程序 ChatServer.py

```
# ChatServer.py

import socket
import threading
import queue
import json
import time
import sys

PORT = 6666
msgque = queue.Queue()  # 消息队列：用于存储转发所有客户端发来的消息至其它已连接的客户端
lock = threading.RLock() # 可重入锁：保证线程安全，多线程操作数据修改不会混乱
users = []  # 在线用户信息列表：包括 (conn, user, addr)


''' 当前在线用户列表，需要在每个客户端实时更新 '''
def onlines():
    online_list = []
    for each in users:
        online_list.append(each[1])
    return json.dumps(online_list).encode() # 将 list 序列化为 JSON 格式并转换为二进制


class ChatServer(threading.Thread):
    global users, que, lock

    def __init__(self, port):
        threading.Thread.__init__(self)
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


    ''' tcp_connect：处理和维持 TCP 连接，接收客户端发送的消息 '''
    def tcp_connect(self, conn, addr):
        user = conn.recv(1024).decode()  # 接收用户名 user

        # 处理用户同名情况
        repeat_times = 0
        for each in users:
            if user == each[1]:
                repeat_times += 1
```

```
        if repeat_times != 0:
            user = user + '_' + str(repeat_times)

        # 用户没有输入用户名，默认 IP:port 作为用户名
        if user == 'default':
            user = addr[0] + ':' + str(addr[1])

        # 将当前用户添加到在线用户列表
        users.append((conn, user, addr))

        print('Connect from user [', user, ']', addr)

        # 刷新所有客户端的在线用户显示
        self.recv(onlines())

        # 维持 TCP 连接，接收客户端发送信息
        try:
            while True:
                data = conn.recv(1024)  # 每次最多接收 1kB
                self.recv(data)
            conn.close()
        except:
            print('Disconnect from user [', user, ']', addr)
            self.delUser(user)    # 将断开连接的用户从 users 列表中删除
            conn.close()


    ''' delUser：将断开连接的用户从 users 列表中删除，更新在线用户显示 '''
    def delUser(self, user):
        for i in range(len(users)):
            if user == users[i][1]:
                users.pop(i)
                self.recv(onlines())
                break


    ''' recv：将从客户端接收到的消息存入消息队列，等待转发至各个客户端 '''
    def recv(self, msg):
        lock.acquire()
        try:
            msgque.put(msg)
        finally:
            lock.release()
```

```
    ''' forward：由一个线程来处理，将队列中的消息转发给所有客户端 '''
    def forward(self):
        while True:
            if not msgque.empty():
                msg = msgque.get()
                for each in users:
                    each[0].send(msg)


    ''' run：运行服务（重写 Thread 类 run 方法） '''
    def run(self):
        self.sock.bind(('0.0.0.0', PORT))  # 监听指定端口 PORT
        self.sock.listen(5)   # 开始监听，等待连接的最大数量为 5
        print('ChatServer starts running ...')
        q = threading.Thread(target=self.forward)  # 创建线程处理数据转发
        q.start()
        while True:
            conn, addr = self.sock.accept()    # 接受一个新连接
            t = threading.Thread(target=self.tcp_connect, args=(conn, addr))    # 处理新的 TCP 连接
            t.start()
        self.sock.close()


if __name__ == '__main__':
    server = ChatServer(PORT)
    server.start()

    while True:
        time.sleep(1)
        if not server.is_alive():
            print("Chat connection lost...")
