第24章

寥入聊天案

网络社交软件的普及，大大方便了我们的生活和沟通。比如我们都用过的QQ群聊.微信 群聊等，就可以实现多人实时聊天的功能。只需要将多个人组建为一个群，就可以在该群的聊 天窗口畅所欲言，群里的每个人都能在本地窗口看到所有人的发言内容。

使用Python语言也可以实现多人聊天功能，本章就带大家实现一款简单的多人聊天室程 序，该程序使用面向对象编程思想进行设计和实现，并且会对前面章节学习的Python GUI库 (Tkinter).多任务编程、网络编程以及数据库编程等相关知识加以应用。

24o 1项圈疔纸

24JJ囊圜矚述

多人聊天室是一款c/s (客户/服务器)模式的软件，它由服务器端程序和客户端程序两 部分组成。客户端发送聊天信息到服务器端，由服务器端将聊天信息转发给其他在线登录用户。

多人聊天室的实现原理如图244所示。

客户^john

燃好  
服务器将嚇好'转发给其他在  
线客户竭  
你好

……'\*…………”…—；笈送聊竇.海蓉"你好”到服务器 客户端smith *D”*

图24』多人聊天室的实现原理

限于篇幅，本章实现的多人聊天室只具备多人聊天的基本功能，并不像QQ群聊功能那么 强大。但是大家理解了其中的设计思路，就可以根据自己的需求开发一款功能更强大的聊天室 程序。

本项目基于Windows 7操作系统，使用的开发语言是Python 3。6，开发工具是PyCharm。 大家在学习时，应当对项目的环境和语言有基本的了解，做到心中有数。

本项目由客户端和服务器端组成,其中客户端包含图形化界面，服务器端不具有图形化界面。 在演示程序时，应首先启动服务器端程序，等待客户端连接。由于服务器端不包含图形化 界面，暂不演示服务器端执行效果。

服务器端程序运行之后，即可启动客户端程序。客户端的主要功能包括用户登录和聊天室 聊天。

1. 用户登录

客户端启动后，首先弹出登录窗口，如图24-2所示。

用户在登录窗口输入用户名和密码信息?单击“登录”按钮将用户名和密码发送给服务器端。 服务器端对用户名和密码进行验证，如果验证通过则登录成功，显示登录成功窗口，如图24-3 所示。

如果验证不通过则登录失败，显示登录失败窗口，如图24・4所示。

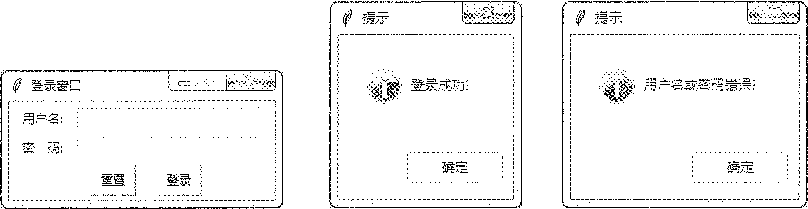


图24-2 登录窗口 图24； 登录成功 图24-4 登录失败

1. 聊天室聊天

客户端程序登录成功之后，将自动进入聊天室界面须如图24・5所示。

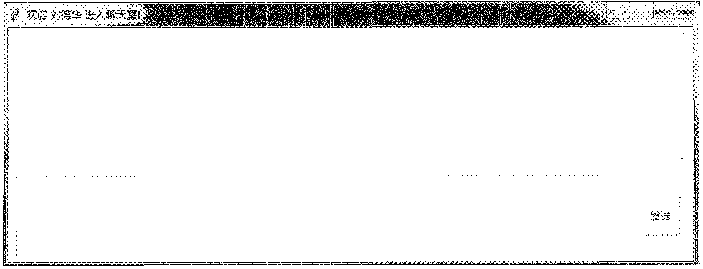


图2心5聊天室界面

用户在聊天室界面下方的输入框内输入聊天内容，单击“发送”按钮，就可以将聊天内容 发送给其他在线用户。当多人同时在线时，所有在线用户都会接收到某个用户发送的聊天内容，

多人聊天界面效果如图24-6所示。

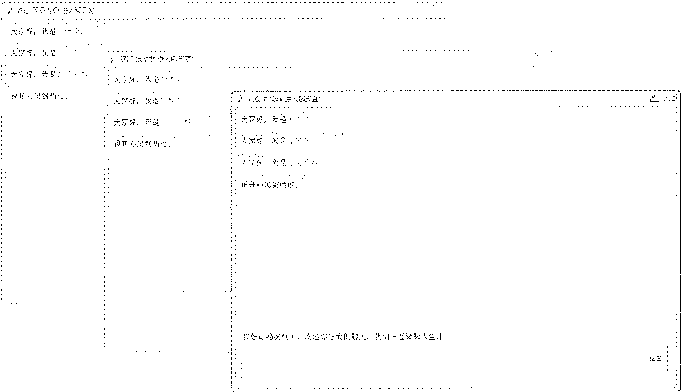
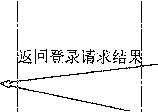


图24-6多人聊天界面

接下来，我们用一张图来简单介绍客户端和服务器端程序运行的总体流程，如图2J7所示。

客户端程序 服务器端程序



登录窗口

服务器 |

.开启单独线程| 线程1：处方矗7] | 的请求 ！ r~—n I

一登录请求处理 j

= I 〜聊天请求处理 j

.2=11 i  
匚一一―——L-一一一. ^一^ J

聊天窗口

发送登录请求

r发送聊天请求

新线程

登录响应处理

悴回聊天请求结耳

聊天响应处理

图24-7多人聊天室总体流程

从流程图可以看出，以客户端作为起点来介绍，则多人聊天室的大致流程如下：

（1） 客户端登录窗口输入用户名密码，单击“发送”按钮向服务器发送登录请•求。

（2） 服务器端接收到客户端连接之后，开启单独线程来处理该客户端所有的请求。这时 首先收到客户端的登录请求，调用登录请求处理方法，处理完毕之后，将登录处理结果（登录

响应）发送给客户端。

（3） 客户端收到了服务器端返回的登录结果。如果登录成功，则显示聊天窗口。

（4） 客户端在聊天窗口输入聊天内容，单击“发送”按钮，向服务器端发送聊天请求。

（5） 服务器端收到聊天请求，交由聊天请求处理方法来处理。处理完毕之后，转发聊天 内容给所有在线客户端。

（6） 客户端收到聊天内容,调用聊天内容处理方法,将聊天内容显示到聊天窗口对应控 件上。

24e2服务器端程序设计

多人聊天室服务器端作为服务的提:供者，负责用户登录验证、聊天信息转发等功能。那么 为什么需要服务器端程序呢？原因是，如果我们想要实现多人聊天，那么任意两个客户端之间 就必须建立连接，如图24・8所示。

如果让客户端之间两两直接相连，一则我们无法及时知晓哪些用户在线或者离线，再则， 随着用户量的增多，要让每个客户端与其他客户端都建立连接是一个庞大而复杂的工程，可想 而知其效率也是无比低下的。

