计算机网络课程设计说明

题目： 即时通讯系统

作者：张鑫 学号：2019141470210

一 采用协议

TCP/IP

二 通信模块的简介

程序中服务器和客户端交流的基础方式是`socket.send()`函数，因此需要自己定义消息的格式。我定义的消息格式是`header+msg`，其中`header`是头部信息，固定格式为5个字符。当服务器或客户端使用`socket.recv()`函数接收到了信息后，会直接进行拆分，消息的`header`就是前5个字符，然后会根据`header`进入不同的操作，可以参见下面列出来的。 regis: register 注册

login: login 登录

signo: sign out 退出登录

addfr: add friend 添加好友

delfr: delete friend 删除好友

sendt: send to 发送消息给...特定用户

chehs: check history 查看聊天记录

delhs: delete history 删除选中记录

chana: change name 更改名字

chapa: change password 更改密码

broad: broadcast 广播给所有用户

具体实现函数

server 与 client 的连接建立以及监听请求

这个函数是server的主要函数，从配置文件中读入host 和 port 绑定套接字，并持续监听客户端的连接请求，当监听到连接请求后会为客户端创建开启一个新的进程名 build_connect，该进程持续监听客户端发来的消息，并对其做出处理。

```
    host = conf.get('address','host')
    port = int(conf.get('address','port'))
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((host, port))
```

```
    # address : id
    client_dict = {}
    # connection : address
    address_dict = {}
    # the max listen number
    maxListen = 10
    s.listen(maxListen)

    print('the server is on!')
    init_database()
    while True:
        # continue to accepting the connection from the clients
        conn, address = s.accept()
        address_dict[conn] = address
        print('~',address, 'connected!')
        threading.Thread(target=build_connect, args=(conn, address)).start()
```

当server和client建立连接后，server会在字典中添加client的conn->address,和address->id的映射行为，这样做的好处是能更快的进行后续的发送行为。

```
# Build the connection between the client and the server
def build_connect(connection, address_port):
    try:
        while True:
            msg = connection.recv(1024).decode()
            if len(msg)<= 0:
                raise Exception
            if msg != '':
                get_request(address_port, msg)
                print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    except:
        print(connection,address_port,client_dict[address_port],'has quit!')
        print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
        # 从字典中移除
        address_dict.pop(connection)
        client_dict.pop(address_port)
```

服务器监听客户端的请求

如前面所描述的，消息的前五个字符被用于作为header，当接收到来自client的消息后，server的会调用get_request（）函数对请求进行划分，以便于下一步操作。

```
# distinguish the kind of  request by the first five character
def get_request(addr, msg:str):
    header = msg[0:5]
    print('header',header)
    message = msg[5:]
    print('message:',message)
```

```
        # login
        if header == 'login':
            print('~request:login')
            login(addr,message)
        # sign out
        if header == 'signo':
            print('~request:signo')
            sign_out(addr)
        # register
        if header == 'regis':
            print('~request:register')
            register(addr,message)
        # delete friend
        if header == 'delfr':
            print('~request:delete friend')
            delete_friend(addr, message)
        # add friend
        if header == 'addfr':
            print('~request:add friend')
            add_friend(addr, message)
        # send to
        if header == 'sendt':
            print('~request:send message')
            send_to(addr, message)
        # check history
        if header == 'chehs':
            print('~request:check history')
            check_history(addr,message)
        # delete history
        if header == 'delhs':
            print('~request:delete history')
            delete_history(addr, message)
        # change name
        if header == 'chana':
            print('~change name')
            change_name(addr, message)
        # broadcast to everyone
        if header == 'broad':
            print('~broadcast to everyone')
            broadcast(addr, message)
        # change password
        if header == 'chapa':
            print('~change password')
            change_password(addr,message)
        # show friends
        if header == 'shofr':
            print('~show friends')
            show_friends(addr,message)
        # show online users
        if header == 'shonl':
            print('~show online users')
            show_online(addr)
```