            sys.exit(0)
```

② 客户端源程序 ChatClient.py

```
# ChatClient.py

import socket
import threading
import tkinter
import tkinter.font as ft
import tkinter.messagebox
from tkinter.scrolledtext import ScrolledText
import json
import time
import os
import re

from package.TkDnD import TkDnD


IP = '' # 服务器 IP
PORT = ''   # 服务器 Port
user = ''   # 用户名
chatWith = '#@[GROUP]@#'  # 聊天对象，默认为群聊
users = []  # 在线用户列表


''' 登录窗口 '''
# 图形界面
loginWin = tkinter.Tk()
loginWin.title('Mini Chatroom')
loginWin.geometry('380x500')
loginWin.resizable(0, 0)  # 限制窗口大小
BgColor = '#E0FFFF'
textColor = '#4169E1'
loginWin['bg'] = BgColor

# 字体设置
titleFont = ft.Font(family='Times', size=30, weight=ft.BOLD, slant=ft.ITALIC)
textFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.BOLD, slant=ft.ROMAN)
inputFont = ft.Font(family='Microsoft YaHei', size=10, weight=ft.NORMAL, slant=ft.ROMAN)

# 标题标签
labelTitle = tkinter.Label(loginWin, text='Mini Chatroom', bg=BgColor, fg=textColor,
font=titleFont)
labelTitle.place(x=50, y=40)
```

```
# 服务器 IP 标签
serverIP = tkinter.StringVar()
serverIP.set('139.196.201.152') # 默认显示的服务器 IP
labelIP = tkinter.Label(loginWin, text='Server  IP', bg=BgColor, fg=textColor, font=textFont)
labelIP.place(x=140, y=130)
entryIP = tkinter.Entry(loginWin, width=80, textvariable=serverIP, font=inputFont)
entryIP.place(x=120, y=170, width=150, height=30)

# 服务器 Port 标签
serverPort = tkinter.StringVar()
serverPort.set('6666')  # 默认显示的服务器 Port
labelPort = tkinter.Label(loginWin, text='Server  Port', bg=BgColor, fg=textColor, font=textFont)
labelPort.place(x=130, y=210)
entryPort = tkinter.Entry(loginWin, width=80, textvariable=serverPort, font=inputFont)
entryPort.place(x=120, y=250, width=150, height=30)

# 用户名标签
userName = tkinter.StringVar()
userName.set('')    # 默认显示的用户名为空
labelUserName = tkinter.Label(loginWin, text='Username', bg=BgColor, fg=textColor, font=textFont)
labelUserName.place(x=140, y=290)
entryUserName = tkinter.Entry(loginWin, width=80, textvariable=userName, font=inputFont)
entryUserName.place(x=120, y=330, width=150, height=30)


# 登录按钮
def login(*args):
    global IP, PORT, user

    IP = entryIP.get()
    PORT = int(entryPort.get())
    user = entryUserName.get()
    if not user:
        tkinter.messagebox.showerror('ERROR', message='Empty Username!')
    else:
        loginWin.destroy()  # 关闭窗口


loginWin.bind('<Return>', login)    # 回车绑定 login 函数
loginButton = tkinter.Button(loginWin, text='Log in', command=login, bg=textColor, fg='white',
font=textFont, activebackground='#191970', activeforeground='white')      # 按钮绑定 login 函数
loginButton.place(x=150, y=400, width=80, height=40)

loginWin.mainloop()
```

```
''' 进行 TCP 连接 '''
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((IP, PORT))
addr = sock.getsockname()  # 获取客户端(IP,port)

if user:
    sock.send(user.encode())   # 发送用户名
else:
    sock.send('default'.encode())  # 没有输入用户名则标记 default
    user = addr[0] + ':' + str(addr[1]) # 用户没有输入用户名，默认 IP:port 作为用户名


''' 聊天窗口 '''
# 图形界面
mainWin = tkinter.Tk()
mainWin.title('Mini Chatroom [ Group Chat ]')
mainWin.geometry('900x720')
mainWin.resizable(0, 0)    # 限制窗口大小
mainWin['bg'] = BgColor

# 创建多行文本框
chatbox = ScrolledText(mainWin)
chatbox.place(x=20, y=20, width=860, height=460)

chatFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.NORMAL, slant=ft.ROMAN)
chatbox.configure(font=chatFont)
# 文本框使用的字体颜色
sysFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.NORMAL, slant=ft.ITALIC)
chatbox.tag_config('sys', foreground='black', font=sysFont) # 系统输出
chatbox.insert(tkinter.END, 'Welcome to Mini Chatroom!\n', 'sys')

chatbox.tag_config('mg', foreground='#4169E1')  # 自己@所有人
chatbox.tag_config('og', foreground='#9400D3') # 别人@所有人
chatbox.tag_config('mo', foreground='#008B45')    # 自己@别人
chatbox.tag_config('om', foreground='#FF3030') # 别人@自己


''' *** 功能实现 *** '''
# 当前功能 0:text(default) / 1:emoji / 2:file
status = 0

# 输入文本框（默认 place）
iptText = tkinter.Text(mainWin, width=120)
```

```
iptText.place(x=20, y=560, width=750, height=140)
iptFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.NORMAL, slant=ft.ROMAN)
iptText.configure(font=iptFont)

# 表情包选择按钮列表
selectButs = []

# 文件列表
filebox = tkinter.Listbox(mainWin, font=iptFont)

# 文件删除按钮
textFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.BOLD, slant=ft.ROMAN)
deleteBut = tkinter.Button(mainWin, text='Delete', command=lambda x=filebox:
x.delete(tkinter.ACTIVE), bg='#FF3030', fg='white', font=textFont, activebackground='#8B1A1A',
activeforeground='white')

# 拖动标志
textFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.BOLD, slant=ft.ITALIC)
dragLabel = tkinter.Label(mainWin, text='<<<   Drag your files here', bg=BgColor, fg=textColor,
font=textFont)



# 发送文件函数
def sendFile():
    global filebox

    for i in range(filebox.size()):
        filename = filebox.get(i)
        with open(filename, 'rb') as f:
            while True:
                data = f.read(1024)
                if not data:
                    break
                info = '$;$' + user + '$;$' + chatWith
                data = '#@[FILE]@#'.encode() + data + info.encode()
                sock.send(data)
                time.sleep(0.1)

        msg = '<EOF>' + '$;$' + user + '$;$' + chatWith
        sock.send(msg.encode())
        time.sleep(0.1)

        msg = '#@[ENDFILE]@#' + filename.split('/')[-1] + '$;$' + user + '$;$' + chatWith
        sock.send(msg.encode())
        time.sleep(0.1)
```

```
    filebox.delete(0, tkinter.END)    # 发送后清空 filebox


# 发送文件按钮
textFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.BOLD, slant=ft.ROMAN)
sendFileBut = tkinter.Button(mainWin, text='Send Files', command=sendFile, bg='#00CD00',
fg='white', font=textFont, activebackground='#008B00', activeforeground='white')


# 发送文本函数
def send(*args):
    global iptText

    users.append('#@[GROUP]@#')
    if chatWith not in users:
        tkinter.messagebox.showerror('ERROR', message='Cannot chat with nobody!')
        return
    if chatWith == user:
        tkinter.messagebox.showerror('ERROR', message='Cannot chat with yourself in private!')
        return

    msg = iptText.get(1.0, tkinter.END)
    if not msg.strip():
        tkinter.messagebox.showerror('ERROR', message='Cannot send nothing!')
        return

    msg = msg + '$;$' + user + '$;$' + chatWith
    sock.send(msg.encode())
    iptText.delete(1.0, tkinter.END)    # 发送后清空文本框


# 发送按钮（默认 place）
textFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.BOLD, slant=ft.ROMAN)
sendBut = tkinter.Button(mainWin, text='Send', command=send, bg='#00CD00', fg='white',
font=textFont, activebackground='#008B00', activeforeground='white')
sendBut.place(x=795, y=605, width=80, height=40)
mainWin.bind('<Return>', send)  # 绑定回车发送信息


# 选择输入文本功能
def inputText():
    global iptText, selectButs, filebox, deleteBut, status
```

```
    if status == 0:
        return
    elif status == 1:
        for i in range(10):
            selectButs.pop().destroy()
    elif status == 2:
        filebox.place_forget()
        deleteBut.place_forget()
        dragLabel.place_forget()
        sendFileBut.place_forget()

    iptText.place(x=20, y=560, width=750, height=140)
    sendBut.place(x=795, y=605, width=80, height=40)
    status = 0

# 文本按钮（默认 place）
textFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.BOLD, slant=ft.ROMAN)
textBut = tkinter.Button(mainWin, text='Text', command=inputText, bg=textColor, fg='white',
font=textFont, activebackground='#191970', activeforeground='white')
textBut.place(x=30, y=500, width=80, height=40)


# 表情包图片列表
emojis = [tkinter.PhotoImage(file='./emoji/'+f) for f in os.listdir('./emoji')] # 打开表情包图片，
存入 emojis 列表中


# 发送表情包函数，由选择按钮 selectButs[i]触发
def sendEmoji(i):
    msg = '#@[EMOJI]@#' + str(i) + '$;$' + user + '$;$' + chatWith
    sock.send(msg.encode())


# 选择表情包函数，由表情包按钮 emojiBut 触发
def selectEmoji():
    global selectButs, iptText, sendBut, filebox, deleteBut, status

    if status == 1:
        return
    elif status == 0:
        iptText.place_forget()
        sendBut.place_forget()
    elif status == 2:
        filebox.place_forget()