因此，我们改变设计思路，加入一个服务器端程序，让服务器端程序负责保存所有在线用 户的连接。在线用户将信息发送给服务器端，服务器端再转发给其他用户，这样每个客户端只 需要与服务器端建立连接，而服务器端只需要与在线客户端保持连接，从而大大减少了连接的 总数量。这样的设计让我们的程序结构更简单，效率更高（该设计图见图244 ）。

客户端edwaird ;

客户端john

图24-8客户端之间互相连接

明确了服务器需要具备的功能之后，就可以设计服务器端的业务流程了，也就是服务器该 如何工作。如图24-9所示,我们设计的服务器工作流程主要包括以下步骤：

（1 ）首先服务器在指定的端口进行监听，等待客户端连接。

（2 ）客户端连接到服务器之后，服务器开启单独线程来处理该用户的请求。

（3） 服务器端在处理线程中等待客户端发送请求。

（4） 接收到客户端请求后，服务器端根据客户端请求类型不同，调用不同的处理方法。

（5） 处理完客户端请求之后，再次回到第（3）步等待接收客户端新的请求。

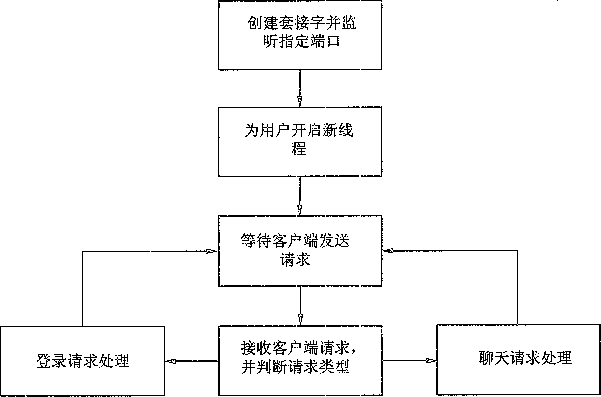


图24-9服务器端工作流程

如果客户端退岀登录，则服务器端也会关闭对应客户端的处理线程，以释放资源。要注意 的是，这里选择开启新的线程来处理客户端请求，是由于服务器端面向的是多客户端请求的处 理，如果不使用多任务技术歹就无法实现客户端即时聊天的要求。

为了实现客户端和服务器端之间的通信，我们需要约定客户端和服务器端数据传输的格式， 即约定客户端发送什么格式的数据给服务器，服务器又返回什么样格式的数据给客户端。客户 端会有不同的请求，所以我们针对不同的请求和响应定义了请求和响应编号，来区分不同的请 求和响应。

们釁矗晌应洛式

针对客户端登录请求,服务器端的响应格式f?1001|ret|nickname|usemameH9其中1001 代表登录结果响应，冀t代表服务器端验证的结果，如果是字符疗，则表示服务器端验证失败， 后面的nickname就是空字符串;如果ret的值是表示服务器端验证成功,则nickname为 服务器返回的该用户的昵称? username为该用户的用户名。

2.聊天响应檣式

对于客户端聊天请求,服务器端的响应格式为：“1002|nicknanw|nie%agesL其中1002代 表聊天响应，nickname为聊天信息发送者的昵称，messages 其发送的聊天信息。

我们在自定义配置文件config^py中定义响应协议和请求协议中需要的常量，包括以下内容。

# 数据协议相关配置-----

REQUEST\_LOGIN = 1 0001? REQUEST\_CHAT = !0002, RESPONSE\_LOGIN\_RESULT = '1001 \* RESPONSE\_CHAT = ?1002! DELIMITER = ? |'

#登录请求

#聊天请求

#登录结果响应

#聊天响应

#自定义协议数据分隔符

在文件中，还定义了登录请求的编码是00019聊天请求的编码是0002,服务器端需要根 据这两个编码来识别客户端请求的类型。实际上，这段数据协议相关的配置内容在客户端程序 和服务器端程序是一样的。

为了方便程序的管理和维护，我们将服务器端程序的相对独立功能使用单独的类和模块来 实现。具体设计的类如表24』所示。

表24-1服务器端程序中类的设计

|  |  |
| --- | --- |
| 类名称 | *含* 义 |
| Server | 服务器端程序类，负责服务器端的运行流程 |
| ServerSocket | 服务器套接字类，用于创建服务器端连接的套接字 |
| Socket Wrapper | 套接字包装类，用于包装客户端套接字，指定传输编码格式 |
| Responseprotocol | 响应拼接类，用于按照协议规定的格式拼接服务器端响应 |
| DB | 数据库交互类，用于与数据库的交互操作 |

同时，在项目中创建若干个模块，每个类单独放在一个模块里，另外还需要一个单独的配 置文件。设计后的模块如表24・2所示。

表24・2模块设计

|  |  |
| --- | --- |
| 模块名称 | 含 义 |
| ser/er.py | 用于包含Server类 |
| server\_socket.py | 用于包含ServerSocket类 |
| socket\_Avrapper.py | 用于包含SockefWrapper类 |
| responseprotocol.py | 用于包含 ResponseProtocol 类 |
| db.py | 用于包含DB*类* |
| config.py | 配置文件，用于保存程序的设置值 |

由于本项目有用户登录功能，所以使用数据库来保存现有用户信息。本项目使用MySQL 数据库管理工具（版本为5.7 ）,大家可以使用MySQL Workbench工具提供的GUI对MySQL 数据库进行管理。

本项目的数据库名称为mini\_chm它包含了一个名为iiseis的表，用于存储多人聊天室系 统的用户信息。users表的结构如图2440所示:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 3-……］ | --”・ |  |
| H…… | I島（ | — | PIU ] NULL | auto\_tncF^ent | |
| *1* user | ! 73rchar(3'(5): | NO ： | 1 null I ' . ■ 1; |
| *1* user\_password | *1* 心洲）*1* | 興 1 |  |
|  |  | %丄 |  |
|  | 图 2440 | users | ；表的结构 |

可以使用如下SQL语句来创建该表:

use mini\_chat;

CREATE TABLE 'users!(

!user\_id! int (11) NOT NULL AUTO\_INCREMENT,

?user\_name? varchar(30) CHARACTER SET utf8 NOT NULL,

? user\_password \* varchar(30) CHARACTER SET utf8 NOT NULL,

J user\_nickname \* varchar(20) CHARACTEFl SET utf8 NOT NULL,

-PRIMARY KEY (!user\_id,)

)ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=latinl;

然后在该表中添加若干测试数据，用于后续测试用户登录和聊天功能，如表24；所示。

表24-3用户表测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| userjd | user\_name | user\_password | user\_nickname |
| 1 | john | 123 | john |
| 2 | smith | 123 | smith |
| 3 | edward | 123 | ed |
| llllllW | emma | 123 | em |
| 5 | elsa | 123 | elsa |
| 6 | mickey | 123 | mi |

! 曝绪器端程序实现

我们知道，多人聊天室项目包含服务器端和客户端两个独立的程序。在明确了服务器端程 序的设计之后，就可以先实现服务器端程序了。

24.3J 劎篷嚥圓浏支繹墓樗

首先使用PyChami新建一个项目，取名为Sorv就。然后按照服务器端程序中模块的设计， 在该项目下分别创建server.py, server\_socket.py, socket\_wrapper.py, response\_protocoLpy, db.py和config.py文件。创建完成后的文件结构如图2441所示。

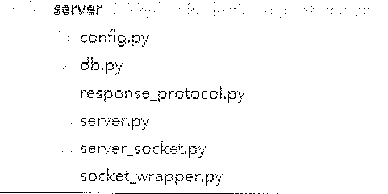


图24-11服务器端程序的文件结构

24n3n2 黨勰 SerwerSocket 襲

ServerSocket类中要创建服务器端的套接字，首先我们要为服务器端设置访问][P和端口 , 为此，在配置文件config.py中添加如下内容。

服务器相关配置

SERVER\_IP = ?127.0 o 0.1?

