

1. 开发过程

(1) 对 client 对象进行编辑

对页面和参数的编写，用 python 指令，编写 client 的 gui 界面，连接发送，标签，以及文本框，还有按钮，编写完成后，开始编写功能

```
#定义一个client类
class client():
    def __init__(self):
        self.root=Tk()
        self.root.title('我是client')
        self.root.geometry('600x400')
        self.sk=socket.socket()
        self.server_ip='127.0.0.1'
        self.server_port=int(8888)
        self.recvbuf=chr(0)*1024 #接收区缓存
        self.sendbuf=chr(0)*1024 #发送区缓存
        self.recvstr=StringVar(value=self.recvbuf)
        self.sendstr=StringVar(value=self.sendbuf)
        self.ip=StringVar(value=self.server_ip)
        self.port=IntVar(value=self.server_port)
        #标签
        self.ip_label=Label(self.root,text='输入服务器IP地址')
        self.port_label=Label(self.root,text='输入服务器端口号(大于1024即可)')
        self.c_label=Label(self.root,text='输入框')
        self.s_label=Label(self.root,text='当前收到')
        self.recorde_label=Label(self.root,text='聊天记录(请在建立连接后通信)')
        #文本框
        self.ip_entry=Entry(self.root,textvariable=self.ip)
        self.port_entry=Entry(self.root,textvariable=self.port)
        self.c_entry=Entry(self.root,textvariable=self.sendstr)
        self.s_entry=Entry(self.root,textvariable=self.recvstr,state='disabled')
        self.recorde=scrolledtext.ScrolledText(self.root,width=50,height=10)
        #按钮
        self.btn0=Button(self.root,text='建立连接',command=lambda:self.started(self.ip,self.port))
        self.btn1=Button(self.root,text='发送',command=lambda:self.sending(self.sk))

        #排列
        self.c_label.grid(row=0,column=0)
        self.c_entry.grid(row=0,column=1)
        self.s_label.grid(row=1,column=0)
        self.s_entry.grid(row=1,column=1)
        self.btn0.grid(row=2,column=0)
        self.btn1.grid(row=2,column=1)
        self.ip_entry.grid(row=3,column=1)
        self.ip_label.grid(row=3,column=0)
        self.port_entry.grid(row=4,column=1)
        self.port_label.grid(row=4,column=0)
        self.recorde_label.grid(row=5,column=1)
        self.recorde.grid(row=6,column=1)
        self.root.after(500,self.update)
        self.root.mainloop()
```

Client 开始时进行连接请求代码编写

```

#开始连接
def started(self,ip,port):
    self.recorde.insert(INSERT,'waiting...\n')
    self.sk.connect((self.server_ip,self.server_port))
    print('连接成功')
    self.recorde.insert(INSERT,time.strftime('%Y-%m-%d %H:%M:%S')+ ' connected\n')
    return

```

然后 client 要进行数据的发送，所以进行发送数据过程的代码编写。

```

#发送数据
def senddata(self,s):
    self.sendbuf=self.sendstr.get()
    s.send(bytes(self.sendbuf,'utf8'))
    self.recorde.insert(INSERT,time.strftime('%Y-%m-%d %H:%M:%S')+ ' 客户端:' +self.sendbuf+'\n')
    return

```

client 数据发送后，要与 sever 进行交互，所以进行 client 的接收数据编写

```

#接收数据
def recvddata(self,s):
    try:
        self.recvbuf=str(s.recv(1024),'utf8')
        if not self.recvbuf:
            return
        self.recvstr.set(self.recvbuf)
        self.recorde.insert(INSERT,time.strftime('%Y-%m-%d %H:%M:%S')+ ' 服务器:' +self.recvbuf+'\n')
    except Exception as e:
        print(e)
    return