服务器端进行的详细操作

因为篇幅原因这里不过多赘述，详细内容可以参见源代码，这里以注册函数为例子。 这个注册函数接受的参数client的地址addr和消息msg。此处的msg是get_request()中接受的msg[5:]的切片。 注册函数首先会打开数据库，并且从收到的msg中切分出用户名字和密码，然后会把所有的已经存在的id取出来存入一个元组id_exist，然后系统会生成一个随机的不存在与id_exist的5位id，并用户信息存入数据库，向client发送成功的信息。 异常时向用户发送警告信息。

```
def register(addr,msg:str):
    try:
        d = sqlite3.connect('data.db')
        u_name = msg[0:5]
        u_password = msg[5:]
        sql = 'select user_id from user_info'
        id_exist = d.execute(sql).fetchall()

        u_id = random.randint(10000,99999)
        # find a not distinct id
        while u_id in id_exist:
            u_id = random.randint(10000, 99999)

        sql = "insert into user_info(user_name,user_id,password) values (?,?,?)"
        v = (u_name,u_id,u_password)
        d.execute(sql,v)
        d.commit()
        print('register successfully!',
              'id =',u_id,
              'name = ',u_name,
              'password = ',u_password)
        # conn.send

        send_msg(addr,'register successfully,id ='+str(u_id))

    except sqlite3.DatabaseError:
        print('register fail!')
        # conn.send
        send_msg(addr,'register fail,database error')
    except:
        send_msg(addr,'register fail,unknown error')
    finally:
        d.close()
```

用户端的图形界面简单介绍

client端的图形界面是写在两个文件中的，因为只有两个界面，一个是登陆界面，一个是主界面。 登陆界面通过下面这个函数来跳转入主界面，先销毁当前的界面，然后创建一个主界面的对象

```
    def login(self):
```

```
        self.build()

        if self.check(self.account_get.get(),self.password_get.get()):
            # 销毁当前页面，并创建一个新的页面
            self.root.destroy()
            # 存入账户方便下次登录
            if  not self.account_get.get() in self.account_values:
                self.write_to('./config/account.txt',self.account_get.get())
            mainWindow.Main(self.s, self.account_get.get(), 0)
```

主界面的创建函数简略如下

```
class Main:

    def __init__(self,s,user_id,recv_id):
    ...
```

这个函数中的s 代表着从login.py中传入的套接字，user_id 是用户帐号，recv_id是接收者的id，默认为 0 表示群聊，所以进入后不进行其他操作是群聊模式。

