Python 实现文字聊天室

在本课程中，我们将实现一个简单的图形界面聊天系统。我们可以通过图形客户端登录聊天室，并与其他成员进行聊天。

**文字聊天室**

## 一、实验介绍

## 二、原理解析

## 三、服务器类

### 3.1 会话类

### 3.2 协议命令解释器

### 3.3 房间

## 四、登陆窗口

### 4.1 聊天窗口

## 五、执行

## 六、项目扩展

## 七、小结

一、实验介绍

**1.1 实验内容**

本实验使用 wxPython 、asynchat、\_thread 等模块开发一个图形界面的聊天室程序。

**1.2 知识点**

* asyncore 、asynchat模块使用
* wxPython 图形开发

**1.3 实验环境**

* python3.5

对于实验环境中使用的 wxPython，在环境中我们已经给出了对应的安装包。

$ wget http://labfile.oss.aliyuncs.com/courses/970/wxPython.whl

$ mv wxPython.whl wxPython-4.0.0a2.dev3038+953a2e5-cp35-cp35m-linux\_x86\_64.whl

$ sudo pip3 install wxPython-4.0.0a2.dev3038+953a2e5-cp35-cp35m-linux\_x86\_64.whl

二、原理解析

在本实验中，我们将实现一个简单的图形界面聊天系统。我们可以通过图形客户端登录聊天室，并与其他成员进行聊天。

由于 Python 是一门带 GIL 的语言，所以在 Python 中使用多线程处理IO操作过多的任务并不是很好的选择。同时聊天服务器将同多个 socket 进行通信，所以我们可以基于 asyncore 模块实现聊天服务器。aysncore 模块是一个异步的 socket 处理器，通过使用该模块将大大简化异步编程的难度。asynchat 模块在 asyncore 模块的基础上做了进一步封装，简化了基于文本协议的通信任务的开发难度。

既然要开发聊天程序，那必然需要设计聊天时使用的协议。为了简单起见，我们将要开发的聊天服务器只支持文本协议，通过command message的方式调用相关的操作。比如如果客户端发送以下文本，将执行相应的操作

# 登录操作

login\n

# 在聊天室中发表 hello 内容

say hello\n

# 查看聊天室在线用户

look\n

# 退出登录

logout\n

以上协议流中，login, say, look, logout 就是相关协议代码。

然后使用下面的命令在 /home/shiyanlou/Code 目录下创建我们需要的 server.py 和 client.py 文件：

$ touch ~/Code/server.py

$ touch ~/Code/client.py

三、服务器类

这里我们首先需要一个聊天服务器类，通过继承 asyncore 的 dispatcher 类来实现，我们编写 server.py 文件：

import asynchat

import asyncore

# 定义端口

PORT = 6666

# 定义结束异常类

class EndSession(Exception):

pass

class ChatServer(asyncore.dispatcher):

"""

聊天服务器

"""

def \_\_init\_\_(self, port):

asyncore.dispatcher.\_\_init\_\_(self)

# 创建socket

self.create\_socket()

# 设置 socket 为可重用

self.set\_reuse\_addr()

# 监听端口

self.bind(('', port))

self.listen(5)

self.users = {}

self.main\_room = ChatRoom(self)

def handle\_accept(self):

conn, addr = self.accept()

ChatSession(self, conn)

这里需要补充说明的是，对于 asyncore 和 asynchat 模块来讲，在 python3.6 中，使用 asyncio 模块代替，但是实验环境中我们使用的是 python 3.5 ，也由于 wxPython 对于Linux 下 CPython 的支持，所以我们依然使用 python 3.5。

3.1 会话类

有了服务器类还需要能维护每个用户的连接会话，这里继承 asynchat 的 async\_chat 类来实现，在 server.py 文件中定义，代码如下：

class ChatSession(asynchat.async\_chat):

"""

负责和客户端通信

"""

def \_\_init\_\_(self, server, sock):

asynchat.async\_chat.\_\_init\_\_(self, sock)

self.server = server

self.set\_terminator(b'\n')

self.data = []

self.name = None

self.enter(LoginRoom(server))

def enter(self, room):

# 从当前房间移除自身，然后添加到指定房间

try:

cur = self.room

except AttributeError:

pass

else:

cur.remove(self)

self.room = room

room.add(self)

def collect\_incoming\_data(self, data):