```

对 senddata()和 recvddata()方法采用多线程，并且更新缓存区

```

#对senddata()和recvdata()方法采用多线程
def sending(self,s):
    threading.Thread(target=self.senddata,args=(s,)).start()
    return
def recving(self,s):
    threading.Thread(target=self.recvdata,args=(s,)).start()
    return
#更新缓存区
def update(self):
    try:
        self.recving(self.sk)
    except Exception as e:
        print(e)
    self.root.after(500,self.update)
    return

```

(2) 对 sever 对象进行编写

代码大致框架

```

from tkinter import *
import socket
import time
import threading
from tkinter import scrolledtext
#定义一个server类
class server():
    def __init__(self):
        self.root=Tk()
        self.root.title('我是server')
        self.root.geometry('600x400')
        self.recvbuf=chr(0) #接收区缓存
        self.sendbuf=chr(0) #发送区缓存
        self.recvstr=StringVar(value=self.recvbuf)
        self.sendstr=StringVar(value=self.sendbuf)
        #标签
        self.ip_label=Label(self.root,text='输入服务器IP地址')
        self.port_label=Label(self.root,text='输入服务器端口号')
        self.s_label=Label(self.root,text='输入框')
        self.c_label=Label(self.root,text='当前收到')
        self.record_label=Label(self.root,text='聊天记录（请在建立连接后通信）')
        #文本框
        self.c_entry=Entry(self.root,textvariable=self.sendstr)
        self.s_entry=Entry(self.root,textvariable=self.recvstr,state='disabled')
        self.record=scrolledtext.ScrolledText(self.root,width=50,height=10)
        #按钮
        self.btn0=Button(self.root,text='建立连接',command=lambda:self.starting())
        self.btn1=Button(self.root,text='发送',command=lambda:self.sending())

```

#排列

```

self.s_label.grid(row=0,column=0)
self.c_entry.grid(row=0,column=1)
self.c_label.grid(row=1,column=0)
self.s_entry.grid(row=1,column=1)
self.btn0.grid(row=2,column=0)
self.btn1.grid(row=2,column=1)
self.record_label.grid(row=5,column=1)
self.record.grid(row=6,column=1)
self.root.after(500,self.update)
self.root.mainloop()

```

服务器端连接代码

```

#开始连接
def started(self):
    self.s=socket.socket()
    self.s.bind(('127.0.0.1',8888))
    self.s.listen(1)
    self.record.insert(INSERT,'waiting...')
    self.client,self.addr=self.s.accept()
    self.record.insert(INSERT,time.strftime('%Y-%m-%d %H:%M:%S')+ '当前连接到IP为'+str(self.addr[0])+'端口号为'+str(self.addr[1])+'\n')
    return

```

服务器端发送数据代码

```

#发送数据
def senddata(self):
    self.sendbuf=self.sendstr.get()
    self.client.send(bytes(self.sendbuf,'utf8'))
    self.recorde.insert(INSERT,time.strftime('%Y-%m-%d %H:%M:%S')+ ' 服务器:' +self.sendbuf+'\n')
    return

```

与客户端相同编写接受数据代码

```

#接收数据
def recvdata(self):
    try_:
        self.recvbuf=chr(self.client.recv(1024),'utf8')
        if not self.recvbuf:
            return
        self.recvstr.set(self.recvbuf)
        self.recorde.insert(INSERT,time.strftime('%Y-%m-%d %H:%M:%S')+ ' 客户端:' +self.recvbuf+'\n')
    except Exception as e:
        print(e)
    return