#服务器工P地址

#服务器端口

SERVER PORT = 8090

接下来在server\_socketopy文件中实现ServerSocket类的功能，包括绑定套接字和监听功能。 其代码如下所示。

from socket import socket, AF\_INET, SOCK\_STREAM from config import SERVER\_IP, SERVER\_PORT class ServerSocket(socket):

“5包装服务器端套接字类“5

def init (self):

#初始化套接字

super(ServerSocket, self). init (AF\_INET, SOCK\_STREAM) #绑定IP和PORT

self obind( (SERVER\_\_IP, SERVER\_PORT))

#设置为被动监听套接字

self.listen(128)

24d3o3 S@ck®tWrapp®r

Socketwrapper类用于封装客户端端口，在发送信息和接收信息时进行编码和解码，以便 处理中文字符。为此,在socket wrapper.py文件中实现如下代码。

class Socketwrapper(object): n n °包装客户端从接字类"°n

def init (self, sock): self.sock = sock

def recv\_data(self): n " n接收数据,s n "

return self.sock.recv(512) .decode("utf-8H)

def send\_data(self, messages):

n ""麦送数据° n n

return self.sock o send(messages.encode("utf-8"))

24.3.4褰嬲DB熒

DB类负责与数据库交互，由于数据库的地址等信息是可变的，所以一般把数据库信息进 行可配置化。为此，先把数据库IP地址及登录信息等保存在配置文件config.py中,代码如下所示。

# -一-数据库相关配置\_\_

DB\_HOST = ?127.0.0.1,

#数据库连接地址

#数据库登录用户名

#数据库登录密码

#数据库端口

#数据库名

DB\_USER = Jroot1

DB\_PASS = 1 123456!

DB\_PORT = 3306

DB NAME = 'mini chat1

然后在dbpy文件中添加代码，实现DB类的功能。DB类有一个get\_one方法，它的功能 是执行通过参数传入的SQL语句，并且将返回结果按字段名为key,值为vain©的格式存储为 一个字典。该方法可用于从数据库查询特定用户的信息。dbpy文件的代码如下所示。

from pymysql import connect from config import \*

class DB(object):

5"数据库交互类° ° "

def init (self):

° ° n初始化数据库连接n ° °

#创建数据库连接

self.conn = connect(host=DB\_HOST, user = DB\_\_USER, password = DB\_\_PASS, database=DB\_N2kME, port = DB\_PORT)

#获得游标

self.cursor = self.conn.cursor()

def get\_one(self, sql):

° ° °执行SQL查〒旬n ° ,f

#执行SQL查询 self.cursor.execute(sql)

#查询结果

query\_result = self.cursor.fetchone()

if not query\_result:

return None

#获得字段列表

fields = [field. [0 ] for field in self. cursor. description] #保存返回结果

return\_data = diet()

for field, value in zip (fields, query\_result): return\_data [field] = value

return return\_data

def close (self):

1 ? \*关闭数据库连接斗八

self.cursor.close() self.conn.close()

在db.py文件的最后，使用如下测试代码进行测试。

if name == ' main ':

db = DB ()

print (db . get\_one ("select \* from users where user\_name= 1 emma '' H )) 运行dhpy文件，可以看到控制台输出如下信息，说明数据库访问成功。

( ' user\_id , : 4 , \* user\_\_name 1 : 1 emma ' , ! user\_password \* : ' 123 \* , 5 user nickname \*: 'em'}

24n3D5 SB ^@spoos®Pr©t@c©Q 鬟

服务器端的响应信息其格式是固定的，有必要使用一个专门的类来按格式组建响应 信息。

之前歹我们已经将与数据协议相关的常量存入config.py文件中，内容如下：

# 数据协议才目关配置一--

REQUEST\_LOGIN = '0001' # 登录请求

REQUEST\_CHAT = ?00021 # 聊天请求

RESPONSE\_LOGIN\_RESULT = 5 1001 \* # 登录结果响应

RESPONSE\_CHAT = 1 1002 \* # 聊天响应

DELIMITER = ? | 9 #自定义协议数据分隔符

接下来,在response\_protocol.py文件中实现ResponseProtocol类的功能,代码如下:

from config import \*

class ResponseProtocol(object):

n " H响应拼接° ° ° @ staticmethod.

def response\_login\_result(result, nickname, username):

n非*u*

拼■接登录响应，数据格式为：；»响应协议编号I登录结果I登录用户昵称I登录用户用户名"

:param result:登录结果,0或者19 0表示登录失败,1表示登录成功

:param nickname:登录用户昵称,如登录失败,则该值为空字符串

:param username: 登录用户名

:return :登录结果响应格式字符串

n n n

return DELIMITER, join ([RESPONSE\_LOGIN\_J^ESULT, result? nickname? username])

@ staticmethod.

def response\_chat(nickname, messages):

n h n

拼接聊天响应，数据格式为:»响应协议编号丨聊天发送者昵称I聊天信息"

:param nickname:聊天内容发送者的昵称

:param messages :聊天内容

:return:聊天响应协议格式字符串

n n n

return DELIMITER.join([RESPONSE\_CHAT, nickname, messages])

Server类包含了服务器端程序的整个运行流程，包括管理客户端、监听客户端请求信息、 响应客户端•请求等功能。服务器端在第一次接收到客户端的信息时，会为该客户端单独开启一 个线程，专门用于客户端的交互。

我们首先在serverpy文件中添加Server类，并实现启动服务器的功能。代码如下：

from threading import Thread

from server\_socket import ServerSocket

from socket\_wrapper import Socketwrapper

from response\_protocol import \*

from clb import DB

from config import \*

class Server(object):

° °服务器"° "

*def* init (self):

#初始化套接字

self.server\_socket = ServerSocket()

def startup(self):

E5启动服务器"° "

while True:

#等待客户端连接

sock, addr = self.server socket.accept()

#给客户端sock增加额外功能

client\_sock = Socketwrapper(sock)

#启动亥程处理该用户请求

Thread(target=lambda:self.request\_handle(client\_sock)).start()

上述代码中，在\_init\_方法中创建服务器套接字并开始监听。当启动服务器程序时会调 用startup方法，在该方法中使用while循环来获得客户端连接，如果有客户端连接到服务器端， 服务器端会获得一个客户套接字来标识与该客户端的连接。然后服务器端开启新的线程来处理 该客户端的连接。新线程执行的方法为Server类的request handle方法,该方法接收客户套接 字作为参数。request\_handle方法是服务器端处理请求的核心方法。