        deleteBut.place_forget()
```

```
        dragLabel.place_forget()
        sendFileBut.place_forget()

    # 不能用循环变量生成 button，command 传参有误！
    selectButs.append(tkinter.Button(mainWin, command=lambda:sendEmoji(0), image=emojis[0],
relief=tkinter.FLAT, bd=0, bg=BgColor))
    selectButs.append(tkinter.Button(mainWin, command=lambda:sendEmoji(1), image=emojis[1],
relief=tkinter.FLAT, bd=0, bg=BgColor))
    selectButs.append(tkinter.Button(mainWin, command=lambda:sendEmoji(2), image=emojis[2],
relief=tkinter.FLAT, bd=0, bg=BgColor))
    selectButs.append(tkinter.Button(mainWin, command=lambda:sendEmoji(3), image=emojis[3],
relief=tkinter.FLAT, bd=0, bg=BgColor))
    selectButs.append(tkinter.Button(mainWin, command=lambda:sendEmoji(4), image=emojis[4],
relief=tkinter.FLAT, bd=0, bg=BgColor))
    selectButs.append(tkinter.Button(mainWin, command=lambda:sendEmoji(5), image=emojis[5],
relief=tkinter.FLAT, bd=0, bg=BgColor))
    selectButs.append(tkinter.Button(mainWin, command=lambda:sendEmoji(6), image=emojis[6],
relief=tkinter.FLAT, bd=0, bg=BgColor))
    selectButs.append(tkinter.Button(mainWin, command=lambda:sendEmoji(7), image=emojis[7],
relief=tkinter.FLAT, bd=0, bg=BgColor))
    selectButs.append(tkinter.Button(mainWin, command=lambda:sendEmoji(8), image=emojis[8],
relief=tkinter.FLAT, bd=0, bg=BgColor))
    selectButs.append(tkinter.Button(mainWin, command=lambda:sendEmoji(9), image=emojis[9],
relief=tkinter.FLAT, bd=0, bg=BgColor))

    for i in range(10):
        selectButs[i].place(x=30+85*i, y=590)
    status = 1


# 表情包按钮（默认 place）
textFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.BOLD, slant=ft.ROMAN)
emojiBut = tkinter.Button(mainWin, text='Emoji', command=selectEmoji, bg=textColor, fg='white',
font=textFont, activebackground='#191970', activeforeground='white')
emojiBut.place(x=130, y=500, width=80, height=40)


# 拖动文件函数
def DragFile():
    global filebox, status

    if status == 2:
        return
    elif status == 0:
        iptText.place_forget()
```

```
        sendBut.place_forget()
    elif status == 1:
        for i in range(10):
            selectButs.pop().destroy()

    filebox.place(x=20, y=560, width=500, height=140)
    dragLabel.place(x=520, y=580)
    deleteBut.place(x=560, y=650, width=80, height=40)
    sendFileBut.place(x=720, y=650, width=120, height=40)
    status = 2


# 文件按钮（默认 place）
textFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.BOLD, slant=ft.ROMAN)
fileBut = tkinter.Button(mainWin, text='File', command=DragFile, bg=textColor, fg='white',
font=textFont, activebackground='#191970', activeforeground='white')
fileBut.place(x=230, y=500, width=80, height=40)


# 拖动组件及绑定
dnd = TkDnD(mainWin)

def drop(files):
    global dnd

    if isinstance(files, str):
        files = re.sub(u"{.*?}", "", files).split() # 通过空格切分多文件，所以文件名中不能有空格

    for file in files:
        filebox.insert(tkinter.END, (file))

dnd.bindtarget(filebox, 'text/uri-list', '<Drop>', drop, ('%D',))


''' 在线用户列表 '''
# 创建多行文本框，显示在线用户
userbox = tkinter.Listbox(mainWin)

textFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.BOLD, slant=ft.ROMAN)
userbox.configure(font=textFont, bg=BgColor)
userbox.place(x=650, y=20, width=270, height=460)
isUserPanel = 0    # 判断在线用户列表面板开关的标志，0 为关

def UserBox():
    global userbox, isUserPanel
```

```
    if isUserPanel == 1:
        userbox.place(x=650, y=20, width=270, height=460)
        isUserPanel = 0
    else:
        userbox.place_forget()  # 隐藏控件
        isUserPanel = 1


# 查看在线用户按钮
textFont = ft.Font(family='Microsoft YaHei', size=12, weight=ft.BOLD, slant=ft.ROMAN)
userBut = tkinter.Button(mainWin, text='Users', command=UserBox, bg=textColor, fg='white',
font=textFont, activebackground='#191970', activeforeground='white')
userBut.place(x=795, y=500, width=80, height=40)


''' 私聊功能 '''
def private(*args):
    global chatWith

    # 获取点击的索引，得到用户名
    indexs = userbox.curselection()
    index = indexs[0]
    if index > 1:
        # 修改客户端名称
        if userbox.get(index) == '[ Group Chat ]':
