TCP Chatting Room

1. Server

1.1 config

```python
# 协议配置
REQUEST_LOGIN = "0001"  # 登录请求
REQUEST_CHAT = "0002"  # 聊天请求
RESPONSE_LOGIN_RESULT = "1001"  # 登录结果响应
RESPONSE_CHAT = "1002"  # 聊天响应
DELIMITER = '|'  # 分隔符

# 服务器配置
SERVER_IP = "127.0.0.1"
SERVER_PORT = 8090

# 客户端账号
accounts = {
    "user1" : ["114514", "konbi"],
    "admin" : ["HytidelSB", "Hytidel"]
}
```

1.2 response_protocol

```python
import Ipynb_importer
from config import *

class ResponseProtocol(object):  # 服务器响应协议的格式字符串处理
    @staticmethod
    def response_login_result(result, nickname, username):  # 生成用户登录结果的字符串
        '''
        :param result: 值为0表示登录失败，值为1表示登录成功
        :param nickname: 登录用户的昵称，若登录失败则为空
        :param username: 登录用户的账号，若登录失败则为空
        :return: 返回给用户的登陆结果的协议字符串
        '''
        return DELIMITER.join([RESPONSE_LOGIN_RESULT, result, nickname, username])
    @staticmethod
    def response_chat(nickname, messages):  # 生成返回给用户的消息字符串
        '''
        :param nickname: 发送消息的用户的昵称
        :param messages: 消息正文
        :return: 返回给用户的消息字符串
        '''
        return DELIMITER.join([RESPONSE_CHAT, nickname, messages])
```

1.3 server_socket

```
import Ipynb_importer
import socket
from config import *

class ServerSocket(socket.socket):  # 自定义套接字，用于初始化服务器套接字所需的参数
    def __init__(self):  # 设置为TCP类型
        super(ServerSocket, self).__init__(socket.AF_INET, socket.SOCK_STREAM)  # 继承父类

        self.bind((SERVER_IP, SERVER_PORT))  # 绑定地址和端口号
        self.listen(128)  # 最多允许128个客户端同时连接

```

1.4 socket_wrapper

```
class SocketWrapper(object):  # 包装套接字
    def __init__(self, sock):
        self.sock = sock

    def recv_data(self):  # 接收客户端发送的消息并解码为字符串
        try:  # 客户端正常发送消息
            return self.sock.recv(512).decode("utf-8")  # 每个消息最多512个字符；接收的是二进制流，需解码
        except:  # 客户端退出
            return "/logout"

    def send_data(self, message):  # 编码字符串并向客户端发送消息
        return self.sock.send(message.encode("utf-8"))  # 发送的是二进制流，需编码

    def close(self):  # 关闭套接字
        self.sock.close()