# 接收客户端的数据

self.data.append(data.decode("utf-8"))

def found\_terminator(self):

# 当客户端的一条数据结束时的处理

line = ''.join(self.data)

self.data = []

try:

self.room.handle(self, line.encode("utf-8"))

# 退出聊天室的处理

except EndSession:

self.handle\_close()

def handle\_close(self):

# 当 session 关闭时，将进入 LogoutRoom

asynchat.async\_chat.handle\_close(self)

self.enter(LogoutRoom(self.server))

3.2 协议命令解释器

在之前的分析中，我们设计了聊天服务器的协议，我们需要实现协议命令的相应方法，具体来说就是处理用户登录，退出，发消息，查询在线用户的代码。在 server.py 文件中定义，

class CommandHandler:

"""

命令处理类

"""

def unknown(self, session, cmd):

# 响应未知命令

# 通过 asynchat.async\_chat.push 方法发送消息

session.push(('Unknown command {} \n'.format(cmd)).encode("utf-8"))

def handle(self, session, line):

line = line.decode()

# 命令处理

if not line.strip():

return

parts = line.split(' ', 1)

cmd = parts[0]

try:

line = parts[1].strip()

except IndexError:

line = ''

# 通过协议代码执行相应的方法

method = getattr(self, 'do\_' + cmd, None)

try:

method(session, line)

except TypeError:

self.unknown(session, cmd)

3.3 房间

接下来就需要实现聊天室的房间了，这里我们定义了三种房间，分别是用户刚登录时的房间、聊天的房间和退出登录的房间，这三种房间都继承自 CommandHandler，在 server.py 文件中定义，代码如下：

class Room(CommandHandler):

"""

包含多个用户的环境，负责基本的命令处理和广播

"""

def \_\_init\_\_(self, server):

self.server = server

self.sessions = []

def add(self, session):

# 一个用户进入房间

self.sessions.append(session)

def remove(self, session):

# 一个用户离开房间

self.sessions.remove(session)

def broadcast(self, line):

# 向所有的用户发送指定消息

# 使用 asynchat.asyn\_chat.push 方法发送数据

for session in self.sessions:

session.push(line)

def do\_logout(self, session, line):

# 退出房间

raise EndSession

class LoginRoom(Room):

"""

处理登录用户

"""

def add(self, session):

# 用户连接成功的回应

Room.add(self, session)

# 使用 asynchat.asyn\_chat.push 方法发送数据

session.push(b'Connect Success')

def do\_login(self, session, line):

# 用户登录逻辑

name = line.strip()

# 获取用户名称

if not name:

session.push(b'UserName Empty')

# 检查是否有同名用户

elif name in self.server.users:

session.push(b'UserName Exist')

# 用户名检查成功后，进入主聊天室

else:

session.name = name

session.enter(self.server.main\_room)

class LogoutRoom(Room):

"""

处理退出用户

"""

def add(self, session):

# 从服务器中移除

try:

del self.server.users[session.name]

except KeyError:

pass

class ChatRoom(Room):

"""

聊天用的房间

"""

def add(self, session):

# 广播新用户进入

session.push(b'Login Success')

self.broadcast((session.name + ' has entered the room.\n').encode("utf-8"))

self.server.users[session.name] = session

Room.add(self, session)

def remove(self, session):

# 广播用户离开

Room.remove(self, session)

self.broadcast((session.name + ' has left the room.\n').encode("utf-8"))

def do\_say(self, session, line):

# 客户端发送消息

self.broadcast((session.name + ': ' + line + '\n').encode("utf-8"))

def do\_look(self, session, line):

# 查看在线用户

session.push(b'Online Users:\n')

for other in self.sessions:

session.push((other.name + '\n').encode("utf-8"))

if \_\_name\_\_ == '\_\_main\_\_':

s = ChatServer(PORT)

try:

print("chat server run at '0.0.0.0:{0}'".format(PORT))

asyncore.loop()

except KeyboardInterrupt:

print("chat server exit")

四、登陆窗口

完成了服务器端后，就需要实现客户端了。客户端将基于 wxPython 模块实现。 wxPython 模块是 [wxWidgets](http://wxwidgets.org/" \t "_blank) GUI 工具的 Python 绑定。所以通过 wxPython 模块我们就可以实现 GUI 编程了。同时我们的聊天协议基于文本，所以我们和服务器之间的通信将基于 telnetlib 模块实现。