            chatWith = '#@[GROUP]@#'
            mainWin.title('Mini Chatroom [ Group Chat ]')
        else:
            chatWith = userbox.get(index).rstrip(' [ user ]')
            mainWin.title('Mini Chatroom [ Private Chat with ' + chatWith + ']')


# 在显示用户列表框上设置绑定事件
userbox.bind('<ButtonRelease-1>', private)

# 文件数据（二进制），设为全局变量，用于数据拼接
fileData = b''

''' 接收信息并打印 '''
def recv():
    global users, fileData, fileMemory, ddl
    while True:
        data = sock.recv(1024)

        if data.startswith(b'#@[FILE]@#'):  # 文件
```

```
            fileData += data.split(b'$;$')[0].replace(b'#@[FILE]@#', b'')

            while True:
                data = sock.recv(1024)

                if b'<EOF>' in data:
                    fileData += data[:data.find(b'<EOF>')]
                    break
                else:
                    fileData += data.split(b'$;$')[0].replace(b'#@[FILE]@#', b'')
            continue


        try:    # 在线用户列表，若为消息则有异常捕获过程
            data = data.decode()
            users = json.loads(data)
            userbox.delete(0, tkinter.END)  # 清空列表框
            userbox.insert(tkinter.END, ('       Users online:  ' + str(len(users))))
            userbox.insert(tkinter.END, (user + ' [ myself ]'))
            userbox.insert(tkinter.END, '[ Group Chat ]')
            userbox.itemconfig(tkinter.END, fg='green')
            for i in range(len(users)):
                if users[i] != user:
                    userbox.insert(tkinter.END, (users[i] + ' [ user ]'))
                    userbox.itemconfig(tkinter.END, fg='green')

        except: # 消息
            try:
                data = data.split('$;$')
                msg = data[0].strip() # 消息
                sender = data[1]    # 发送者
                receiver = data[2]  # 接收者（群聊为'#@[GROUP]@#'，否则为user）
            except:
                tkinter.messagebox.showerror('ERROR', message='This program CANNOT receive message
like $;$!')
                return

            color = ''
            if receiver == '#@[GROUP]@#':   # 群聊
                if sender == user:  # 自己->所有人
                    color = 'mg'
                    chatbox.insert(tkinter.END, '[myself] @ [all] :\n', color)
                else:   # 别人->所有人
                    color = 'og'
                    chatbox.insert(tkinter.END, sender + ' @ [all] :\n', color)
```

```
            else:   # 私聊
                if sender == user:  # 自己->别人
                    color = 'mo'
                    chatbox.insert(tkinter.END, '[myself] @ ' + receiver + ' :\n', color)
                elif receiver == user:   # 别人->自己
                    color = 'om'
                    chatbox.insert(tkinter.END, sender + ' @ [myself] :\n', color)
                else:   # 别人->别人（不显示）
                    continue


            if msg.startswith('#@[EMOJI]@#'):   # 表情
                chatbox.image_create(tkinter.END, image=emojis[int(msg[-1])])
                chatbox.insert(tkinter.END, '\n', color)

            elif msg.startswith('#@[ENDFILE]@#'):   # 文件结束
                chatbox.insert(tkinter.END, '[File] ' + msg[13:] + '\n', color)
                if not os.path.isdir('./download'):
                    os.mkdir('./download')
                with open('./download/' + msg[13:], 'wb') as f:
                    f.write(fileData)
                fileData = b''

            else:   # 一般消息
                chatbox.insert(tkinter.END, msg + '\n', color)

            chatbox.see(tkinter.END)  # 显示在最后


# 创建线程用于实时接收信息
r = threading.Thread(target=recv)
r.start()

mainWin.mainloop()
sock.close()  # 关闭图形界面后断开 TCP 连接
```

③ TkDND 中间脚本 TkDnD.py

```
# TkDnD.py

class TkDnD:
    def __init__(self, tkroot):
        self._tkroot = tkroot
        tkroot.tk.eval('package require tkdnd')
        # make self an attribute of the parent window for easy access in child classes
        tkroot.dnd = self

    def bindsource(self, widget, type=None, command=None, arguments=None, priority=None):
        '''Register widget as drag source; for details on type, command and arguments, see bindtarget().