```

1.5 server

```
import Ipynb_importer
from server_socket import ServerSocket
from socket_wrapper import SocketWrapper
from threading import Thread
from response_protocol import ResponseProtocol
from config import *

class Server(object):  # 服务器核心
    def __init__(self):  # 创建服务器套接字
        self.server_socket = ServerSocket()

        # 存不同请求对应的函数的字典
        self.request_handle_function = {}
        self.register(REQUEST_LOGIN, self.request_login_handle)
        self.register(REQUEST_CHAT, self.request_chat_handle)

        self.clients = {}  # 保存当前在线用户的字典
```

```
18
19      def register(self, request_id, handle_function):  # 注册消息与处理函数到字典中
20          self.request_handle_function[request_id] = handle_function
21
22      def startup(self):  # 获取客户端连接并提供服务
23          while True:
24              # 获取客户端连接
25              print("Waiting for connection ...")
26              soc, addr = self.server_socket.accept()
27              print("Connected to the client sucessfully.")
28
29              # 收发消息
30              client_soc = SocketWrapper(soc)
31
32              # 开启多线程: 一般写法
33              # thread = Thread(target = self.request_handle, args = (client_soc, ))  # 注意只有一个元素的元组也要加,
34              # thread.start()
35
36              # 开启多线程: Lambda函数写法
37              Thread(target = lambda: self.request_handle(client_soc, )).start()
38
39              # soc.close()  # 主线程不关闭客户端套接字
40
41      def request_handle(self, client_soc):  # 处理客户端请求
42          while True:
43                  # 接收客户端消息
44                  message = client_soc.recv_data()
45                  if message == "/logout": break
46                  # print(message)
47                  # client_soc.send_data("The server got " + message)  # 向客户端发送提示信息
48
49                  # 解析数据
50                  parse_data = self.parse_request_text(message)
51
52                  # 分析请求类型并处理
53                  # print("parse_data: %s" % parse_data)
54
55                  # handle_function = self.request_handle_function[parse_data["request_id"]]  # 这样写key不存在时会报错
56                  handle_function = self.request_handle_function.get(parse_data["request_id"])
57                  if handle_function:  # 若存在该函数则调用
58                      handle_function(client_soc, parse_data)
59
60
61          self.remove_offline_user(client_soc)
62          client_soc.close()  # 关闭客户端套接字
63
64      def remove_offline_user(self, client_soc):  # 处理下线的客户端
65          # print("Connection closed.")
66          for username, info in self.clients.items():
67              if info["sock"] == client_soc:
68                  del self.clients[username]
69                  print("Now online: ")
70                  print(self.clients)
71                  break
72
```

```
73
74      def parse_request_text(self, text):  # 解析客户端发送的数据
75          '''
76          登录请求: 0001 | username | password
77          聊天请求: 0002 | username | message
78          '''
79          print("The client sent: " + text)
80
81          # 按 | 分割消息
82          request_list = text.split(DELIMITER)
83          request_data = {}
84          request_data["request_id"] = request_list[0]
85
86          if request_data["request_id"] == REQUEST_LOGIN:  # 登录请求
87              request_data["username"] = request_list[1]
88              request_data["password"] = request_list[2]
89          elif request_data["request_id"] == REQUEST_CHAT:  # 聊天请求
90              request_data["username"] = request_list[1]
91              request_data["message"] = request_list[2]
92          return request_data
93
94      def request_login_handle(self, client_soc, request_data):  # 处理登录请求
95          # print("A login handle was got")
96
97          # 获取账号、密码
98          username = request_data["username"]
99          password = request_data["password"]
100
101         # 检查是否能登录
102         result, nickname, username = self.check_user_login(username, password)
103         if result == "1":  # 若登录成功则保存在线用户
104             self.clients[username] = { "sock" : client_soc, "nickname" : nickname }
105
106         # 拼接消息并发给客户端
107         response_text = ResponseProtocol.response_login_result(result, nickname,
    username)
108         client_soc.send_data(response_text)
109
110     def check_user_login(self, username, password):  # 检查用户能否登录成功, 若能则result =
    "1", 否则result = "0"
111         if username not in accounts:
112             return "0", username , ""
113         elif password != accounts[username][0]:
114             return "0", username , ""
115         else:
116             return "1", accounts[username][1], username
117
118     def request_chat_handle(self, client_soc, request_data):  # 处理聊天请求
119         # print("A chat handle was got.")
120
121         # 获取消息内容
122         username = request_data["username"]
123         message = request_data["message"]
124         nickname = self.clients[username]["nickname"]
125
126         msg = ResponseProtocol.response_chat(nickname, message)  # 拼接发送给客户端的消息
127
128         # 将消息转化给除发送者外的在线用户
```

```
        for _username, info in self.clients.items():
            if username != _username:
                info["sock"].send_data(msg)

if __name__ == "__main__":
    Server().startup()
```

2. Client

2.1 config

```
# 协议配置
REQUEST_LOGIN = "0001"  # 登录请求
REQUEST_CHAT = "0002"  # 聊天请求
RESPONSE_LOGIN_RESULT = "1001"  # 登录结果响应
RESPONSE_CHAT = "1002"  # 聊天响应
DELIMITER = '|'  # 分隔符

# 服务器配置
SERVER_IP = "127.0.0.1"
SERVER_PORT = 8090
```

