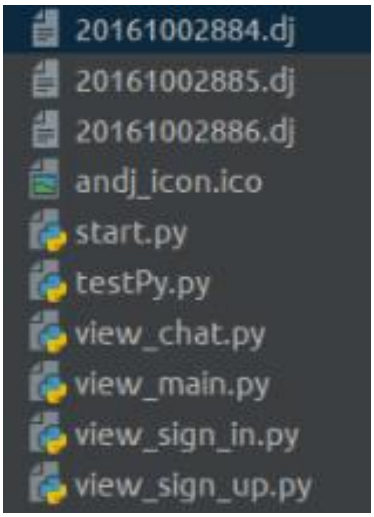
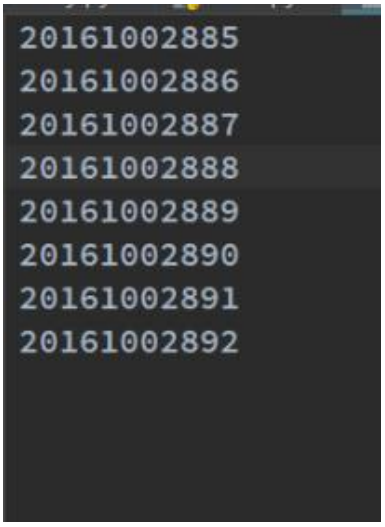
基于 UDP SOCKET 实现一个简单的 QQ 客户端

1、代码文件结构





1. *.dj 文件

   配置文件。由于服务端没有针对每一个账号的好友列表进行存储，这里统一存储为：账号.dj 文件，该文件结构如右上角所示，即当前账号的所有朋友账号单行存储。由于 python 文件读写的特殊原因，在本文件中最后一行有换行符，每一行的格式为“账号\n”的形式。

   该程序不包含程序内添加好友的功能，因此如果有添加好友的需求，请将账号添加至对应账号的配置文件中，并记得最后一行换行。

2. Andj_icon.ico 文件

   图标文件，用于替换 tkinter 自带的左上角窗口图标。

3. Start.py 文件

   程序启动文件。该脚本执行后打开应用程序。

4. Test.py 文件

   测试文件。用于向 20161002884 账号以 20161002885,20161002886 的身份发送消息。

5. View_chat.py 文件

   聊天视图文件。该文件中封装了关于聊天界面的类。

6. View_main.py 文件

   主窗口文件。类似于 QQ 的主窗口，该文件中封装了好友列表存在的窗口。

7. View_sign_in.py 文件

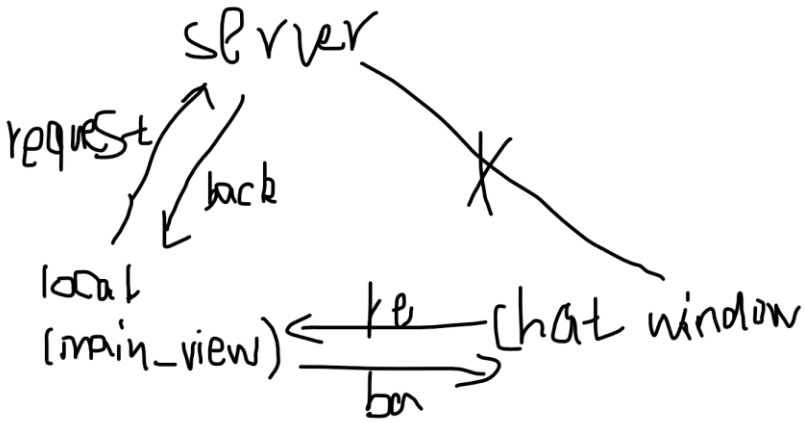
   注册窗口文件。该文件封装了注册窗口。

8. View_sign_up.py 文件

   登录窗口文件。该文件中封装了登录窗口。

2、代码整体架构

1. MVC 式架构，view 视图封装为四个以 view 开头的脚本中，start 为主程序，在主程序中实现交互。同时，利用 Python 多线程进行辅助运行。
2. 功能分开，登录，注册，聊天等分开执行，用多线程进行类似于广播的效果。
3. 数据传输过程:



即用户与 chat window 进行交互，chat window 不直接与 server 进行信息交互，而是通过在本地运行的后台程序 main_window 将消息中转至服务器，服务器来的消息也是这样。

3、交互细节

1. 用户登陆

执行 start.py 脚本，开启 view_sign_in 窗口，用户输入并拼接字符串，用 02 协议进行登陆。弹出窗口进行提示操作，会出现已登录，登陆失败等情况出现。同时，如果需要注册账号，那么打开注册窗口，填入相应信息，拼接字符串并用 01 协议进行注册，同样返回状态码。此时，需要手动关闭 view_signed_up 窗口返回进行登陆。