测试

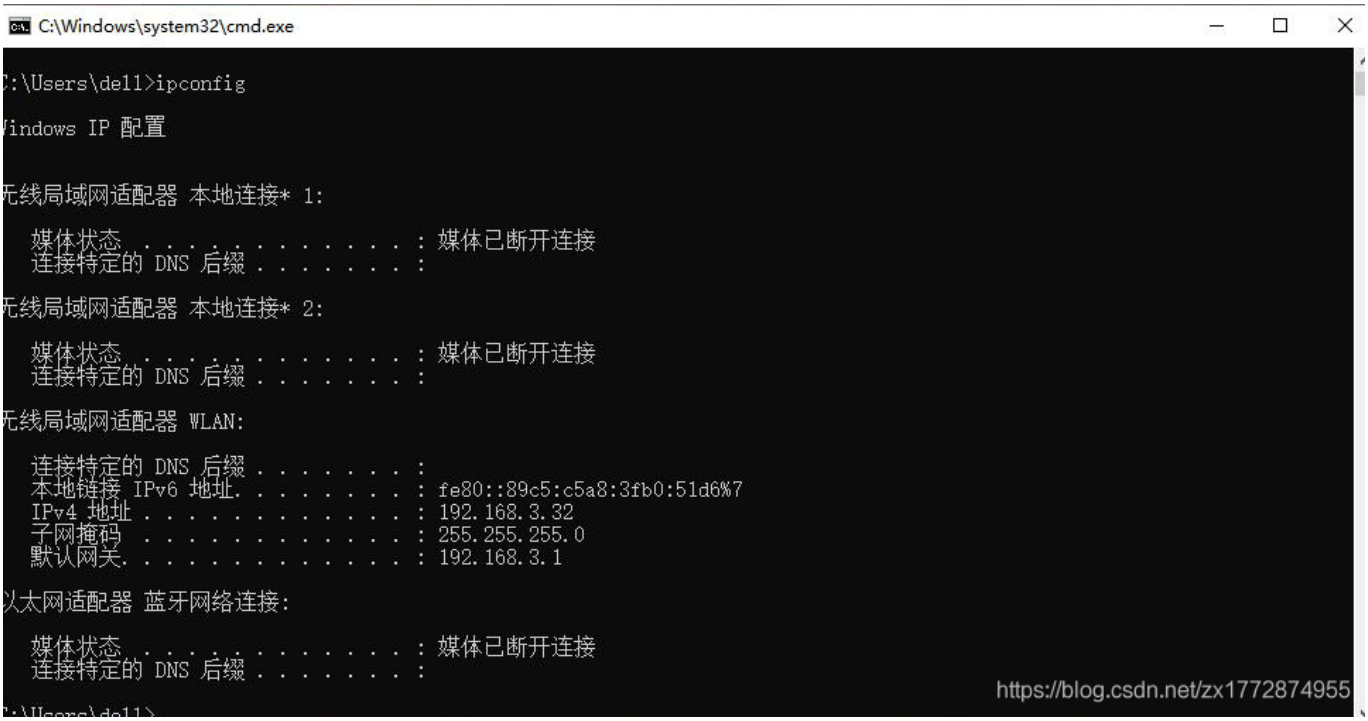
主要功能测试

主要功能的测试可残见软件使用文档中如何使用，这里进行的是在三台计算机上的测试。

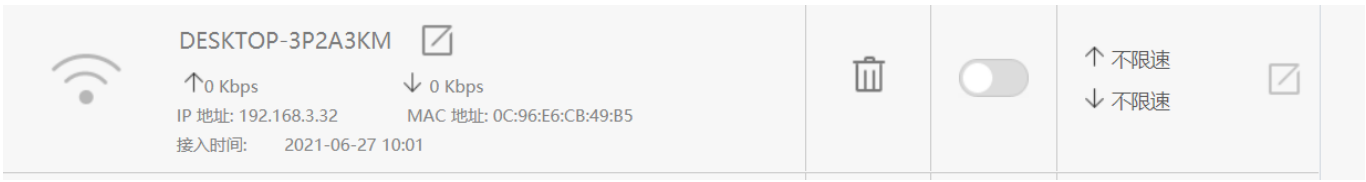