2.2 request_protocol

```
import Ipynb_importer
from config import *

class RequestProtocol(object):
    @staticmethod
    def request_login_result(username, password):
        # 0001|usermame|password
        return DELIMITER.join([REQUEST_LOGIN, username, password])

    @staticmethod
    def request_chat(username, message):
        # 0002|username|message
        return DELIMITER.join([REQUEST_CHAT, username, message])
```

2.3 client_socket

```
import Ipynb_importer
import socket
from config import *

class ClientSocket(socket.socket):  # 客户端套接字
    def __init__(self):
        super(ClientSocket, self).__init__(socket.AF_INET, socket.SOCK_STREAM)  # 设置为TCP
套接字

    def connect(self):  # 自动连接到服务器
```

```
        super(ClientSocket, self).connect((SERVER_IP, SERVER_PORT))

    def recv_data(self):  # 接收服务器发送的数据并解码为字符串
        return self.recv(512).decode("utf-8")

    def send_data(self, message):  # 编码并发送数据
        return self.send(message.encode("utf-8"))
```

2.4 window_login

```
from tkinter import Tk
from tkinter import Label, Entry, Frame, Button
from tkinter import LEFT, END

class WindowLogin(Tk):  # 登录窗口
    def __init__(self):
        super(WindowLogin, self).__init__()  # 调用父类方法初始化窗口
        self.window_init()  # 设置窗口属性
        self.add_widgets()  # 填充控件

        self.reset_button_click(lambda: (self.clear_username(), self.clear_password()))

    def window_init(self):  # 设置窗口属性
        self.title("Login")  # 设置窗口标题

        self.resizable(False, False)  # 设置窗口不可拉伸

        # 设置窗口位置
        window_width, window_height = 255, 95
        screen_width, screen_height = self.winfo_screenwidth(), self.winfo_screenheight()
        pos_x, pos_y = (screen_width - window_width) / 2, (screen_height - window_height) / 2
        self.geometry("%dx%d+%d+%d" % (window_width, window_height, pos_x, pos_y))

    def add_widgets(self):  # 填充空间
        # 用户名标签
        username_label = Label(self)
        username_label["text"] = "Username: "
        username_label.grid(row = 0, column = 0, padx = 10, pady = 5)  # 其他控件会与该控件的间隔对齐

        # 用户名输入框
        username_entry = Entry(self, name = "username_entry")
        username_entry["width"] = 20
        username_entry.grid(row = 0, column = 1)

        # 密码标签
        password_label = Label(self)
        password_label["text"] = "Password: "
        password_label.grid(row = 1, column = 0)

        # 密码输入框
        password_entry = Entry(self, name = "password_entry")
        password_entry["width"] = 20
        password_entry["show"] = '*'
        password_entry.grid(row = 1, column = 1)
```

```

        # 创建框架
        button_frame = Frame(self, name = "button_frame")
        button_frame.grid(row = 2, columnspan = 2, pady = 5)

        # 重置按钮
        reset_button = Button(button_frame, name = "reset_button")
        reset_button["text"] = "Clear"
        reset_button.pack(side = LEFT, padx = 20)  # 两按钮的间距

        # 登录按钮
        login_button = Button(button_frame, name = "login_button")
        login_button["text"] = "Login"
        login_button.pack(side = LEFT)

    def get_username(self):  # 获取输入的用户名
        return self.children["username_entry"].get()

    def get_password(self):  # 获取输入的密码
        return self.children["password_entry"].get()

    def clear_username(self):  # 清空用户名输入框
        self.children["username_entry"].delete(0, END)

    def clear_password(self):  # 清空密码输入框
        self.children["password_entry"].delete(0, END)

    def reset_button_click(self, command):  # 重置按钮的响应注册
        reset_button = self.children["button_frame"].children["reset_button"]
        reset_button["command"] = command

    def login_button_click(self, command):  # 登录按钮的响应注册
        login_button = self.children["button_frame"].children["login_button"]
        login_button["command"] = command

    def window_close(self, command):  # 窗口关闭事件的处理
        self.protocol("WM_DELETE_WINDOW", command)

if __name__ == "__main__":
    window = WindowLogin()
    window.mainloop()
```