登录窗口通过继承 wx.Frame 类来实现，编写 client.py 文件，代码如下：

import wx

import telnetlib

from time import sleep

import \_thread as thread

class LoginFrame(wx.Frame):

"""

登录窗口

"""

def \_\_init\_\_(self, parent, id, title, size):

# 初始化，添加控件并绑定事件

wx.Frame.\_\_init\_\_(self, parent, id, title)

self.SetSize(size)

self.Center()

self.serverAddressLabel = wx.StaticText(self, label="Server Address", pos=(10, 50), size=(120, 25))

self.userNameLabel = wx.StaticText(self, label="UserName", pos=(40, 100), size=(120, 25))

self.serverAddress = wx.TextCtrl(self, pos=(120, 47), size=(150, 25))

self.userName = wx.TextCtrl(self, pos=(120, 97), size=(150, 25))

self.loginButton = wx.Button(self, label='Login', pos=(80, 145), size=(130, 30))

# 绑定登录方法

self.loginButton.Bind(wx.EVT\_BUTTON, self.login)

self.Show()

def login(self, event):

# 登录处理

try:

serverAddress = self.serverAddress.GetLineText(0).split(':')

con.open(serverAddress[0], port=int(serverAddress[1]), timeout=10)

response = con.read\_some()

if response != b'Connect Success':

self.showDialog('Error', 'Connect Fail!', (200, 100))

return

con.write(('login ' + str(self.userName.GetLineText(0)) + '\n').encode("utf-8"))

response = con.read\_some()

if response == b'UserName Empty':

self.showDialog('Error', 'UserName Empty!', (200, 100))

elif response == b'UserName Exist':

self.showDialog('Error', 'UserName Exist!', (200, 100))

else:

self.Close()

ChatFrame(None, 2, title='ShiYanLou Chat Client', size=(500, 400))

except Exception:

self.showDialog('Error', 'Connect Fail!', (95, 20))

def showDialog(self, title, content, size):

# 显示错误信息对话框

dialog = wx.Dialog(self, title=title, size=size)

dialog.Center()

wx.StaticText(dialog, label=content)

dialog.ShowModal()

4.1 聊天窗口

聊天窗口中最主要的就是向服务器发消息并接受服务器的消息，这里通过子线程来接收消息，在 client.py文件中定义，代码如下：

class ChatFrame(wx.Frame):

"""

聊天窗口

"""

def \_\_init\_\_(self, parent, id, title, size):

# 初始化，添加控件并绑定事件

wx.Frame.\_\_init\_\_(self, parent, id, title)

self.SetSize(size)

self.Center()

self.chatFrame = wx.TextCtrl(self, pos=(5, 5), size=(490, 310), style=wx.TE\_MULTILINE | wx.TE\_READONLY)

self.message = wx.TextCtrl(self, pos=(5, 320), size=(300, 25))

self.sendButton = wx.Button(self, label="Send", pos=(310, 320), size=(58, 25))

self.usersButton = wx.Button(self, label="Users", pos=(373, 320), size=(58, 25))

self.closeButton = wx.Button(self, label="Close", pos=(436, 320), size=(58, 25))

# 发送按钮绑定发送消息方法

self.sendButton.Bind(wx.EVT\_BUTTON, self.send)

# Users按钮绑定获取在线用户数量方法

self.usersButton.Bind(wx.EVT\_BUTTON, self.lookUsers)

# 关闭按钮绑定关闭方法

self.closeButton.Bind(wx.EVT\_BUTTON, self.close)

thread.start\_new\_thread(self.receive, ())

self.Show()

def send(self, event):

# 发送消息

message = str(self.message.GetLineText(0)).strip()

if message != '':

con.write(('say ' + message + '\n').encode("utf-8"))

self.message.Clear()

def lookUsers(self, event):

# 查看当前在线用户

con.write(b'look\n')

def close(self, event):

# 关闭窗口

con.write(b'logout\n')

con.close()

self.Close()

def receive(self):

# 接受服务器的消息

while True:

sleep(0.6)

result = con.read\_very\_eager()

if result != '':

self.chatFrame.AppendText(result)

if \_\_name\_\_ == '\_\_main\_\_':

app = wx.App()

con = telnetlib.Telnet()

LoginFrame(None, -1, title="Login", size=(320, 250))