这里需要注意一点，服务器端在获得与标识客户端连接的套接字之后，使用了 SocketWrapper类对其进行了包装(client\_sock = SocketWrapper(sock)),这是由于使用原始的 套接字在发送和接收数据的时候，需要encode编码和decode解码，我们就在SocketWrapper类 中封装了编解码的操作。

24.3.7 褻嬲Server襲——遷霧姻1 .

request\_handle方法用于处理客户端的请求，该方法的设计思路分为如下几个步骤：

1. 接收客户端请求。
2. 解析客户端请求。
3. 判断请求类型。
4. 调用请求对应处理方法。

结合该设计思路的实现代码如下：

def request\_\_handle (self, client\_sock):

""n响应虹理方法° ° °

while True:

#「读取客户端数据

request\_text = client\_sock.recv\_data()

if not request\_text:

print ( v客户端下线! v )

self . remove\_offline\_user (client sock)

break

* 2 .解析请求数据

request\_data = self o parse\_request \_text(request\_text)

* 3 .获取响应处理方法

handle\_function = self o request\_handle\_functions [request\_data [ \* recjuest id\* ]]

* 4。执行响应处理方法 if handle\_function:

handle\_function(client\_sock? request\_data)

在上述代码中，第(1)步接收客户端的请求，如果接收数据为空，表示客户端关闭连接， 我们就从在线客户端中移除该客户端信息。移除某个客户端的方法如下：

def remove\_offline\_user (self, client\_sock):

"n °移除离线用户\*接° ° "

username = None

第24章多人聊天室603 for uname, csock in self.clients.items():

if csock[\* sock'j .sock == client\_sock.sock:

username = uname

#删除用户信息 del self.clients[username]

第(2)步解析该客户端请求，解析的目的就是从接收到的数据中提取岀请求编码是什么， 请求的数据是什么。然后根据请求类型不同将数据存储成不同的格式。

第(3)、(4)步根据提取出的请求编码，来获得要处理的方法。这里需要说明一下，我 们会提前存储不同的请求协议编码对应的处理方法。如果请求编码对应的方法不存在，则会忽 略该请求。请求编码和对应处理方法对应关系以字典的形式存储在Server类的request\_handle\_ functions属性中(该属性的定义将在后续介绍)。

请求信息的解析由parse\_request\_text方法处理，该方法实现思路比较简单，代码实现如下：

@staticmethod

def parse\_request\_text(request\_text):

”5解吊请求信息5町

request\_text\_list = request\_text.split(DELIMITER)

#保存请采数据— —

request\_data = diet() request\_\_data[? request\_id!] = request\_text\_list[0] if request\_text\_list[0] =- REQUEST\_LOGIN:

request\_data[?username'] = request\_text\_list[1] request\_data [ ! password \* ] = request\_text\_\_list [2] if request\_text\_list[0] == REQUEST\_CHAT:

request\_data[1 username 1] = request\_text\_list[1] request\_data[1 messages \*] = request\_text\_list[2] return request\_data

请求信息的格式是由常量DELIMITER所定义的字符进行分隔，所以我们根据 DELIMITER常量来拆分请求信息中的各个数据，然后按照请求类型的不同，把这些请求数据 组装成不同的字典。

要注意的是，由于parse\_\_request\_text方法体内部没有使用类属性或者对象属性，所以把它 定义为一个静态方法，可以减少参数传递。

24.3^8 SB S®TOr饗——薦豪褲谟筍tfcH有灌髓瓣飕箋裹

之前介绍过不同的请求类型会对应不同的处理方法,该对应关系是通过字典属性request handle\_fiinctions来存储的。为此,我们在Server类的 init 方法中添加该属性,并将请求类 型和对应的处理方法保存在该属性中。代码如下：

class Server(object):

° ""服务器° ° °

def init (self):

#初始化套接字

self . server\_\_socket = ServerSocket ()

#保存客户端基接套接字 self.clients = diet()

#请求处理方法

self.request\_handle\_functions = diet()

#注册请求处理务法 ~

self.register(REQUEST\_LOGIN, lambda sf 9 data : self . request\_\_login\_handle (sf, data)) self.register(REQUEST\_CHAT,

lambda sf, data: self.request\_chat\_handle(sf, data))

def register(self, request\_id, handle\_function): n n °注册请求处理方法° ° "

self . request\_handle\_\_functions [request\_id] = handle\_\_function

在request\_handle\_functions字典中,请求信息的编号作为key,对应的处理方法作为 valueo如果想要找到请求类型对应的处理方法，只需要根据key值来查找即可。

需要注意的是，请求处理方法要接收两个参数sf和data。其中，时表示客户端连接的套接 字,data表示请求的数据。

从代码中可以看出,往request\_handle\_functions字典中添加了两个请求处理方法:登录请 求的处理方法(request\_login\_handle )和聊天请求的处理方法(request\_chat\_handle ),接下来 我们就来探讨这两个方法的实现思路。

牝登录请求的处理方法

登录请求的处理方法是request\_login\_handle 9该方法的实现思路大体如下：

(1 )获得登录的用户名与密码。

1. 验证数据库中是否存在对应的用户。
2. 如果用户验证成功，则在服务器端保存该用户信息；否则不保存。
3. 将登录结果返回给客户端。

依据该方法的实现思路，其实现代码如下：

def request\_\_login\_handle(self, client\_sock, requet\_data): n ° °处理用户登录° "

* 1.获得登录用户名和密码

username = requet\_data[!username!] password = requet\_\_data [ \* password , ]

* 2.验证用户是否合弟

ret, nickname? username = self.check\_user\_login(username, password)

* 3.如果登录成功，则保存用户连接套接字 - —

if ret == \* 1 \*:

self.clients[username] = (1 sock!: client\_sock, \* nickname': nickname}

* 4。组装响应结果

response\_text = ResponseProtocol.response\_login\_\_result(ret? nickname, username)

#发送响应结果 client\_sock. send\_data (response\_\_text)

上述代码的第1步是获得用户名与密码。这里直接从解析好的请求数据中获取对应的用户 名与密码即可。

第2步，验证该用户信息在数据库是否存在，即用户是否合法。这里调用了 check\_user\_ login方法。该方法实现如下：

def check\_user\_login(self, username, password):

皿，，m用户名和密码验证„ „ »

#查询SQL

sql = "select \* from users where user\_name=s" + username + "? °

#创建数据库连接对象

db\_conn = DB()

results = db\_conn.get\_one(sql)

#未查询到数据 —

if not results:

return n 0 n, " K, username

#用户名和密码不相等 if results [ 1 user\_password.1 ] ! = password: return "0", "n, username

return "1", results[nuser\_nickname"], username

check\_userjogin方法中调用了 DB类中定义的get\_\_one方法来查询用户信息，并与用户输 入的信息进行校验。如果校验成功则返回登录状态为”尸、用户昵称、用户名；如果查询或校 验失败则返回登录状态“0L并且昵称为空字符串。

第3步，如果登录成功，贝0在服务器端记录该客户端登录相关信息，信息记录在类 的clients字典中，Iwy为用户名，value ＞包含了客户套接字和昵称信息的字典。