2.5 window_chat

```
import Ipynb_importer
from tkinter import Toplevel
from tkinter.scrolledtext import ScrolledText
from tkinter import Text, Button
from tkinter import UNITS, END
from time import localtime, strftime, time

class WindowChat(Toplevel):  # 聊天窗口
    def __init__(self):
        super(WindowChat, self).__init__()  # 初始化聊天窗口

        self.geometry("%dx%d" % (795, 505))  # 设置窗口大小
```

```
        self.resizable(False, False)  # 设置窗口不可拉伸

        self.add_widgets()  # 填充组件

    def add_widgets(self):  # 填充组件
        # 聊天区
        chat_text_area = ScrolledText(self)
        chat_text_area["width"] = 110
        chat_text_area["height"] = 30
        chat_text_area.grid(row = 0, column = 0, columnspan = 2)

        # 初始化颜色标签
        chat_text_area.tag_config("user", foreground = "green")
        chat_text_area.tag_config("system", foreground = "red")
        self.children["chat_text_area"] = chat_text_area

        # 输入区
        chat_input_area = Text(self, name = "chat_input_area")
        chat_input_area["width"] = 100
        chat_input_area["height"] = 7
        chat_input_area.grid(row = 1, column = 0, pady = 10)

        # 发送按钮
        send_button = Button(self, name = "send_button")
        send_button["text"] = "Send"
        send_button["width"] = 5
        send_button["height"] = 2
        send_button.grid(row = 1, column = 1)

    def set_title(self, title):  # 设置窗口标题
        self.title("Welcome % s!" % title)

    def send_button_click(self, command):  # 注册发按钮点击的事件
        self.children["send_button"]["command"] = command

    def get_inputs(self):  # 获取文本框内容
        return self.children["chat_input_area"].get(0.0, END)  # 获取从0.0开始到字符串结束的内
容


    def clear_inputs(self):  # 清空文本框内容
        self.children["chat_input_area"].delete(0.0, END)

    def append_message(self, sender, message):  # 添加消息到聊天区
        send_time = strftime("%Y-%m-%d %H:%M:%S", localtime(time()))  # 将获取到的时间转化为当
前时区的时间
        send_info = "%s: %s\n" % (sender, send_time)
        self.children["chat_text_area"].insert(END, send_info, "user")
        self.children["chat_text_area"].insert(END, " " + message + "\n")

        self.children["chat_text_area"].yview_scroll(3, UNITS)  # 向下滚动屏幕

    def window_close(self, command):  # 窗口关闭时的事件
        self.protocol("WM_DELETE_WINDOW", command)

if __name__ == "__main__":
    WindowChat().mainloop()
```