在局域网环境下三台电脑上测试

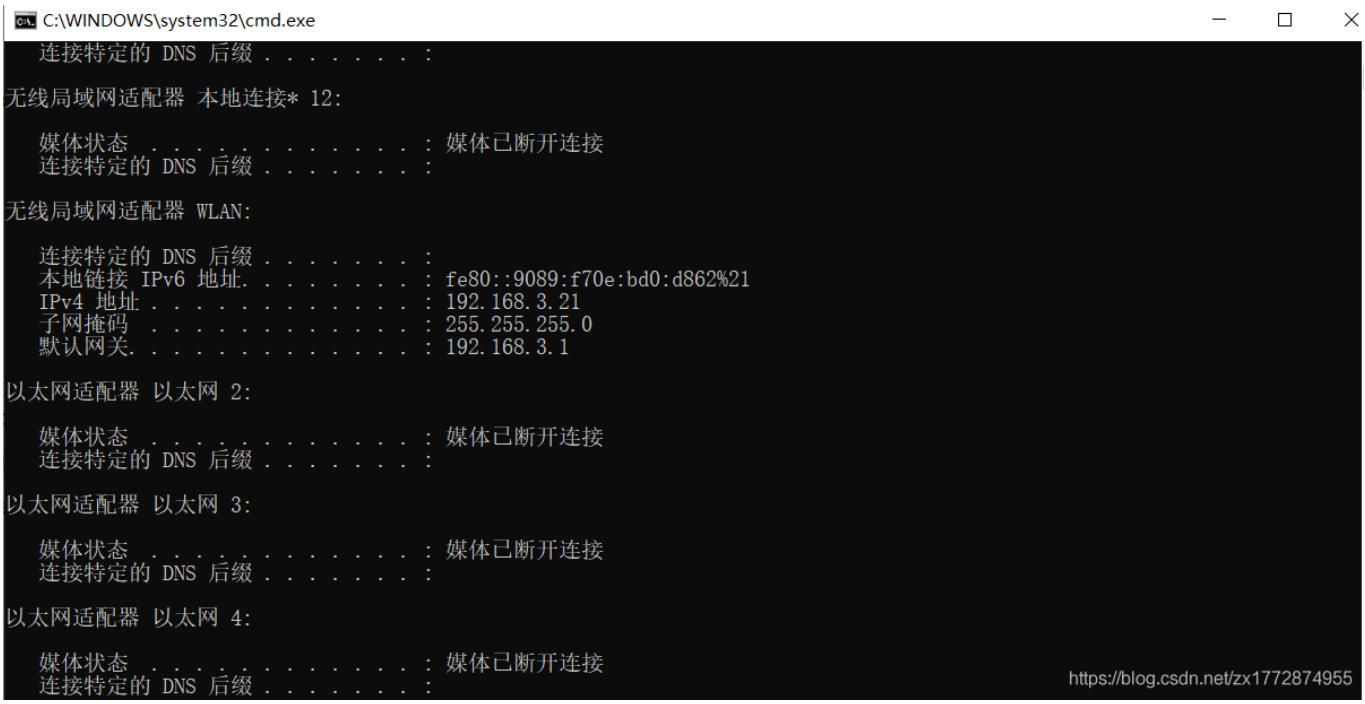
测试的方法为我自己的电脑开启server，与其他两台电脑聊天。 我的电脑信息如下：