2. 后台服务开启

用户登陆成功后，main_view 开启，打开了 QQ 的主界面。

首先，程序会根据登陆用户查找该用户的配置文件，配置文件内容为该用户在本地存储的好友列表。如果该用户第一次登陆，没有响应配置文件，那么程序为该用户建立配置文件，文件名为 账户.dj 。读取文件自动载入主界面好友列表，即一个 button list。

需要注意的是，程序另外加入了一个 stranger 的 button，这个好友是指接收非好友所发的消息，在之后再做解释。此外，为了方便对所有的好友的消息进行缓存，采用一个字典结构进行存储，该字典中以 friendname:[message list]作为键值对进行消息缓存，message list 是一个 list。该结构的对象名为 receivemessages。

界面构建执行完毕。开启两个消息收发线程。由上面的架构图可知，主界面需要承担两个功能，即向服务器进行消息应答，以及对聊天窗口进行接收消息和应答。

因为该后台程序需要与 chat window 进行 socket 通信，**绑定的 localhost 端口为 21567，是固定的，因此本程序不能在同一个 PC 上同时登陆两个账户。**

线程一：承担与 server 进行信息交换，该线程 sleep2 秒钟向 server 进行请求，请求的方式为 08#，返回针对当前该账号的所有未读信息条数。之后，向服务器进行该条数数量的 09#请求，获取目前所有服务器存储的消息。每次消息获取时，判定该消息是哪一个 friend 发来的，并存储至字典对应的列表。这里明显解决的问题是，如果有消息的来源方在好友列表里找不到，那么程序标记为 stranger，统一放置在 stranger 的 list 里面。

请求完之后，为了让获得的数据引起用户的注意，程序遍历了字典，如果字典里某一个 list 有存储信息，即该 list 相对应的 friend 发的消息还没有被接收，那么将该 list 对应的 button 背景色改为红色，这样就起到了注意的作用。如果对应 list 为空，就改为白色。每 2s 刷新一次。

线程二：承担与 chat 进行通信。该线程接收 chat window 的消息。聊天窗口产生的消息有两种，**一种是将消息发给指定用户，另一种是查看现在有没有用户给当前账户发消息。**

1. 将消息发送给指定账户，chat window 会给 21567 端口发消息，消息格式为 sending:friendname:message，针对该字符串进行切割和拼接，使用 03 协议将该消息发送给指定账户，将发送的状态码返回给 chat_window。（stranger 窗口已禁用输入框）
2. 收到 chat window 的检查是否有未读信息的消息，该消息格式为： friendname，即与当前账户聊天的账户。这里因为之前将服务器端存储的信息进行了本地缓存，存至字典 receivemessages 中，因此通过关键字 friendname 取出消息 list，遍历并通过 socket 将消息逐条发送至 chat window，实现了与 chat window 交互。

3. 聊天窗口开启

聊天窗口进行了初始化，chat window 进行的交互主要有两个：

1. 用户发消息，将该消息发送给之前的本地服务发送给指定账户。
2. 如果有消息发送至该账户，那么将消息读取并显示。

第一个任务由 button 触发，添加一个 button 用户发送消息，该 button 将 sending:friendname:message 格式的消息发送至 21567 端口程序，返回发送消息状态码。

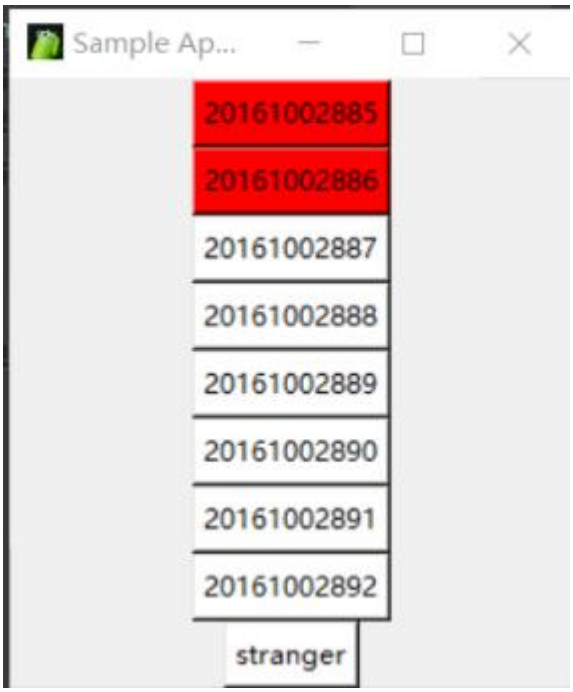
第二个任务需要不断向 21567 发送请求，这里另开了一个线程，将 friendname 发送至 local，如果有消息，将返回的消息截取字符串插入至聊天消息界面。