第4步就是组装和发送响应结果。

2.聊天请求

聊天请求的处理方法是request\_chat\_handle,该方法的设计思路大体如下：

(1 )获得聊天信息发送者的用户名、聊天信息、以及昵称。注意昵称是在服务器端存储的。 (2)将聊天信息转发给其他在线客户端。

该方法的实现代码如下：

def request\_chat\_handle(self, client\_sock, request\_data): ° n n处理用户聊天n "11

* 1.获得当前用户名、发送信息、昵称 username = request\_data[\* username ?] messages = request\_data[Messages v] nickname = self.clients[username]['nickname?] #创建聊天响应信息

response\_text = Responseprotocol.response\_chat(nickname, messages)

* 2。将有息发送到每一个登录客户端 ~

for' uname, csock in self.clients.items ():

#不给自己再发信息

if uname == username: continue

#给其他用户发送消息 csock[? sock"] .send\_data(response\_text)

上述代码中的第1步是获得用户名、聊天内容、昵称，并拼接符合聊天响应格式的字符串

数据。

第2步是转发聊天数据给其他所有在线客户端。由于clients中存储了所有登录用户的连接 套接字，所以只需要遍历获取其他所有用户的连接套接字，发送数据即可。

在server.py文件的末端添加如下代码,负责创建和启动服务器端程序对象,以作为程序运 行的入口。

if name == \* main ?: server = Server() server。startup()

此时运行server.py文件,就可以启动服务器端程序T。

至此，服务器端程序已经全部实现，接下来就是客户端程序的设计和开发。

容戶端程序设计

客户端程序的主要功能包括用户登录、聊天信息发送、聊天信息接收等，这些功能的操作 都是基于图形界面的。

明确了客户端的主要功能之后，我们首先针对客户端程序的业务流程进行设讨％如图24J2 所示。

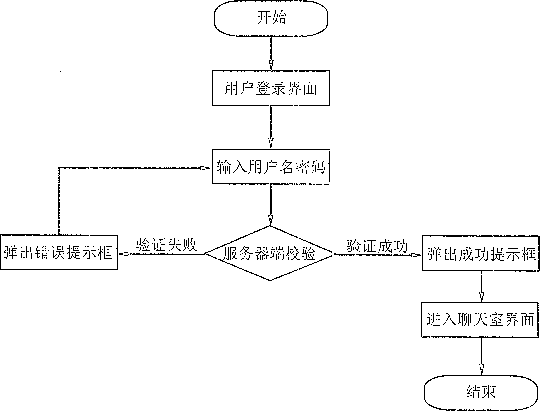


图24J2客户端程序流程图

客户端程序的业务流程主要包括：

1. 用户在登录界面输入用户名和密码。
2. 客户端向服务器端发送登录请求。

(3 )服务器端对用户信息进行校验并返回验证结果。

(4 )客户端接收验证结果。如果验证失败，则弹岀对话框提示用户重新输人 如果验证成功, 则进入聊天界面开始聊天，直到程序结束。

用户登录和在聊天界面发送聊天信息都需要和服务器端通信，所以客户端也和服务器端约 定了发送何种格式的数据给服务器，如下所示。

®登录请求格式为：H0001|usemam司passwonT,其中"0001" 登录请求编码，username为登 录用户名9 password为登录密码。

®聊天请求格式:,,0002|usemame|messagesn,其中”0002"为聊天请求编码,username为聊 天信息发送者的用户名，messages为其发送的聊天信息。

之前我们在服务器端的配置文件中添加了常量来表示客户端和服务器端之间的数据交换用

到的常量，同样地，在客户端的config,py文件中也要对这些常量进行定义，一般来说，其内容 与服务器端是一模一样的，如下所示。

# 数据协议相关配置

REQUEST\_LOGIN = ? 00015 REQUEST\_CHAT = 1 0002 ,

#登录请求

#聊天请求

#登录结果响应

#聊天响应

#自定义协议数据分隔符

RESPONSE\_\_LOGIN\_RESULT = ' 1001 ! RESPONSE\_\_CHAT = \* 1002 \*

DELIMITER = 5 |?

2盛痢警 襲靈滇聽谩讲

为了方便程序的管理和维护，我们将客户器端程序的相对独立功能使用单独的类和模块来 实现。具体设计的类如表24.4所示。

表24-4类的设计

|  |  |
| --- | --- |
| 类名称 | 含 义 |
| Client | 客户端程序类，负责客户端的整个流程 |
| ClientSocket | 客户端套接字类，隹于创建客户端端连接的套接字 |
| RequestProtocol | 请求拼接类，用于按照协议规定的格式拼接客户端请求信息 |
| WindowLogin | 登录窗口类，负责登录窗口的显示和功能 |
| Windov/Chat | 聊天窗口类，负责聊天窗口的显示和功能 |

同时，在项目中创建若干个模块，每个类单独放在一个模块里，另外还需要一个单独的配 置文件。设计后的模块如表2&5所示。

表24-5模*块*设计

|  |  |
| --- | --- |
| 模块名称 | 含 义 |
| client.py | 用于包含Client类 |
| client\_socket.py | 用于包含ClientSocket类 |

续表

|  |  |
| --- | --- |
| 模块名称 | 含 义 |
| requestjprotocol.py | 用于包含RequestProtocol类 |
| window\_login.py | 用于包含WindowLogin类 |
| window\_chat.py | 用于包含WindowChat类 |
| config.py | 配置文件，用于保存程序的所有设置值 |

I 客户端程序实嬲

对客户端程序设计完成之后，就可以基于这些设计进行开发了。我们知道，客户端的主要 操作都是基于图形界面，这里使用Pythoti自带的Tkinter实现。接下来就为大家介绍客户端程 序的实现步骤。

24SJ创還囑圍測支鮮鏡酒

首先使用PyCharm新建一个项目，取名为Cfeif。然后按照客户端程序中模块的设计，在 该项目下分另g 仓!］建 client.py、client socket.py config.py^ reguestjprotocoLpy window\_chat.py 和window login.py文件。创建完成后的文件结构如图24-13所示。

*、* rfight ' "- .

*fjfv.* €Hent.py

■,…, cBent\_socketpv fCD-nfig,py req u est\_p roto c oLpy U wi nd-ow\_chat. py

图2443客户端程序的文件结构

24d5d2 .