```

采用多线程，进行代码编写，并且编写更新缓存区代码

```

#对start(),senddata()和recvdata()方法采用多线程
def starting(self):
    threading.Thread(target=self.started).start()
    return
def sending(self):
    threading.Thread(target=self.senddata).start()
    return
def recving(self):
    threading.Thread(target=self.recvdata).start()
    return
#更新缓存区
def update(self):
    try:
        self.recving()
    except Exception as e:
        print(e)
    self.root.after(500,self.update)
    return

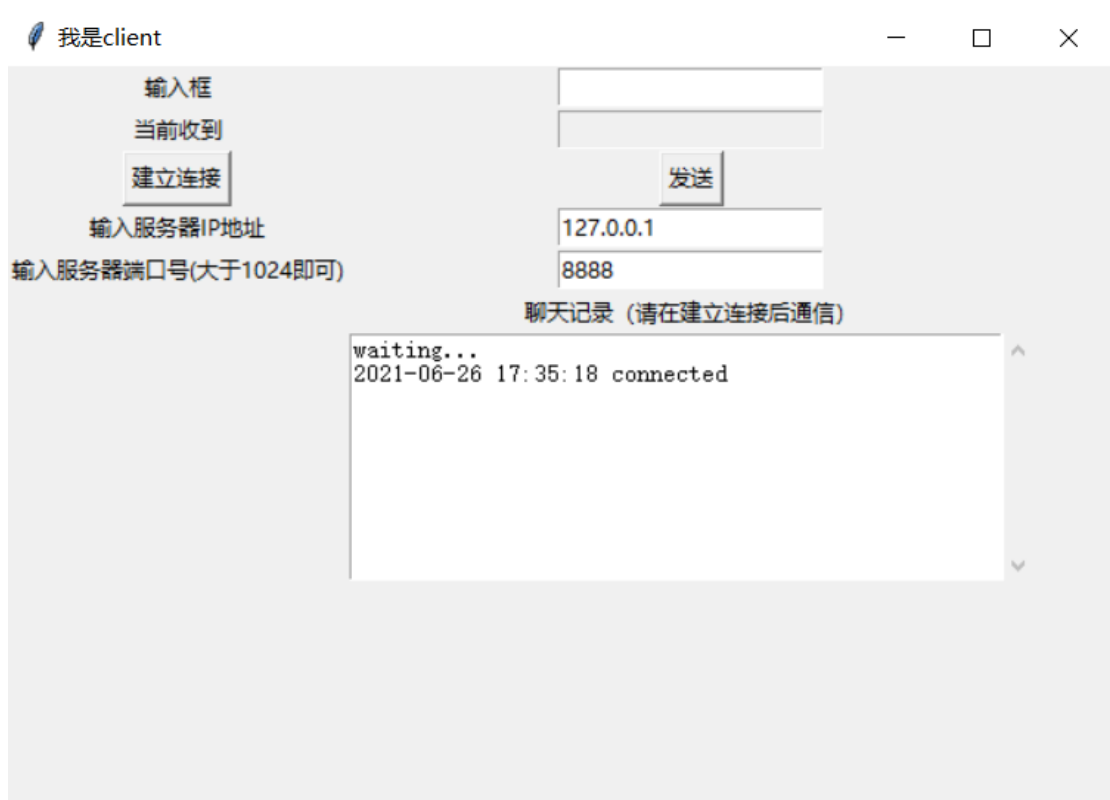
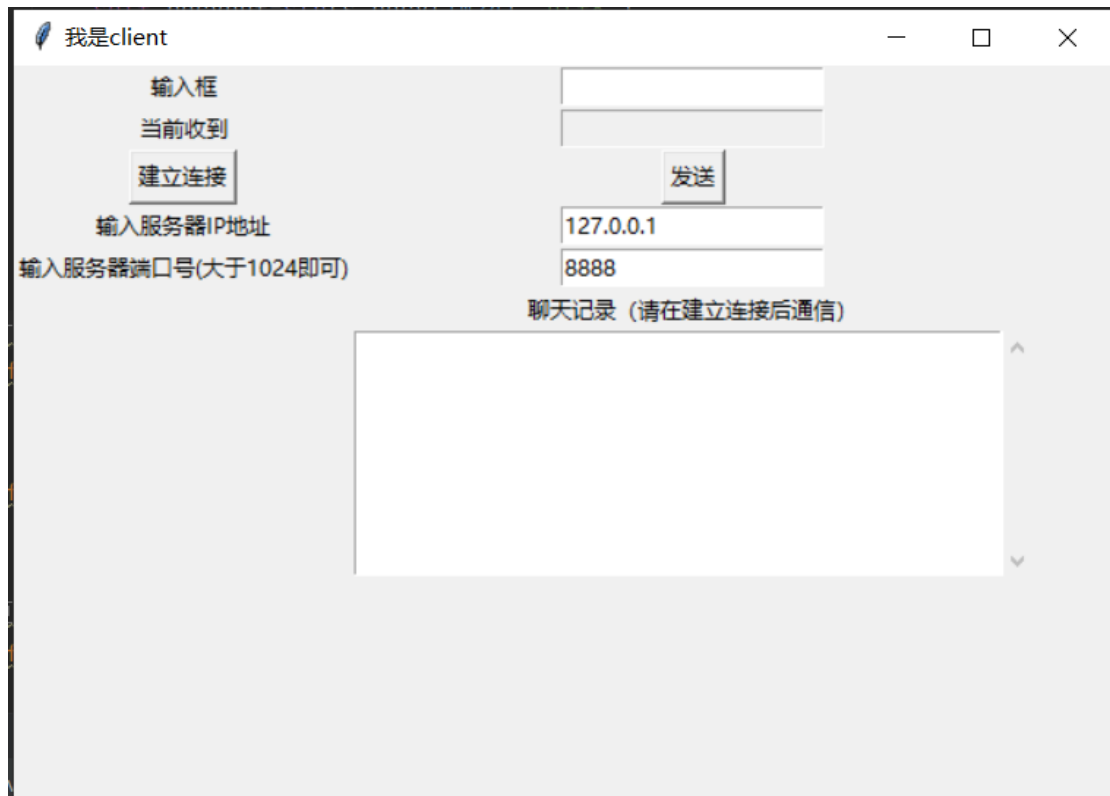
```

