一、项目概述

使用 Python 来操作 socket ，实现一个聊天室的一些主要功能。一般的聊天室都有多个用户可以同时在线，他们可以实时获取到消息，实时发送消息。

服务端的实现功能：

1、监听客户端的连接
2、同时操作多个用户
3、广播消息通知

客户端的实现功能：

1、用户可以运行一个聊天室软件
2、可以在里面看到所有聊天室用户的消息
3、自己可以编辑消息进行发送

二、服务端代码实现

我们要使用 socket 套接字编程，消息记录需要时间。还要做到 “同时” 去操作用户，就需要用到多线程：因此，引入一下几个包。

```
# 套接字编程包导入
import socket
# 线程包导入
from threading import Thread
# 时间包导入
import time
```

接着我们要对服务器端进行初始化，绑定 ip 地址和端口号，设置最大服务用户数量，并设置绑定数据字典。

```
# 初始方法
def __init__(self):
    # 无参数默认ipv4，tcp协议
    self.server = socket.socket()
    # 绑定ip,bind((ip地址,端口号))
    self.server.bind(("127.0.0.1",8989))
    # 设置最大挂起数量
    self.server.listen(5)
    # 设置所有的客户端
    self.clients=[]
    # 使用用户名字与ip的绑定信息设置用户字典
    self.clients_username_ip={}
    # 创建实例时开始连接
    self.get_conn()
```

设置监听客户端连接函数，若有新客户端接入端口执行的一系列操作。
1.将该用户添加进入服务器的用户列表。
2.为客户端启动线程维护服务。

```
# 监听客户端连接
def get_conn(self):
    while True:
        # 获取连接客户端的信息
        client,address=self.server.accept()
        print(address)
        data="与服务器连接成功！请你输入昵称才可以聊天。"
        # server与client通信通过send()(!!需要encode())和recv()(!!需要decode())
        client.send(data.encode())
        # 把连接的用户添加到服务器的用户列表当中
        self.clients.append(client)
        # 服务器启动多个线程，处理每个客户端的消息，一个线程维护一个客户端
        Thread(target=self.get_msg,args=(client,self.clients,self.clients_username_ip,address)).start()
```

对所有客户端发送的信息进行处理。

```
# 进行所有客户端的消息处理
def get_msg(self,client,clients,clients_username_ip,address):
    # 接收客户端发来的昵称
    username=client.recv(1024).decode()
    print("from client "+username)
    # 将昵称与ip绑定
    clients_username_ip[address]=username
    # 循环监听所有客户端的消息
    while True:
        # 异常检测，获取用户的所有发送的消息
        try:
            recv_data=client.recv(1024).decode()
        # 抛出任何异常
        except Exception as e:
            # 直接break退出
            self.close_client(client,address)
            break

        #如果用户退出，输入Q
        if recv_data.upper()=="Q":
            self.close_client(client,address)
            break
        # 给每一个用户发送信息
        for c in clients:
            # 谁在什么时候发送了什么消息
            c.send((clients_username_ip[address]+" "+time.strftime("%x")+"\n"+recv_data).encode())
```

用户断开连接并离开后，服务器端的操作。

```
# 关闭资源
def close_client(self,client,address):
    self.clients.remove(client)
    client.close()
    print(self.clients_username_ip[address]+"已经离开了")
    for c in self.clients:
        c.send((self.clients_username_ip[address]+"已经离开了").encode())
```

三、客户端代码实现

客户端初始化绘制界面，并与服务器连接。

```
def __init__(self):
    # 初始化界面
    QWidget.__init__(self)
    # 设置窗口的大小与位置
    self.setGeometry(600,300,360,300)
    # 设置标题
    self.setWindowTitle("聊天室")
    # 添加背景
    palette = QtGui.QPalette()
    bg=QtGui.QPixmap(r"./image/background.jpg")
    palette.setBrush(self.backgroundRole(),QtGui.QBrush(bg))
    self.setPalette(palette)
    self.add_ui()
    # 与服务器链接
    self.client = socket.socket()
    self.client.connect(("127.0.0.1",8989))
    self.work_thread()
```

设置客户端中界面的各个组件。

```
# 设置界面中的组件
def add_ui(self):
    #多行文本显示，显示所有的聊天信息
    self.content=QTextBrowser(self)
    self.content.setGeometry(30,30,300,150)

    # 单行文本，消息发送框
    self.message = QLineEdit(self)
    self.message.setPlaceholderText(u"输入发送内容")
    self.message.setGeometry(30,200,300,30)

    # 发送按钮
    self.button = QPushButton("发送",self)
    self.button.setFont(QFont("微软雅黑",10,QFont.Bold))
    self.button.setGeometry(270,250,60,30)
```

客户端发送信息、接受信息等处理函数过程。

```
# 发送消息
def send_msg(self):
    msg = self.message.text()
    self.client.send(msg.encode())
    if msg.upper()=="Q":
        self.client.close()
        self.destroy()
    self.message.clear()

def btn_send(self):
    self.button.clicked.connect(self.send_msg)


# 接收消息
def recv_msg(self):
    while True:
        try:
            data = self.client.recv(1024).decode()
            print(data)
            data=data+"\n"
            self.content.append(data)
        except:
            exit()



# 线程处理
def work_thread(self):
    Thread(target=self.btn_send).start()
    Thread(target=self.recv_msg).start()
```