登录窗口提供用户名和密码输入，重置及登录功能。该窗口包含的控件很少，只有两个标 签(Label).两个文本框(Entry)和两个按钮(Button)。

登录窗口设定宽度为255像素，高度为95像素，并且设置窗口大小不可修改，窗口起始 位置坐标为屏幕中央位置。

为此，我们在window\_login„py文件中创建WindowLogin类，让它继承自Tk类。然后在 该类中添加windowjnit方法，用于创建登录窗口。代码如下:

from tkinter import Tk, Label, Entry, Button, Frame

from tkinter import LEFT, END

class WindowLogin(Tk):

° ° °登录窗口 n ° "

def window\_init(self):

° e窗口属性初始化» « "

#设置窗体标题

self .title (° 登录窗口 ")

#设置窗体大小不能修改

self»resizable(False, False)

#窗口宽高•

window\_\_width = 255

window\_height = *95* #屏幕宽高

screen\_\_width = self. winfo\_screenwidth ()

screen\_height = self.winfo\_screenheight()

#屏幕車心位置坐标 一

pos\_x = (screen\_width - window\_width) / 2

pos\_\_y = (screen\_height - window\_height) / 2

#疫置屏幕宽、高、&位置、y位置 ~

self. geometry (' %dx%d+%d+%d5 % (window\_width, window\_height, pos\_\_x? pos\_y))

登录窗口内包含的控件布局如图2444所示。

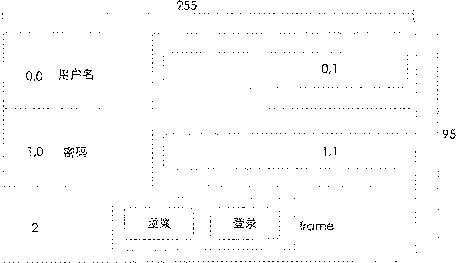


图24-14 登录窗口界面布局

从布局24-14图可知，登录窗口整体采用grid表格布局，其中用户名标签放置在第1行第 1列，对应的用户名输入框放置在第1!行第2列。密码标签放置在第2行第1列，密码输入框 放置在第2行2列。对于重置按钮和登录按钮，由于使用了穿id表格布局，所以将这两个按钮 水平布局于一个Frame中，再将该Frame放置在登录窗口的第3行，占据2列，这样这两个按 钮就居于第3行的正中央了。

接下来，在WindowLogin类中添加add\_widgds方法，实现往窗口中添加控件的功能。对 应代码如下：

def a.dd.\_widgets (self):

创建所需控件"° "

#用户名提示标签

username\_label = Label (self)

username\_\_label [ ' text' ] = v 用户名 :1

username\_label.grid(row=0 ? column=0, padx=10, pady=5)

#用户名输入文本框

username\_entry = Entry(self, name=1username\_entry\*)

username\_entry[!width \*] = 25

username\_entry.grid(row=0， column=l)

#密码提奈标签

password\_label = Label(self) password\_\_label [ 1 textv ] = *1 密* 码:1 password\_label.grid(row=l, column=0)

#密码输人文本框

password\_entry = Entry (self, name= ' password\_\_entry \* ) password\_\_entry [ f show! ] = \* \* ?

password\_entry[1 width \*] *= 25* password\_entry.grid(row=l, column=l) #创建Frame

button\_frame = Frame(self, name= \*button\_frame!)

* .重亶”按钮 —

reset\_button = Button(button\_frame, name= \* reset\_button1) reset\_button [ 1 text \* ] = 1 重置 , reset\_button.pack(side=LEFT, padx=20)

* “疯”按钮

login\_button = Button(button\_frame? name=1login\_button') login\_\_button [ ? text? ] = 5 登录 1 login\_button spack(side=LEFT)

button\_frame.grid(row=2, columnspan=2, pady=5)

然后在初始化方法中调用创建窗口和添加控件的方法，代码如下所示。

def init (self) : #调用父类初始化方法

super(WindowLogin, self). init () #设置窗体属性 — —

self.window\_init()

#创建控件

self. add\_\_widgets ()

这样，登录窗口的界面就能显示了，但是还缺少对用户操作的处理，比如单击按钮，关闭 窗口等。为此，我们为登录窗口类封装了一些操作方法，可在需要的时候对窗口进行指定的操作。 这些操作方法包括：

'•获得输入的用户名。

•获得输入的密码。

-清除输入的用户名。

。清除输入的密码。

-给登录按钮绑定处理方法。

•给重置按钮绑定处理方法。

•给窗口关闭按钮绑定处理方法。

具体代码实现如下：

def get\_username(self):

° °获得用户名ee

return self。children[!username\_entry 5] .get () def clear\_username(self):

n n "清裕输入的用户名"° °

self o children[1username\_entry5] s delete(0 3 END) def clear\_password(self):

??nn清除输入的密码5叫

self. children [ \*password\_\_entry1 ] . delete (0, END)

def get\_\_password (self):

° " n获得密码n n °

return self.children[1password\_entry5] .get()

def on\_\_login\_button\_click (self, command):

"""登录扌豺丑绑定点备事件处理方法心E

self. children [' button\_frante? ] . children [1 loginJoutton' ] [? command \* ] = command def on\_\_reset\_button\_click (self, command):

“EV重置扌宓丑点击事律处理方法E5

self。children [' button\_frame! ]。children [' reset Joutton1 ] [ \* conimand ? ] = command def on\_window\_closed(self, command):

° n °窗口关甬事件处理方法° ° °

#设置窗口关闭事件

self.protocol('WM\_DELETE\_WINDOW?, command)

在上述代码中，

® get()方法用于获得 Entry 文本框输入的内容(例如 sel£children[?usemame\_entryl].get())。

® delete(O,END)删除从0位置到最后所有的输入内容(例如self.children[!usemame\_entry,]. delete(0?END)) o

• self.protocol(?WM\_DELETE\_WINDOW\ command)用于给窗口设置当右上角“关闭”按 钮单击时要执行的方法。

要注意的是，在封装操作方法时获取控件使用的是类似self.children[,password\_\_entry,]的 方法，这是由于文本框、按钮等控件都是作为Tk的子对象，被存储在了 Tk的字典类型的 children.属性中。

24.5.3 '

聊天窗口主要用于发送聊天信息，以及展示多人聊天的信息。

我们将窗口设定宽度为795像素，高度为505像素，并且设置窗口大小不可修改。为此， 在window\_chat.py文件中添加WindowChat类，继承自Toplevel类。在它的初始化方法里创建 聊天窗口，代码如下所示。

from tkinter.scrolledtext import ScrolledText

from tkinter import Toplevel, Text, Button

from tkinter import END, UNITS

from time import strftime, localtime, time

class WindowChat(Toplevel):

def init (self):

F调用交柔方法初始化窗体

super(WindowChat, self). init ()

#设置窗体宽高 — —

self.geometry(!%dx%d! *% (7 95,* 505))

#设置聊天室宽高不可修改

selfresizable(False? False)

窗口内包含一个ScrolledText控件用于显示聊天信息? 一个Text控件用于输入聊天信息， 一个Button按钮用于发送聊天信息。这些控件在窗口中的布局设计如图2445所示。

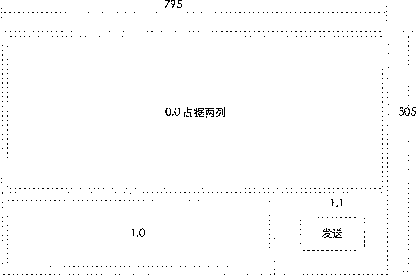


图2445聊天窗口界面布局设计

接下来，在WindowChat类中添加add\_widgets方法须用于往界面上添加控件。代码如下:

def add\_widgets(self):

#疏天内容显示文本框

chat\_textarea = ScrolledText(self) chat\_textarea[1 width 1] = 110

chat\_textarea[5 height'] = 30 chat\_textarea.grid. (row=05 column=0, columnspan=2) #添加2个标签

chat\_textarea.tag\_config(? green \*, foreground= \* #008B00') chat\_textarea. tag\_config (1 system \*, f oreground=1 red \*) self. children [' chat\_textarea. \* ] = chat\_textarea

