南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

实验报告

简单聊天室程序报告

姓名：林雪

学号：2120220589

专业：计算机科学与技术

2022 年 12 月 19 日

目录

一、 简单聊天室程序报告

(一) 编程环境

Windows10 x64、Python 3.11.0、PyMySQL-1.0.2、mysql8.0.16

(二) 程序流程

服务器端的流程如如下所示：首先根据用户输入的进行 IP 地址和端口的绑定，在绑定之前需要进行有效性的检查，检查成功后则在端口处进行监听，监听成功则建立新的线程同客户端进行通信。

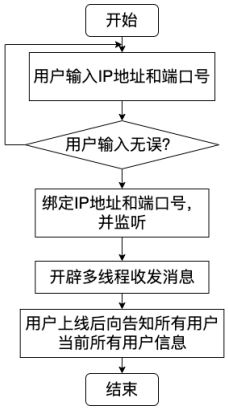

图 1: 服务器端流程图

客户端的流程图如下：用户首先需要注册，注册成功后利用注册好的用户名进行登陆，登陆成功后输入服务器的 IP 地址和端口和服务器建立连接，并输入自己的 IP 地址和端口号进行绑定，并将自己的 IP 地址和端口号发送至服务器端，服务器端收到后进行转发，然后根据其他用户的用户名获取 IP 地址和端口号，从而实现群聊和私聊。结束后可以点击下线，结束程序。

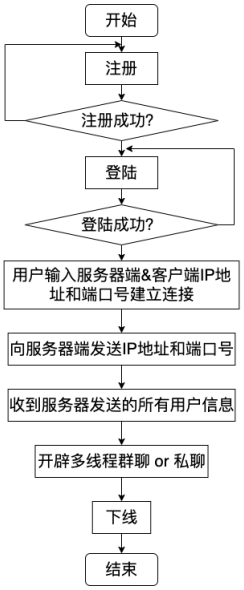

图 2: 客户端流程图

(三) 关键问题

1. 用户注册——连接 mysql

用户在进行聊天之前首先要通过用户名和密码进行登录，本程序选择将用户名和密码的对应关系存储在 mysql 数据库中，因此用户需要先进行注册，注册的时候用户名是唯一的，作为数据库中的主键，只有注册后才可以进行登录，同时要求登录的密码和注册时的密码保持一致，只有在登录后才可以和服务器进行连接从而和其他用户聊天。具体实现的代码如下所示：

```
def Register():
    # 获取用户注册时输入的用户名和密码
    username = textUsername0.get("0.0",END).rstrip("\n")
    password = textPassword.get("0.0",END).rstrip("\n")

    # 判断用户输入是否为空，为空则弹出新窗口，提醒用户
    if username == "":···

    # 和数据库表建立连接
    conn=pymysql.connect(host = '127.0.0.1',user = 'root',passwd='123456',port= 3306,db='user',charset='utf8')
    cur = conn.cursor()
    # 执行select语句，判断用户输入的用户名是否存在
    sql="select username from `user` "
    cur.execute(sql)
    data = cur.fetchall()
    for i in data:
        if username == i[0]:···

    # 如果用户名不存在，则进行注册，即将用户名和密码的对应关系存储在数据库表中
    sql = "insert into user values(%s,%s,%s)"
    param = (username, password, "")
    cur.execute(sql,param)
    cur.close()
    conn.close() # 关闭连接
```

图 3: 用户注册

用户注册时，首先判断用户是否输入内容，如果没有，则弹出新窗口提示用户。用户输入后，利用 pymysql 库的 connect 函数和创建好的数据库表进行连接，然后通过 select 语句查询数据库，判断用户名是否重复，如果重复则弹出新窗口提示用户。如果用户输入的用户名并不重复，则利用 insert 语句将其插入到数据库表中，最后利用 commit 语句将操作同步更新至数据库中。