4、测试

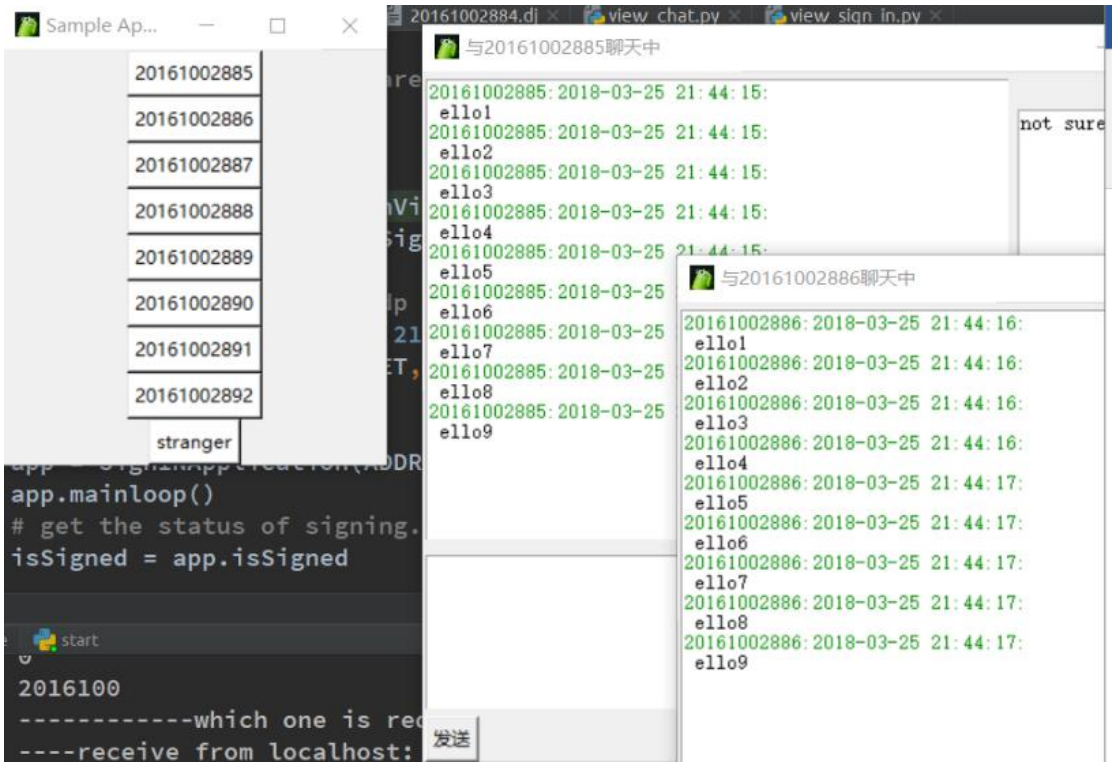
1. 启动测试脚本 testPy.py，该脚本执行的逻辑为，分别登录 20161002885 20161002886，