#聊天文本输入框

chat\_inputarea = Text(self, name=? chat\_inputarea') chat\_inputarea[1 width'] = 100

chat\_inputarea[\* height?] = 7

chat\_inputarea . grid (row=l, column=:0, pady=10) #发遑按钮

send\_button = Button(self, name=\*send\_button1) send\_button [ 5 text \* ]='发送 f

send\_button[!width \*] = 5

send\_button[\* height \*] = 2 send\_button.grid(row=l, column=l)

然后在—init \_方法的末尾调用add\_widgets方法,代码如下所示。

#设置窗体

self。add\_widgets()

这样就可以成功地显示聊天窗口了，下一步就是为该窗口的操作提供方法支持，这些方法 包括：

。设置窗口标题。

°清除聊天内容。

•获得输入聊天内容。

•追加聊天内容。

。窗口关闭事件处理方法。

• “发送”按钮单击事件处理方法。 这些方法的具体实现代码如下:

def set\_title(self, title): f设置窗口标题"° "

self. title (!欢迎 %s 进入聊天室! 1 % title)

def clear\_inputs(self): """.清除聊天内容"° ° #清空输入框内容

self.children[? chat\_inputarea'] .delete(0.0, END) def get\_inputs(self):

"""获得输入内容° ° n

return self.children['chat\_inputarea1].get(0.0, END) append\_message(self, sender, message):

def

def

def

心顷追加聊天内容° ° "

#获得当前时间

send\_time = strftime(?%Y-%m-%d %H:%M:%S \*, localtime(time())) sender\_info = 1%s: %s\n1 % (sender, send\_time)

self. children [ f chat\_textarea f o insert (END, sender\_inf o, \* green 1 ). selfochildren[\* chat\_textarea \*] .insert(END, 1 1 + message + J\n \*) #滚动条移动3个单位

self.children['chat\_textarea'].yview\_scroll(3, UNITS) on\_window\_closed(self? command):

° °窗口关闭事件处理方法° °,v

self .protocol ( \* WM\_\_DELETE\_\_WINDOW\* , command) on\_send\_button\_click(self, command):

n ""发送按钮点击事件处理方法心顷

self.children[\*send\_button'][?command®] = command

24L鼠盛 靈碘襲

客户端发送的请求信息其格式是固定的，我们有必要使用一个专门的类来按格式组建请求

信息。

首先，将与数据协议相关的配置文件存入config.py文件中，内容如下:

# 数据协议相关配置

REQUEST\_LOGIN = ? 0001 1 # 登录请求

REQUEST\_CHAT = ' 0002 ! # 聊天请求

RESPONSE LOGIN RESULT = \* 1001 1 # 登录结果响应

RESPONSE\_CHAT = '1002' # 聊天响应

DELIMITER = f | ? #自定义协议数据分隔符

然后在request\_protocol.py文件中创建RequestProtocol类,负责按照格式组建登录请求信 息和聊天请求信息，代码如下：

from config import \* class RequestProtocol(object):

f » 1请求协议组装J 5 1

@staticmethod

def request\_login(username? password):

拼接登录请求，数据格式:"请求协议编号|登录用户名|登录密码"

:param username : 登录用户名

:pajram password: 登录密码

:return :登录请求格式字符串

H H H

return DELIMITER.join([REQUEST\_LOGIN, username, password])

@staticmethod

def request\_chat(username, message):

h n n

拼接聊天请，数据格式:”请求协议编号I聊天内容发送者用户名I聊天内容“

:param username :聊天信息发送者的用户名

:param messages :聊天内容

:return :聊天请求格式字符串

H H H

return DELIMITER.join([REQUEST\_CHAT, username, message])

24n5a5 S® 襲

Clientsocket类负责客户端连接服务器端以及数据传递时的编码和解码。要连接服务器， 首先要知道服务器端的IP和端口。为此，在配置文件config.py中添加如下内容。

# 服务器相关配置-■—-

SERVER\_\_IP = ? 127.0.0.1 5 # 服务器工P 地址

SERVER\_PORT = 8090 #月艮务器端口号

然后在client\_socket.py文件中添加ClientSocket类的代码实现，如下所示。

from socket import socket, AF\_INET, SOCK\_STREAM from config import SERVER\_IP, SERVER\_PORT class Clientsocket(socket):

def init (self): ° ° "初始化套接字"n n super (Clientsocket, self) . init(AF\_\_INET, SOCK\_STREAM) def connect\_server(self):

"n n连接鼠务器n ° n

self.connect((SERVER\_IP, SERVER\_PORT))

#设置当前套接字为非阻启莫式

self.setblocking(0)

def recv\_data(self):

"顷获得服务器返回数据“顷

return self.recv(512) »decode(°utf-8n)

def send\_data(self, messages): j”向服务器发送数据"j

self.send(messages,encode("utf-8 n))

需要注意的是，这里将客户端的套接字设置为非阻塞模式。

24q5q® 雲勰割您展鑿——理繇匯槊簷31

Client类负责整个客户端程序的运行流程，该类定义于dient.py模块中。首先进行初始化 操作，这些初始化操作主要包括：

1. 创建登录窗口对象和聊天窗口对象，但设置聊天窗口为隐藏不可见状态。

(2 )创建用于连接服务器的套接字对象conn0

(3 )创建response\_handle\_functions属性用于存储响应协议编号与对应处理方法的对应 关系。

首先在client.py文件中添加Client类要用到的引用库，如下所示。

from tkinter»messagebox import showinfo

from threading import Thread

from time import sleep

import sys

from request\_protocol import RequestProtocol

from window\_login import WindowLogin

from window\_chat import WindowChat

from client\_socket import Clientsocket

from config import \*

然后创建Client类，实现它的初始化方法，代码如下：

class Client(object):

"° "客户端n ° n

def init (self):

#创建登录窗口

self.window = WindowLogin()

#创建聊天室窗口并隐藏

self.room = WindowChat()

self.room.withdraw()

#服务器连接对象

self <. conn = Clientsocket ()

#存储响应处理方法

self.response\_handle\_functions = diet()

#处理服务器响应戋程是否｝E在运行

self o is\_running = True

#当前登录用*户*,未登录之前为None

self o username = None

客户端对象初始化完毕之后，我们创建startup方法用于启动客户端，该方法负责连接服务 器，并启动单独的线程用来处理服务器返回的响应。具体实现如下：

def startup(self):

ee启动客户端° " °

#连接服务器

self.conn.connect\_server()

#创建线程处理服务器道回的响应

Thread(target=lambda:self。response\_handle()) .start ()

#窗口主循环

se 1 f o window .,mainloop ()

从代码可知,启动新线程调用的是response handle方法,下一步就来实现这个方法。

24.5.7 COfcot BSWBB

response\_handle方法是Client类的响应处理方法,主要用来处理服务器返回的响应信息。 例如，客户端返回聊天内容了，则该方法调用相应方法来将聊天内容显示到聊天窗口上。

response handle方法的设计思路如下:

(1 )每隔OJ s检查一下服务器是否返回数据。

(2 )拿到服务器返回的响应数据之后，调用pa.rsing\_response\_text方法解析响应数据。

(3 )到字典response handle functions中查找该响应类型所对应的处理方法,如果不存在, 则忽略该响应信息；如果存在，则将解析后的响应数据作为参数来调用该处理方法。

response handle方法的实现代码如下:

def response\_handle(self):

while self.is\_running: response\_text = None try:

#接受服务器端返回的数据

response\_text = self . conn . recv\_\_data () except BlockinglOError:

sleep (0.1)

continue

#响应编号和响应内容 response\_data = self.parsing\_response\_text(response\_text) #获取对应处理方法 handle\_function =

self o response\_handle\_functions[response\_data[\* response\_id1 ]] if handle\_\_function:

handle\_function(response\_data)

parsing response text方法是响应信息解析方法,在response\_handle方法中调用了 parsing\_ response\_text方法来解析响应信息。那么parsing\_response\_text方法是如何实现的呢?