2. 用户登录——连接 mysql

用户通过上面的步骤成功注册后便可以利用注册好的用户名和密码进行登录，登录的时候只需要查询数据库判断用户名是否存在并且用户名和密码是否对应即可。具体实现的代码如下图所示：

```
def Authentication():
    # 获取用户登录时输入的用户名和密码
    username = textUsername0.get("0.0",END).rstrip("\n")
    password = textPassword.get("0.0",END).rstrip("\n")

    # 判断用户输入是否为空
    if username == "":···

    conn=pymysql.connect(host = '127.0.0.1',user = 'root',passwd='123456',port= 3306,db='user',charset='utf8')
    cur = conn.cursor()
    sql="select * from `user` "
    cur.execute(sql)
    data = cur.fetchall() # 获取查询结果

    flag = 0
    for i in range(len(data)): # 查询对应的用户名是否存在
        if username == data[i][0] and password == data[i][1]:
            flag = 1
        elif username == data[i][0]:
            flag = 2

    # 用户名不存在
    if flag == 0:···
    elif flag == 2: # 用户名存在但是密码不正确···

    cur.close()
    conn.close()
    return username # 返回用户名，以在其他函数中利用
```

图 4: 用户登录

和注册过程类似，首先获取用户输入并判断用户输入是否为空，然后查询数据库，判断用户名是否存在并且用户名和密码是否对应，如果出现了上述问题，则会弹出新窗口提示用户，否则成功登录，返回用户名，为后续做准备。

3. 服务器端登录

客户端需要和服务器端建立连接才能获取其他用户的 IP 地址等信息，因此需要服务器首先绑定 IP 和端口，然后在相应端口处监听是否存在连接，客户端便可以通过指定的 IP 地址和端口同服务器端建立连接并通信。

```
def start():
    ipStr = textServer.get("0.0",END).split(":")[0]
    ipStr = ipStr.rstrip("\n")
    portStr = textServer.get("0.0", END).split(":")[1]
    portStr = portStr.rstrip("\n")

    # 初始化socket。并绑定IP和端口
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((ipStr, int(portStr)))
    server.listen(10) # 进行监听

    text.insert(tkinter.INSERT, "服务器启动成功！\n")

    while True:
        connect, addrss = server.accept() # 和客户端建立连接
        t = threading.Thread(target=run, args=(connect, addrss))
        t.start()
```

图 5: 服务器端登录

首先初始化一个 socket 套接字，然后和输入的 IP 地址 + 端口进行绑定，绑定成功后在相应端口处进行监听，并且为了实现实时通讯的目的，为每一个新的用户都开辟一个单独的线程完成用户通信。

4. 客户端和服务器建立连接

用户成功登录后便可以根据服务器的 IP 地址和端口建立连接，建立连接时需要将自己的 IP 地址和端口号发送至客户端，服务器端也会将此时在线的所有用户的信息转发至该客户端，便于后续客户端间直接进行通信。

```
client_name_str = textUsername.get('0.0', END)
client_name_str = client_name_str.rstrip("\n")
print(client_name_str)

global client # 为客户端绑定IP地址和端口，便于客户端之间直接进行通信
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)# 客户端之间通过UDP方式
client_ip_str = textClient.get('0.0', END).split(":")[0]
client_ip_str = client_ip_str.rstrip("\n")
client_port_str = textClient.get('0.0',END).split(":")[1]
client_port_str = client_port_str.rstrip("\n")
client.bind((client_ip_str,int(client_port_str)))
```

图 6: 客户端绑定 IP 地址

```
global ck # 获取服务器的IP地址和端口号
server_ip_str = textServer.get("0.0", END).split(":")[0]
server_ip_str = server_ip_str.rstrip("\n")
server_port_Str = textServer.get("0.0", END).split(":")[1]
server_port_Str = server_port_Str.rstrip("\n")

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)# 和服务器之间采用TCP方式
try:
    server.connect((server_ip_str, int(server_port_Str)))#连接服务器
except:
    text.insert(tkinter.INSERT,"登陆失败，请检查服务器ip是否正确，以及客户端的ip是否重复\n")

#将自己的登录名发送给服务器，函数会附带自己的IP信息
server.send((client_name_str+" "+client_ip_str+" "+client_port_str).encode("utf-8"))
```