        priority can be a value between 1 and 100, where 100 is the highest available priority (default:
50).
        If command is omitted, return the current binding for type; if both type and command are omitted,
        return a list of registered types for widget.'''
        command = self._generate_callback(command, arguments)
        tkcmd = self._generate_tkcommand('bindsource', widget, type, command, priority)
        res = self._tkroot.tk.eval(tkcmd)
        if type == None:
            res = res.split()
        return res

    def bindtarget(self, widget, type=None, sequence=None, command=None, arguments=None,
priority=None):
        '''Register widget as drop target; type may be one of text/plain, text/uri-list,
text/plain;charset=UTF-8
        (see the man page tkDND for details on other (platform specific) types);
        sequence may be one of '<Drag>', '<DragEnter>', '<DragLeave>', '<Drop>' or '<Ask>' ;
        command is the callback associated with the specified event, argument is an optional tuple of
arguments
        that will be passed to the callback; possible arguments
include: %A %a %b %C %c %D %d %L %m %T %t %W %X %x %Y %y
        (see the tkDND man page for details); priority may be a value in the range 1 to 100 ; if there
are
        bindings for different types, the one with the priority value will be proceeded first (default:
50).
        If command is omitted, return the current binding for type, where sequence defaults to '<Drop>'.
        If both type and command are omitted, return a list of registered types for widget.'''
        command = self._generate_callback(command, arguments)
        tkcmd = self._generate_tkcommand('bindtarget', widget, type, sequence, command, priority)
        res = self._tkroot.tk.eval(tkcmd)
        if type == None:
```

```
            res = res.split()
        return res

    def clearsource(self, widget):
        '''Unregister widget as drag source.'''
        self._tkroot.tk.call('dnd', 'clearsource', widget)

    def cleartarget(self, widget):
        '''Unregister widget as drop target.'''
        self._tkroot.tk.call('dnd', 'cleartarget', widget)

    def drag(self, widget, actions=None, descriptions=None, cursorwindow=None, command=None,
arguments=None):
        '''Initiate a drag operation with source widget.'''
        command = self._generate_callback(command, arguments)
        if actions:
            if actions[1:]:
                actions = '-actions {%s}' % ' '.join(actions)
            else:
                actions = '-actions %s' % actions[0]
        if descriptions:
            descriptions = ['{%s}'%i for i in descriptions]
            descriptions = '{%s}' % ' '.join(descriptions)
        if cursorwindow:
            cursorwindow = '-cursorwindow %s' % cursorwindow
        tkcmd = self._generate_tkcommand('drag', widget, actions, descriptions, cursorwindow, command)
        self._tkroot.tk.eval(tkcmd)

    def _generate_callback(self, command, arguments):
        '''Register command as tk callback with an optional list of arguments.'''
        cmd = None
        if command:
            cmd = self._tkroot._register(command)
            if arguments:
                cmd = '{%s %s}' % (cmd, ' '.join(arguments))
        return cmd

    def _generate_tkcommand(self, base, widget, *opts):
        '''Create the command string that will be passed to tk.'''
        tkcmd = 'dnd %s %s' % (base, widget)
        for i in opts:
            if i is not None:
                tkcmd += ' %s' % i
        return tkcmd
```