2. 系统过程演示（本代码可以在局域网同一网段运行）

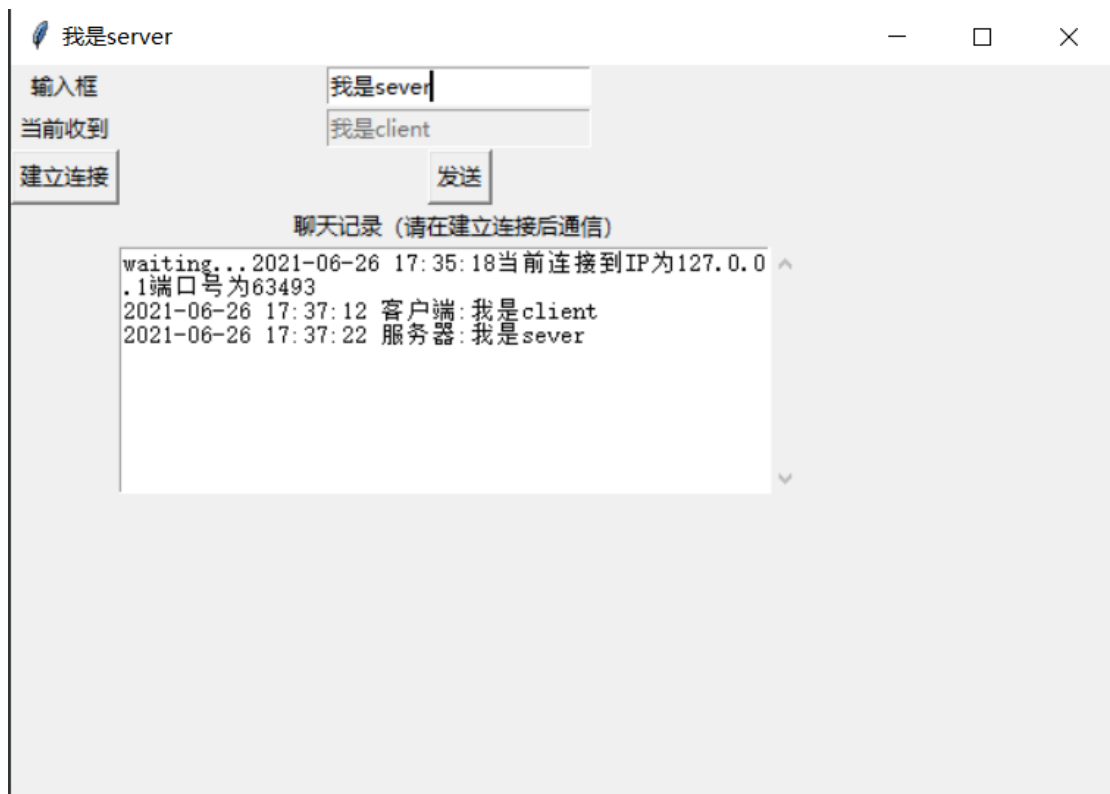
- (1) 首先启动 sever，然后点击连接，连接成功



(2) 启动 client，点击连接连接成功



(3) 然后进行聊天操作（这个聊天可以在同一网段上进行）



3. 开发总结

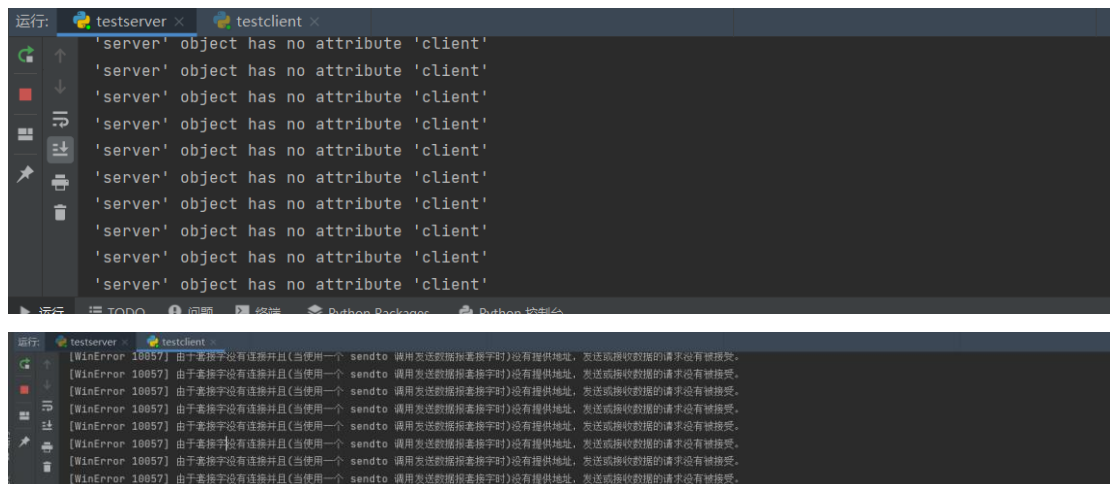
开发时遇到的问题:

开发代码时, 会遇到 client 和 sever 是否连接成功问题, 所以在开发时用了发送指令观察是否连接成功, 于是就开发了在未连接的情况, 一直发送未连接指令, 单独启动时会发送信息

提

示

:



The image shows two screenshots of a Python IDE's terminal window. The top screenshot shows the output of a testserver process, which repeatedly prints the message: `'server' object has no attribute 'client'`. The bottom screenshot shows the output of a testclient process, which repeatedly prints a `WinError 10057` message. The message text is: `[WinError 10057] 由于套接字没有连接并且(当使用一个 sendto 调用发送数据给套接字时)没有提供地址, 发送或接收数据请求没有被接受。`

实验总结：在这个过程中我学会了 C/S 结构，以及 python 代码的编写和基本的 python 实时聊天系统的架构，以及学会了在局域网内同一网段上进行实时聊天的操作