图 7: 客户端和服务器建立连接

建立连接后客户端需要将自己的 IP 地址和端口号告知与服务器端，服务器收到消息后会将客户端 IP 地址和时钟等信息同步至所有用户。

5. 同步信息

服务器在用户建立连接后首先记录用户的上线消息，并且将用户上线和下线信息作为日志记录在文件中。然后将建立连接后收到的用户的 IP 地址和端口以及用户名存储起来，然后转发至

所有用户，并且将时间也进行转发，达到消息的同步。

```
dir = os.getcwd()
file = dir + "/log.txt"
# 将用户上线信息记录到日志中
with open(file, mode='a', encoding='utf-8') as file_obj:
    file_obj.write(userName + "连接\n")

# 告知用户登录成功，并告知其余的用户的用户名、ip地址以及端口号
printStr ="登录成功!\n"+"当前在线的好友有："+str(list(users.keys()))+"\n"
for key in users:
    printStr += key + ":ip: " + users[key][1] + " port: " + users[key][2] + "\n"
connect.send(printStr.encode())

# 告知其余用户某个用户已上线，并发送所有用户的用户名、ip地址以及端口号
printStr = time.strftime('%Y-%m-%d %H:%M:%S ', time.localtime())
printStr += userName + "已上线\n"+"当前在线的好友有：\n"
for key in users:
    printStr += key + ":ip: " + users[key][1] + " port: " + users[key][2] + "\n"
print(printStr)
for key in users:
    if key != userName:
        users[key][0].send(printStr.encode())
```

图 8: 同步信息

同时客户端在收到消息后则需要解析消息，将所有用户的用户名、IP 地址以及端口号存储下来，并且由于用户名是唯一的，因此可以将用户名作为字典的 key 存储。

6. 群聊 & 私聊

当用户指定了发送给谁后，会进行私聊，并且会将发送消息的时候一起发送到指定用户，当用户并未指定发送给谁时，则默认是群发，则发送至所有用户。发送消息时，通过用户名查找存储的所有用户的信息，即用户名对应的 IP 地址和端口。然后将用户输入的内容加上时间戳发送至指定用户/群发.

```
def sendMail():
    global friend
    friend_name = textFriend.get("0.0", END).rstrip("\n") # 获取用户指定的接收方
    try:
        friend_ip_str = users[friend_name][0]
        friend_port_str = users[friend_name][1]
    except:
        if friend_name != "":
            text.insert(tkinter.INSERT,"输入昵称不存在，请重新输入\n")

    sendStr = time.strftime('%Y-%m-%d %H:%M:%S ', time.localtime()) # 获取发送消息的时间
    username  = textUsername.get("0.0", END).rstrip("\n")
    sendStr += username # 加上发送方的名字
    sendStr += ":"
    sendStr += textSend.get("0.0", END)

    if len(friend_name) != 0: # 私聊
        text.insert(tkinter.INSERT, time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())
        + '\n'+'我对'+friend_ip_str+":"+friend_port_str+'说: ' + sendStr+'\n')
        friend = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # 发送给指定的人
        client.sendto(sendStr.encode(), (friend_ip_str,int(friend_port_str)))
    else:# 群聊
        for key in users: # 发给除了自己以外的人
            if key != username:
                client.sendto(sendStr.encode(), (users[key][0], int(users[key][1])))
```

图 9: 同步信息

首先检测用户输入的用户名是否存在，然后利用用户是否输入内容判断是私聊还是群聊，私聊则利用用户输入的接收方的用户名查找存储的用户信息，找到对应的 IP 地址和端口号，利用 UDP 协议的 sendto 函数发送消息，如果是群聊，则向除了自己之外的所有用户发送消息。并且需要注意的是在发送消息之前要加上时间。

7. 图形界面

本程序采用 tkinter 库设计图形界面，主要分为客户端和服务器端两个主要的图形界面，其中客户端在登录之前和登录之后采取两个界面，并且在一些输入有问题的时候会弹出新窗口进行提示。

```
login_window.destroy() # 登陆后删除原有窗口，建立新窗口
client_window.focus_force()
client_window.mainloop()
```