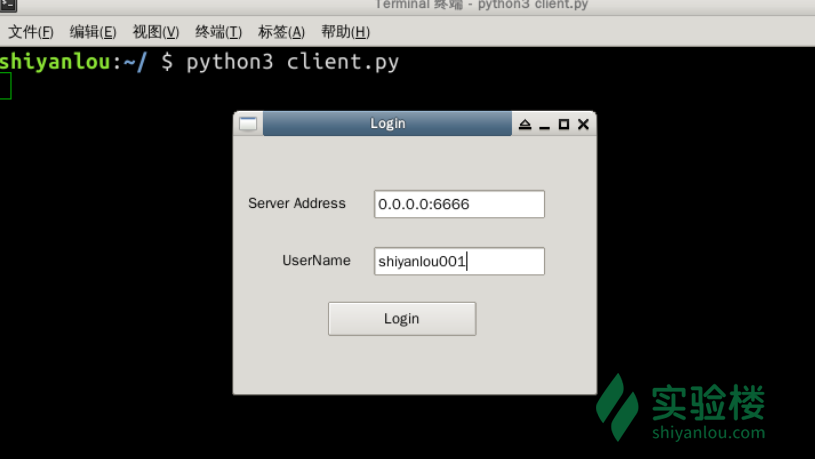
app.MainLoop()

五、执行

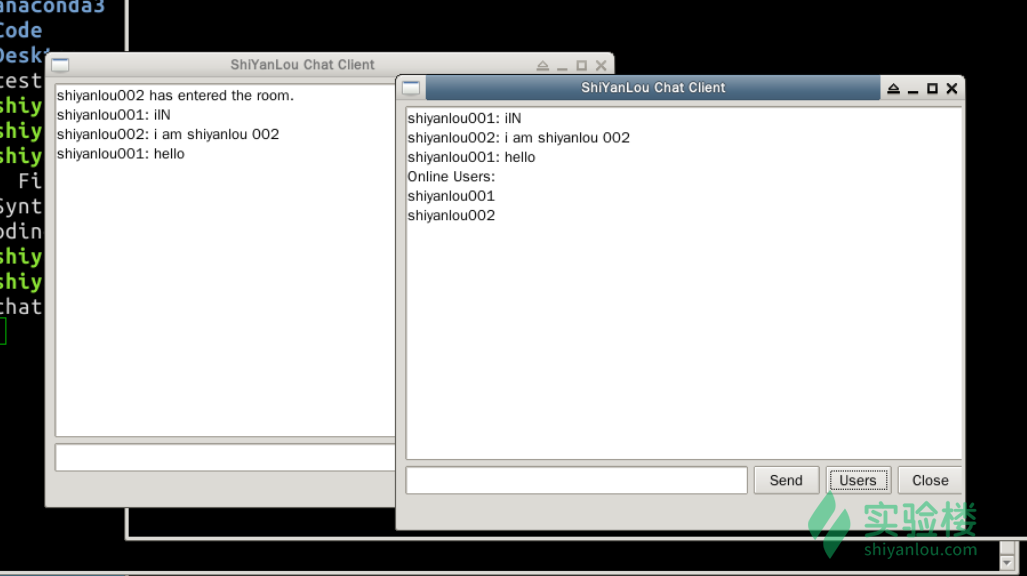
* 首先，我们执行 server.py ，如下图所示:



* 这时，我们再打开一个终端，执行 client.py 文件，如下图:



* 输入对应的信息之后，点击 Login ，再次重复上一步骤，使用另一用户名 shiyanlou002登陆，如下图:



* 在最终的示例中，我们可以分别通过 shiyanlou001 和 shiyanlou002 的客户端发送消息，此时，所有的在线用户都可以收到对应的消息。

如果你在实验中遇到问题，使用下面的命令获取实验楼提供的参考代码：

# 下载

wget http://labfile.oss.aliyuncs.com/courses/970/code.tar

# 解压

tar -xvf code.tar

六、项目扩展

这里的图形界面使用的是 wxPython，试着换一个图形界面包来实现客户端。

七、小结

最后就可以运行程序进行聊天了，注意需要先启动服务器再启动客户端。这个项目中使用了 asyncore 的 dispatcher 来实现服务器，asynchat 的 asyn\_chat 来维护用户的连接会话，用 wxPython 来实现图形界面，用 telnetlib 来连接服务器，在子线程中接收服务器发来的消息，由此一个简单的聊天室程序就完成了。