以下为其他两台测试电脑的信息： 我们应该关心他们的ip地址，这是在一个局域网中进行测试的，该局域网中连接的设备大致如下，这表明此程序是可以在局域网中正常运行的。



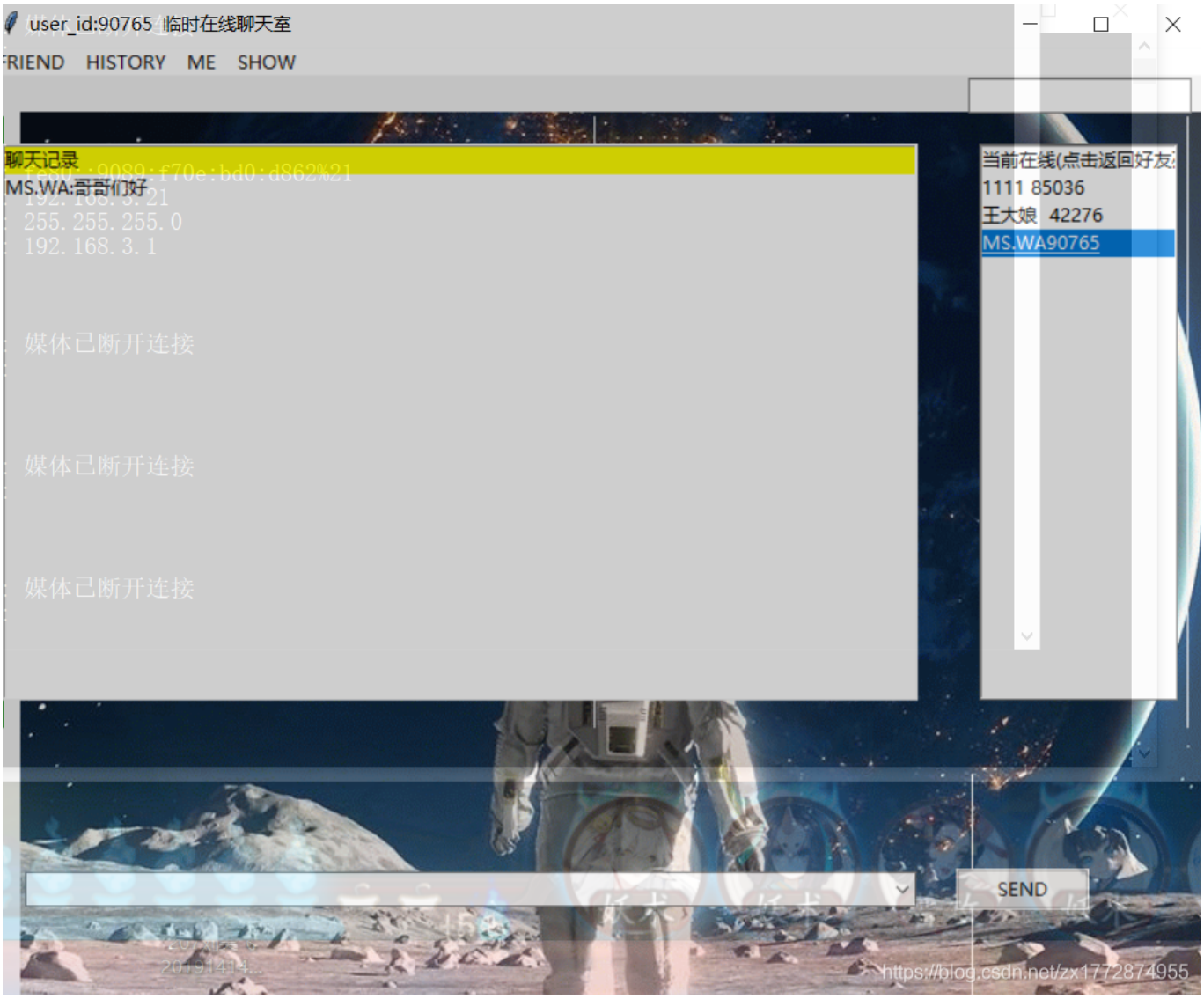
ip: 192.168.3.32（截图时未连接WiFi）

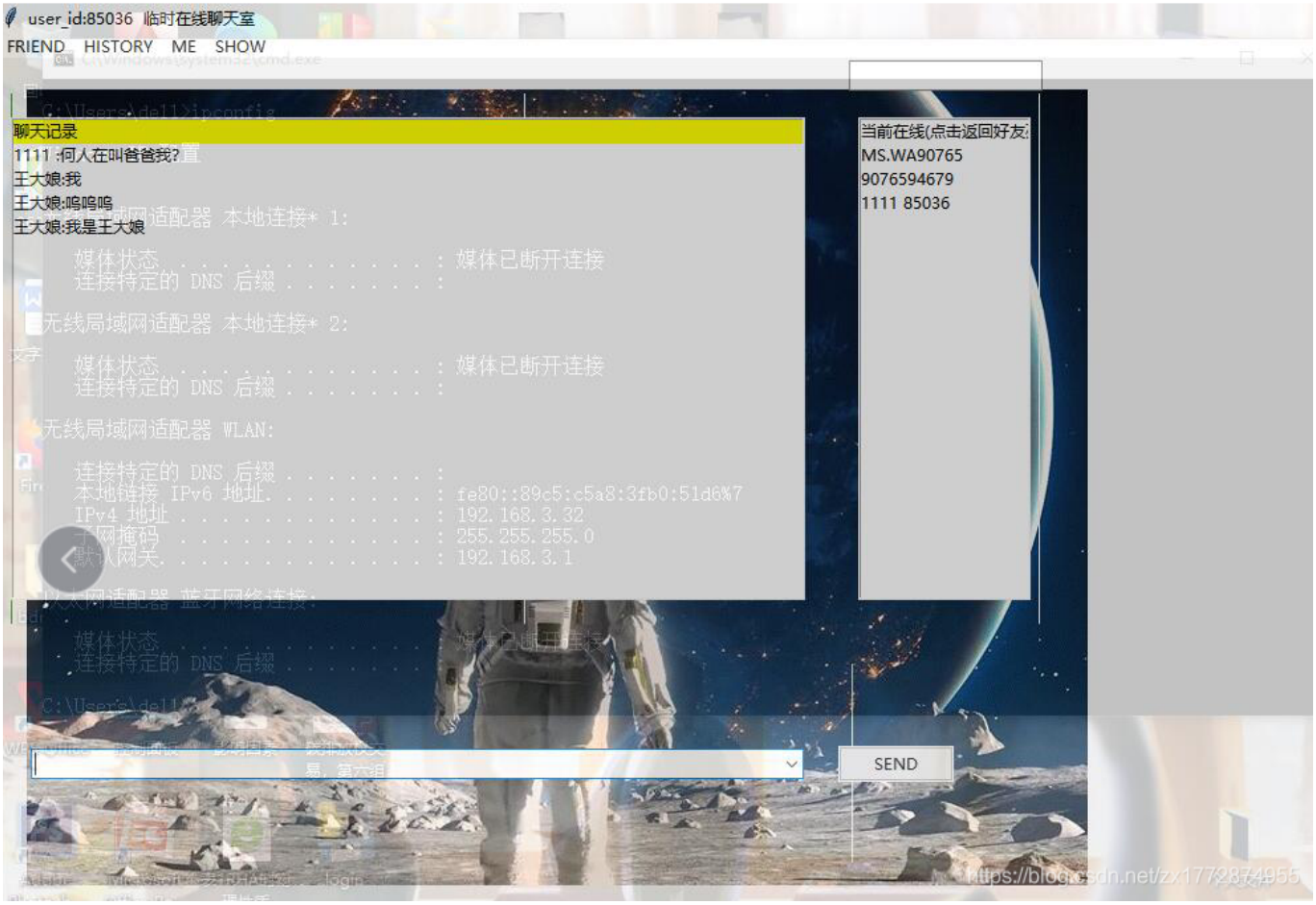




ip: 192.168.3.21

两台电脑上发送消息：


user_id:90765 临时在线聊天室
FRIEND HISTORY ME SHOW
聊天记录
MS.WA:哥哥们好
媒体已断开连接
媒体已断开连接
媒体已断开连接
当前在线(点击返回好友
1111 85036
王大娘 42276
MS.WA90765
SEND
https://blog.csdn.net/zx1772874955



在他们发送信息后我可以在server端看到他们发送消息的记录：

缺陷

文档中显示的测试只是一部分测试截图，有些测试时发生bug并未及时截图，所以未在此文档的测试中标注出来。但是此程序的缺陷仍有不少，不仅限于以下列出来的。

设计缺陷：

1 未在server或者client端设计一个timeout，即当客户端或者服务端因为某种异常原因死掉后，另一方可以在有限次数内尝试与之发送探测报文，如果未能收到有效回复则关闭套接字。这样做可以减少资源的浪费。 2 在编码初期没有认真的构建设计模式。对于在server中的消息处理完全可以用一个工厂模式来设计，server只负责调用抽象，抽象调用具体，这样才符合OCP开闭原则。

编码缺陷：

1 命名不规范，在早期写server时变量函数都用的是小驼峰命名法，然而pycharm编译器检验的合法命名是下划线连接的命名法，因此有许多函数和变量名的命名是不规范的，也没有修改过来，虽然不影响程序正常运行，但是对于维护和阅读造成了一定的困扰。 2 程序中的类过于臃肿，这也是设计的不合理之处，应当尽量保持单一功能原则与接口分离原则，让每个类中每个方法所完成的功能尽量的单一。