图 10: 图形界面

```
labelServer = tkinter.Label(server_window, text="ip:Port").grid(row=1, column=0)
eserver = tkinter.Variable()
textServer = tkinter.Text(server_window, height=1, width=35)
textServer.grid(row=1, column=1)


button = tkinter.Button(server_window, text="启动", command=startSever).grid(row=1, column=2,padx=5)
text = tkinter.Text(server_window, height=15, width=35)
labeltext = tkinter.Label(server_window, text='连接消息').grid(row=3, column=0)
text.grid(row=3, column=1)

server_window .mainloop()
```

图 11: 图形界面

（四） 测试截图

测试结果如下，首先是客户端可服务器端初始界面，对于客户端，首先需要输入用户名和密码进行登录（需要先注册），对于服务器端，可以直接进行 IP 地址和端口号的绑定。

图 12: 服务端初始界面

图 13: 客户端初始界面

对于服务器端，用户需要输入 IP 地址和端口号，其中输入放在同一行，以: 冒号作为分隔符，地址绑定成功后，会提示用户成功。

对于客户端，当成功登录后，会看到用户名已经写在了界面上，减少用户输入的内容，然后用户需要输入自己的 IP 地址和端口号以及服务器的 IP 地址和端口号，同样是以: 冒号作为分隔符。