2.6 client

```python
import Ipynb_importer
from window_login import WindowLogin
from request_protocol import RequestProtocol
from client_socket import ClientSocket
from threading import Thread
from config import *
from tkinter.messagebox import showinfo
from window_chat import WindowChat
import os

class Client(object):  # 客户端核心
    def __init__(self):  # 初始化客户端资源
        # 初始化登录窗口
        self.window = WindowLogin()
        self.window.reset_button_click(self.clear_inputs)
        self.window.login_button_click(self.send_login_data)
        self.window.window_close(self.exit)  # 关闭窗口时退出程序

        # 初始化聊天窗口
        self.window_chat = WindowChat()
        self.window_chat.send_button_click(self.send_chat_data)
        self.window_chat.withdraw()  # 隐藏窗口
        self.window_chat.window_close(self.exit)  # 关闭窗口时退出程序

        self.soc = ClientSocket()  # 创建客户端套接字

        # 初始化消息处理函数
        self.response_handle_function = {}
        self.register(RESPONSE_LOGIN_RESULT, self.response_login_handle)
        self.register(RESPONSE_CHAT, self.response_chat_handle)

        self.username = None  # 登录用户账号
        self.self = None  # 登录用户昵称

        self.is_running = True  # 程序正在运行的标记

    def register(self, request_id, handle_function):  # 注册响应和对应的方法到字典中
        self.response_handle_function[request_id] = handle_function

    def startup(self):  # 开启窗口
        self.soc.connect()
        Thread(target = self.response_handle).start()  # 在mainloop前开启接收消息的子线程
        self.window.mainloop()

    def clear_inputs(self):  # 清空文本框内容
        self.window.clear_username()
        self.window.clear_password()

    def send_login_data(self):  # 发送登录消息到服务器
        # 获取输入的账号和密码
        username = self.window.get_username()
        password = self.window.get_password()

        # 发送登录消息到服务器
        request_text = RequestProtocol.request_login_result(username, password)
```

```
56          self.soc.send_data(request_text)
57
58          # 接收登录结果
59          # recv_data = self.soc.recv_data()
60          # print(recv_data)
61
62      def send_chat_data(self):  # 发送聊天消息到服务器
63          message = self.window_chat.get_inputs()
64          self.window_chat.clear_inputs()  # 清空文本框
65
66          self.window_chat.append_message(self.nickname, message)  # 将消息添加到聊天区
67
68          request_text = RequestProtocol.request_chat(self.username, message)  # 拼接协议文本
69          self.soc.send_data(request_text)
70
71      def response_handle(self):  # 接收服务器消息
72          while self.is_running:
73              try:
74                  recv_data = self.soc.recv_data()  # 接收服务器消息
75                  # print("A message was got: " + recv_data)
76                  response_data = self.parse_response_data(recv_data)  # 解析消息
77
78                  # 根据响应类型分别处理
79                  handle_function =
    self.response_handle_function[response_data["response_id"]]
80                  if handle_function:
81                      handle_function(response_data)
82              except:
83                  break
84
85      @staticmethod
86      def parse_response_data(recv_data):  # 解析消息
87          '''
88          登录响应: 1001|0/1|nickname|username
89          聊天响应: 1002|nickname|message
90          '''
91          response_data_list = recv_data.split(DELIMITER)  # 按分隔符分割消息
92
93          # 解析消息的各部分
94          response_data = {}
95          response_data["response_id"] = response_data_list[0]
96
97          if response_data["response_id"] == RESPONSE_LOGIN_RESULT:  # 登录响应
98              response_data["result"] = response_data_list[1]
99              response_data["nickname"] = response_data_list[2]
100             response_data["username"] = response_data_list[3]
101         elif response_data["response_id"] == RESPONSE_CHAT:  # 聊天响应
102             response_data["nickname"] = response_data_list[1]
103             response_data["message"] = response_data_list[2]
104
105         return response_data
106
107     def response_login_handle(self, response_data):  # 登录响应
108         print("A login answer received", response_data)
109         result = response_data["result"]
110
111         # 登录失败
112         if result == "0":
```

```
113             showinfo("Login failed.", "Account or password error!")
114             return
115 
116         # 登录成功
117         self.nickname = response_data["nickname"]  # 保存登录用户的昵称，供将消息添加到聊天区使
    用
118         self.username = response_data["username"]  # 保存登录用户的账号，供发送消息使用
119         showinfo("Login successfully.", "Hello " + self.nickname + "!")
120 
121         # 显示聊天窗口
122         self.window_chat.set_title(self.username)
123         self.window_chat.update()  # 刷新窗口内容
124         self.window_chat.deiconify()
125 
126         self.window.withdraw()  # 隐藏登录窗口
127 
128     def response_chat_handle(self, response_data):  # 聊天响应
129         # print("A chat answer received", response_data)
130         sender = response_data["nickname"]
131         message = response_data["message"]
132         self.window_chat.append_message(sender, message)
133 
134     def exit(self):  # 释放资源并退出程序
135         self.is_running = False  # 停止子线程
136         self.soc.close()  # 关闭套接字
137         os._exit(0)  # 退出程序
138 
139 if __name__ == "__main__":
140     client = Client()
141     client.startup()
```