功能缺陷：

1 client端过快的点击一个按钮，会造成服务器端来不及处理消息而造成的线程崩溃，在线程崩溃后服务器端仍可以继续为其他client服务，但是该client则无法再请求服务，必须要重启该client。

解决方法：可以在client和server中添加一个线程用来确定两者的连接状态，如果连接发生异常，那么在有限次数的timeout内请求重连，如果连接失败则告知用户，并关闭程序。

2 client端的消息显示不合理，在使用文档的测试中可以发现，当一个用户接收到另一个用户的消息后，消息会直接出现在消息记录的面板上，虽然在好友栏有红色标识表明有私聊的信息，但是这样的显示方式不是太过于稳妥。

解决方法：应当在接收到消息后将其存入临时的一个对象或者文件中，而不是直接显示在消息记录中，当用户点击了对应发送者的聊天框后再显示到聊天记录中。

反思与收获

通过这次的课程设计，我对于python的套接字编程和用户界面设计更加熟悉了。在这次的课程设计中，我将软件工程导论，软件开发环境与工具，数据库与信息系统，计算机网络三门课所学内容结合起来，利用软件工程导论所学的内容对程序进行建模、设计、编码、测试以及项目管理，用软件开发环境与工具中所学的工具辅助完成工作，利用数据库与信息系统中所学的数据库知识构建含有数据库的小程序，利用计算机网络中所学的套接字编程进行程序之间的通信。虽然完成的和预期不是很符合，很多开始设想的功能如发送图片，发送音频等因为技术原因被阉割了，同时因为未能进行严密的建模就开编码，导致了程序出现了很多BUG，但是最终也算是完全独立的完成这个小程序的开发，看到它能够运行起来，感觉付出的一切都是蛮值得的。在这次课程设计的过程中，不仅巩固了许多学过的知识，同时也因为需要实现某些功能而学会了一些新的知识，如`.ini`文件的使用。