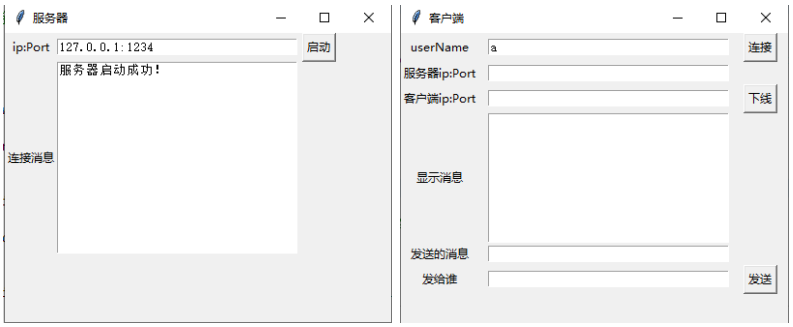

图 14: 服务端绑定地址成功

图 15: 客户端登陆后界面

对于客户端，如果注册成功，则数据库表会相应改变，下图是通过 mysql bench 观察到的数据表的变化。

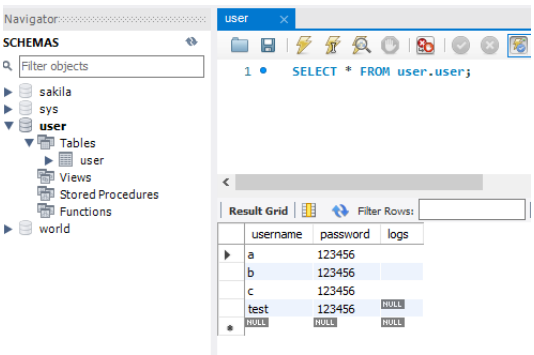

图 16: mysql

当用户和服务器成功建立连接后，服务器会告知所有用户此时在线的所有用户以及对应的

IP 地址和端口号。

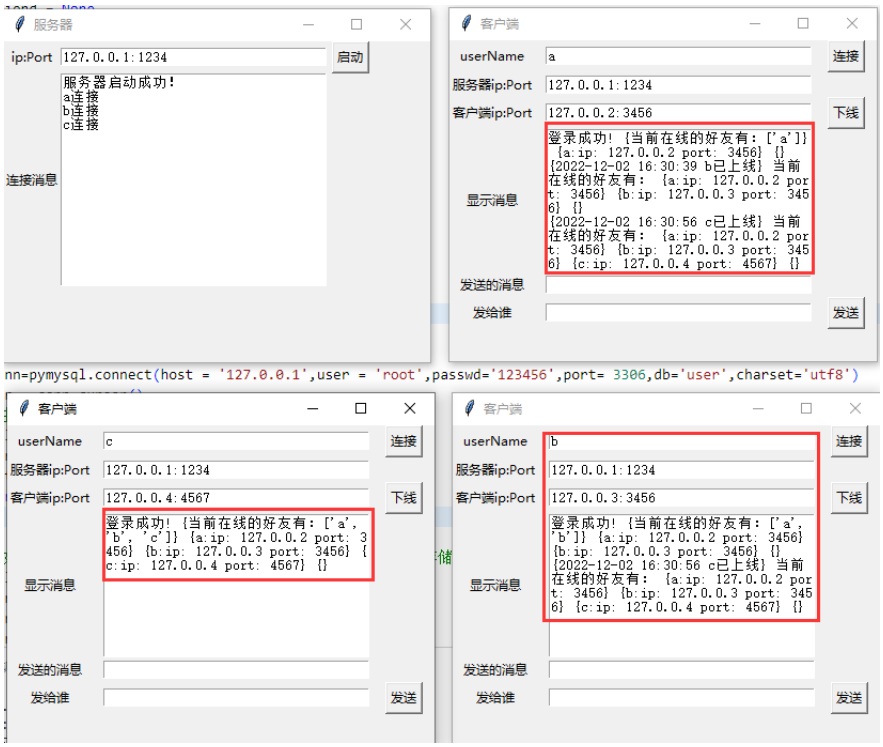

图 17: 建立连接

如果在用户在登录和注册时发生了错误，会产生新窗口提示用户。

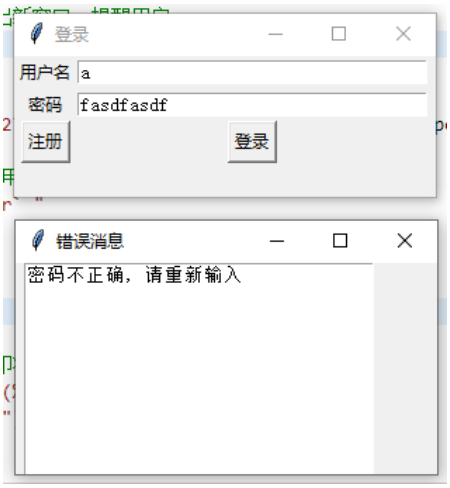

图 18: 登陆失败