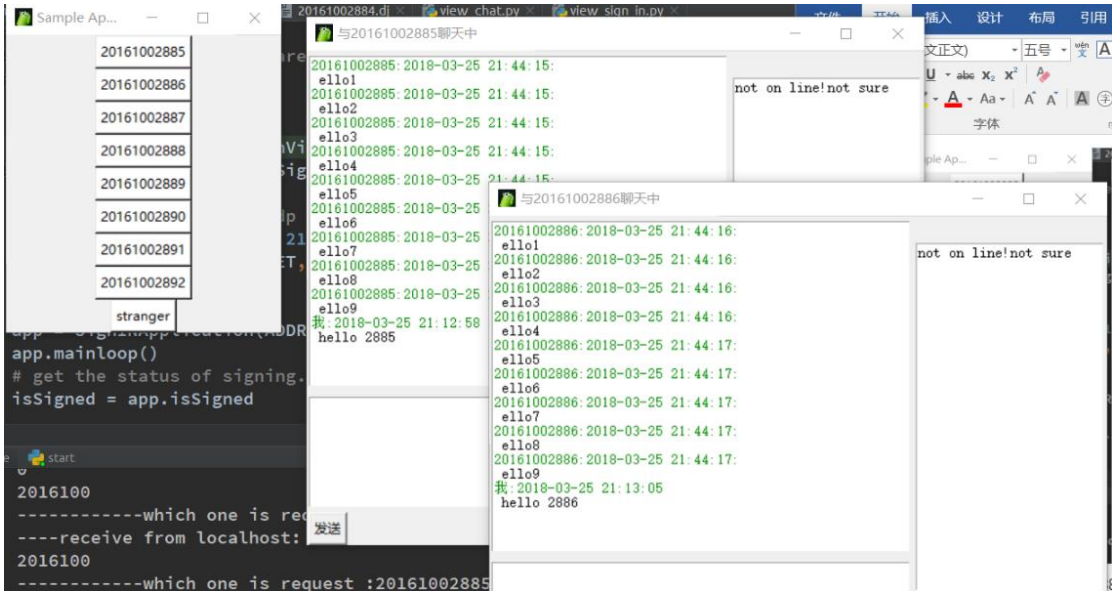
每一个账户给 20161002884 发 9 条消息。共 18 条消息。

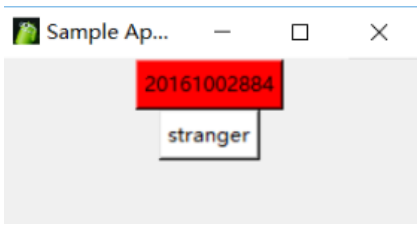
2. 20161002884，20161002885，20161002886 的好友列表配置文件已给出：注意 20161002886 是空好友列表。
3. 启动 start.py 开启程序，登录账号 20161002884 ，密码为 363787
4. 

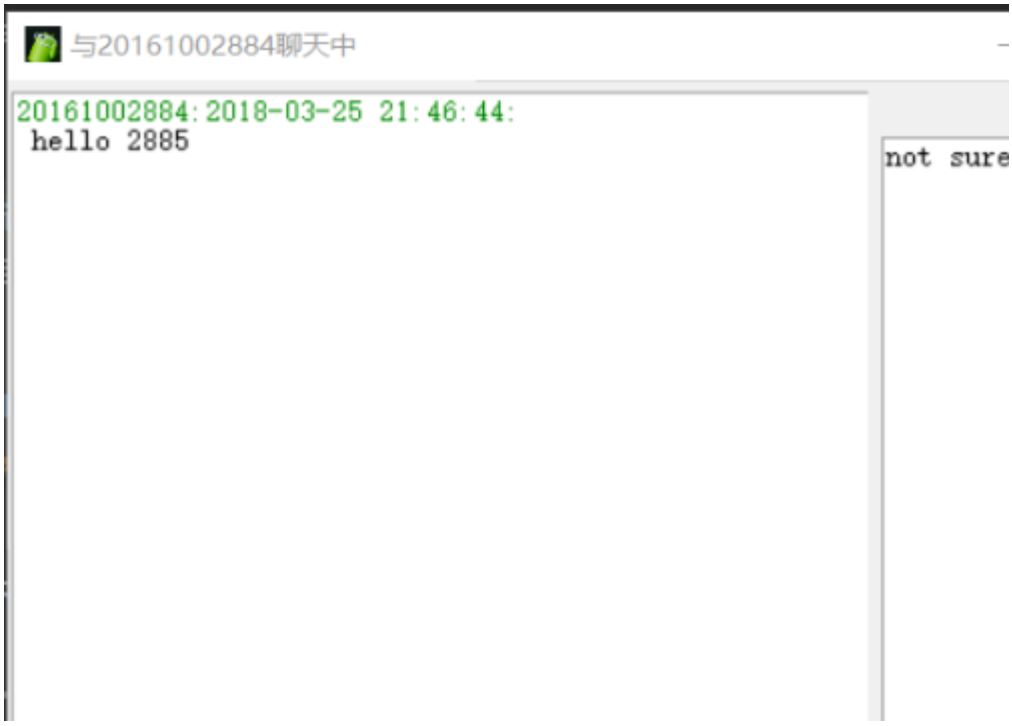
5. 跳转至 main view 界面，等待数秒，20161002885 和 20161002886 的 button 亮为红色。
6. 

7. 分别点击 20161002885 和 20161002886，打开两个聊天窗口，分别可以接收 9 条消息。

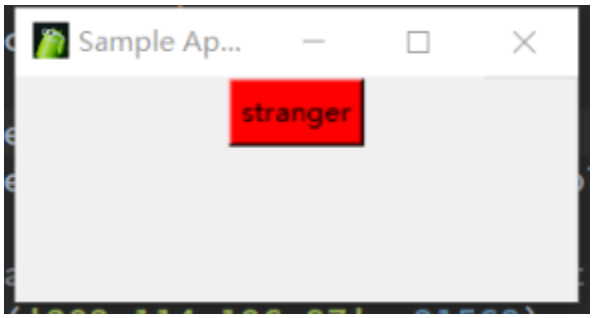
8. 

9. 给 20161002885 发消息：hello 2885**（不能是中文，不支持中文）**
10. 给 20161002886 发消息：hello 2886
11. 

12. 分别登陆两个账号，20161002885 的 20161002884 button 亮红色并接收消息，20161002886 的 stranger button 亮红色并接收消息。
13. 20161002885：
14. 



15. 20161002886:

16. 

17.