其实parsing\_response\_text方法的实现思路比较简单，只需要按照预先定义的响应协议格 式来拆分数据，然后将数据存储为字典对象返回。实现代码如下？

@staticmethod

def parsing\_response\_text(response\_text):

° "解析服务器返回品字符串n n n

#响应编号 response\_text\_list = response\_text.. split (DELIMITER)

#保存解折后的竅据 —

response\_data = diet()

response\_data[°response\_id"] = response\_text\_list[0]

#如果响应的是登录结果

if response\_\_text\_list[0] == RESPONSE\_LOGIN\_RESULT: response\_data["result"] = response\_text\_list[1] response\_data["nickname"] = response\_text\_list[2] response\_data["username"] = response\_text\_list[3]

#如果响应的是聊天内容

if response\_text\_list[0] == RESPONSE\_CHAT: response\_data[?nickname 1] = response\_text\_list[1] response\_data. ['messages 1] = response\_text\_list[2] return response\_data

在解析得到响应的类型后，要调用该响应对应的处理方法进行处理。为此，首先要在初始

化方法中将这两个处理方法进行注册，代码如下：

#注册响应处理方法

self.register(RESPONSE\_LOGIN\_RESULT, lambda

data:self.response\_login\_handle(data))

self.register(RESPONSE\_CHAT, lambda

data:self.response\_chat\_handle(data))

register方法负责注册具体的响应处理方法，该方法代码如下:

def register (self, response\_\_id, handle\_function):

"5注册响应处理方法“5

self.response\_handle\_functions[response\_id] = handle\_ function

这两个响应处理方法中，response\_login\_handle方法用于处理登录响应，其实现思路大体 如下：

1. 判断登录响应状态码，如果状态码为” 0”，则表示登录失败，此时需要弹窗提 示用户。
2. 如果状态码为” 1”表示登录成功，此时应隐藏登录窗口，显示聊天窗口，并将聊天 窗口的标题设置为当前用户的昵称。

response\_login\_handle方法的实现代码如下:

def response\_login\_handle(self? response\_data):

»»'»登录响应处理° ° "

result = response\_data["result"] nickname = response\_data[nnickname"] if result == °0":

showinfo (°提示\*用户名或密码错误° )

return

#登录成功处理

self.username = response\_data["username"]

showinfo ("提示n , °登录成新！")

#显示聊天窗口

self.room.set\_title(nickname)

self » roomo update()

self.room.deiconify()

#隐藏登录窗口

self.window.withdraw()

response\_chat\_handle方法主要用于聊天响应的处理,其实现思路如下:

(1)获得聊天响应的数据。

(2 )将聊天内容显示到聊天窗口。

该方法实现代码如下。

def response\_chat\_handle(self, response\_data):

n ° "聊天响应处理° " n

nickname = response\_data['nickname!]

messages = response\_data['messages ?]

#显示聊天内容 ~

self - room.append\_message(nickname? messages)

2凰忐思 靈嬲 csfeoft a—

登录窗口、聊天窗口都有一些按钮事件没有实现，我们在这里给这些按钮绑定对应的处理 方法。例如，当登录窗口的“重置”按钮被单击时，使用clearJnputs方法处理；“登录”按钮 被单击时，调用send\_login\_data方法处理。当窗口右上角的"关闭”按钮被单击时，我们调用 exit方法来处理。

首先在Client类的初始化方法中添加代码,将窗口的事件与对应的处理方法进行绑定,如 下所示。

# “登录”按钮绑定单击的处理方法

self.window.on\_reset\_button\_click(lambda: self.clear\_inputs()) self o window•on\_login\_button\_click(lambda: self。send\_login\_data()) self.window.on\_window\_closed(lambda: self.exit())

#聊天室窗口发送按钮绑定处理方法

self . room. on\_send\_button\_click (lambda : se 1 f:» send\_chat\_data ())

self o room.on\_window\_closed(lambda:self。exit())

然后在Client类中依次实现这些事件处理方法。其中，登录窗口的事件处理方法如下所示。

def

exit(self): n n n退出程序n "" self»is\_running = False self.conn.close()

sys.exit(0)

def

clear\_inputs(self): si清空输入框内容叫5

self.window.clear\_username() self.window.clear\_password() send\_login\_data(self): n " w麦送登•录竅据到服务器"W "

def

#获得用户输入内容

username = self.window.get\_username() passowrd = self.window.get\_password() #构造登录请求 ―

request\_text = Requestprotocol.request\_login(username, passowrd) print ( 5 发送的聊天内容:5 , request\_text)

#发送请求 -

self.conn。send\_data(request\_text) 聊天窗口的事件处理方法如下所示。

def send\_\_chat\_data (self):

5"麦送聊灵数据到服务器心顷

#获得聊天内容 chat\_contents = self.room.get\_inputs()

#清空输入内容

self.room.clear\_inputs()

#显示聊天内容 self . room。append\_\_message ("我 七 chat\_contents)

#生成聊天协议内容 request\_text = Requestprotocol. request\_chat (self ausername, chat contents)

#向服务器端发送聊天数据

self.conn.send\_data(request\_text)

在client.py文件的末端添加如下代码，负责创建和启动客户端程序对象，以作为客户端程 序运行的入口。

if name == \* main , :

client = Client() client.startup ()

此时运行client.py文件，就可以启动客户端程序了。

2<6项同体急

至此，多人聊天室项目的功能已全部实现。我们先梳理了服务器端和客户端的工作流程， 然后分别对服务器端程序和客户端程序进行了设计和实现。通过多人聊天室项目的开发多达到 了如下两个目的：

(1)让大家更好地体会面向对象编程的设计思想。整个项目都贯彻了面向对象的设计方法， 将所有的元素都封装在类中，程序的功能由一个个单独的类负责提供，从而使得程序的责任更 明确，项目结构更清晰。

(2 )对前面章节学习的Python GUI库(Tkinter)、多任务编程、网络编程及数据库编程 等相关知识进行应用。其中客户端程序的登录窗口和聊天窗口都采用了 Tkinte『实现，服务器 端对每个客户端都新开一个线程进行处理是对多任务编程的应用，而服务器端和客户端的连接 使用了 Socket网络编程技术，服务器端还使用了 MySQL数据库来存储用户信息。综上所述， 本项目是一个融合了多种技术的综合应用型项目，大家通过这个项目的学习，能够加深对前面 章节所学知识的体会，达到融会贯通的目的。