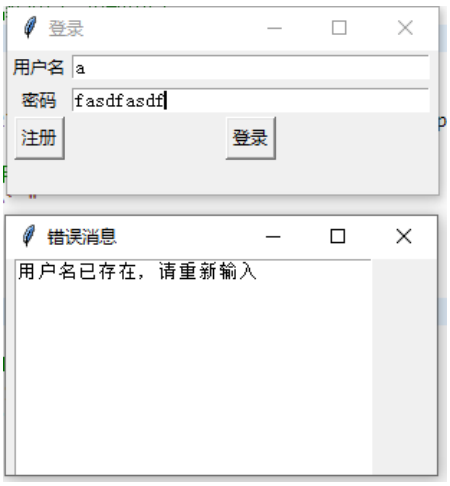

图 19: 注册失败

当用户根据从服务器端获取到的信息进行聊天时，直接输入接收方的用户名即可，如果不输入则默认是群发

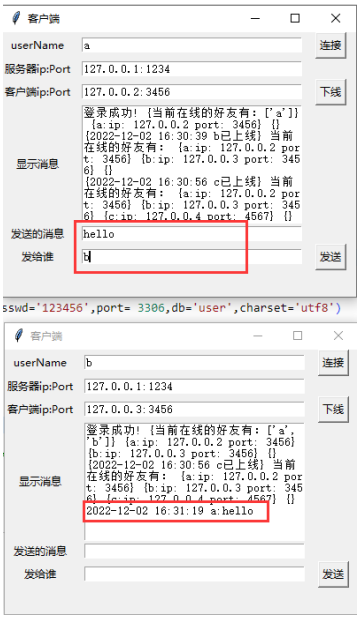

图 20: 私聊

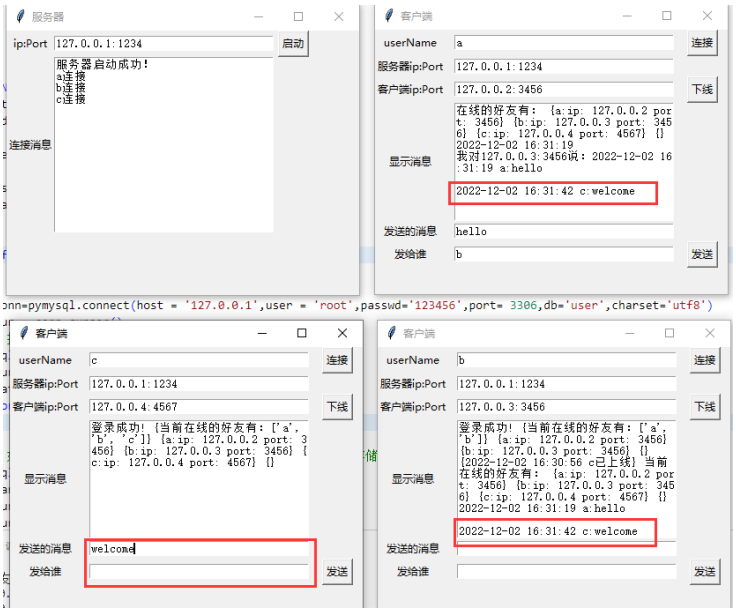

图 21: 群聊

用户可以点击下线按钮，和服务器端断开连接，这个信息也会同步的告知所有用户，其余用户将不能与该用户再次联系

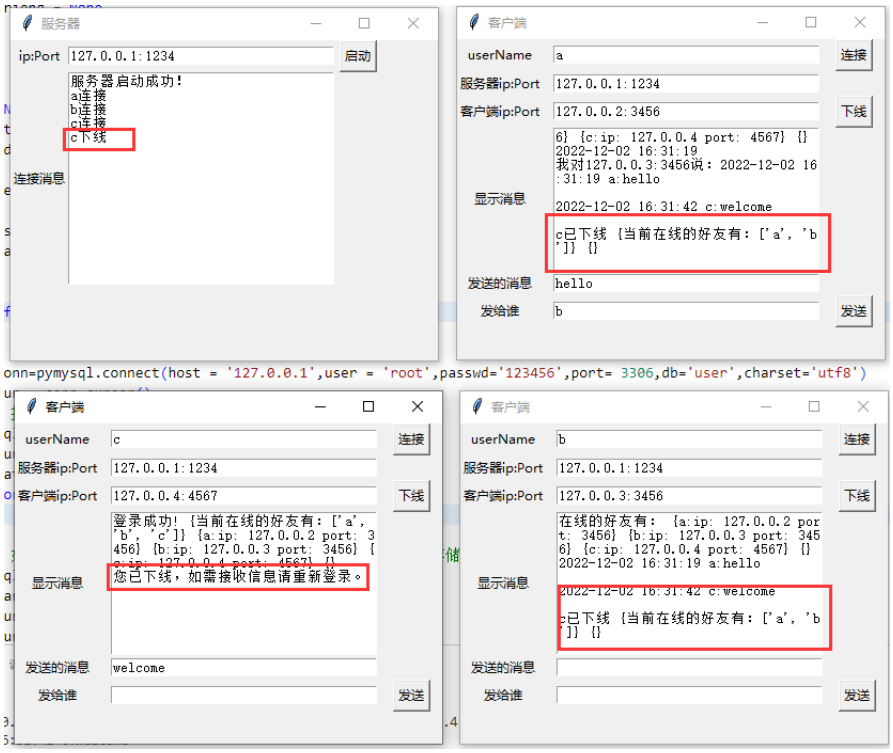

图 22: 用户下线

其中服务器会记录用户上线和用户下线的日志存储在 log.txt 文件中。

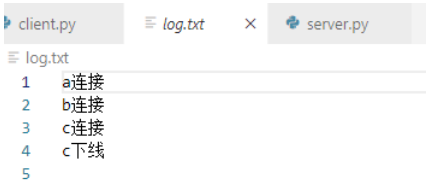

图 23: 日志