

Socket 编程报告

17307130009 李逸飞

目录

1.功能介绍

2.使用说明

3.文件目录

4.代码详解

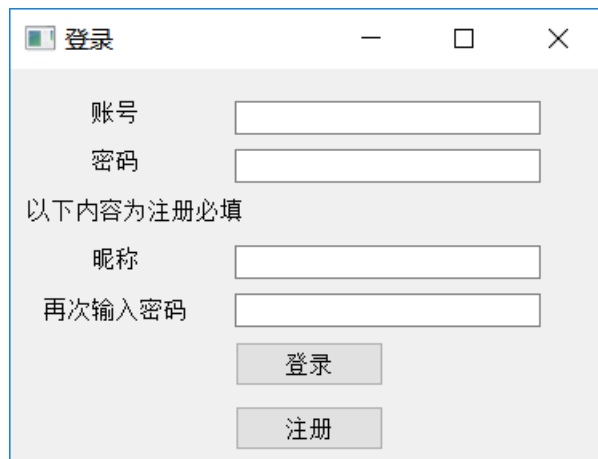
- socket 基本
- 协议基本
- 基本架构
- 协议详细

5.总结

一. 功能介绍

我选择的项目为设计一个简单的聊天程序，客户端大致界面如下：

登陆界面 UI



登录

账号

密码

以下内容注册必填

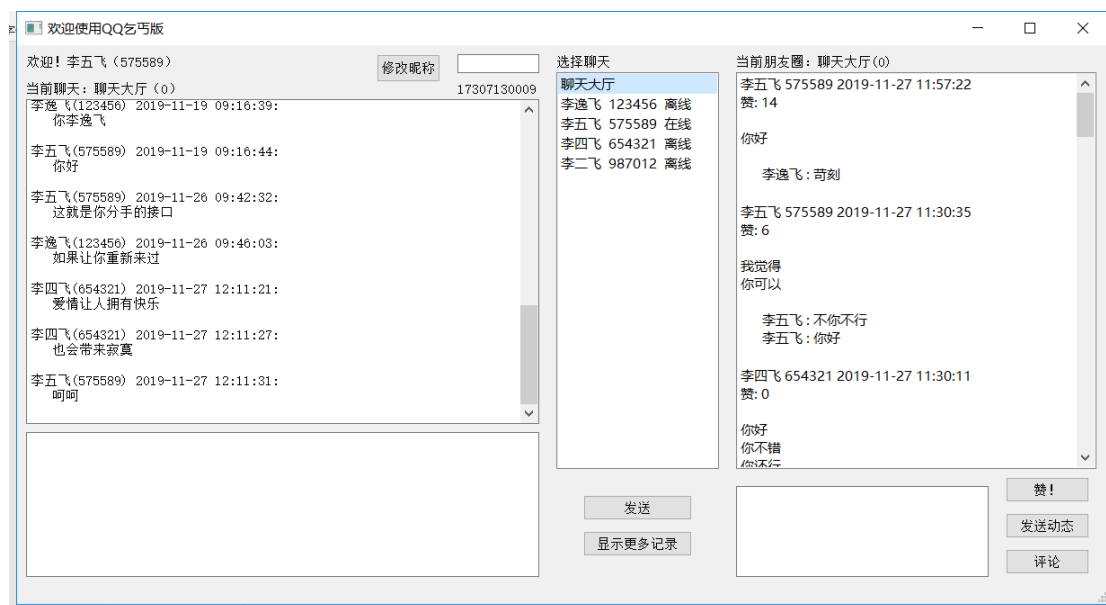
昵称

再次输入密码

登录

注册

主界面 UI



欢迎使用QQ乞丐版

当前聊天: 聊天大厅 (0)

李逸飞(123456) 2019-11-19 09:16:39: 你李逸飞

李五飞(575589) 2019-11-19 09:16:44: 你好

李五飞(575589) 2019-11-26 09:42:32: 这就是你分手的接口

李逸飞(123456) 2019-11-26 09:46:03: 如果让你重新来过

李四飞(654321) 2019-11-27 12:11:21: 爱情让人拥有快乐

李四飞(654321) 2019-11-27 12:11:27: 也会带来寂寞

李五飞(575589) 2019-11-27 12:11:31: 呵呵

选择聊天

聊天大厅

李逸飞 123456 离线

李五飞 575589 在线

李四飞 654321 离线

李二飞 987012 离线

当前朋友圈: 聊天大厅(0)

李五飞 575589 2019-11-27 11:57:22 赞: 14

你好

李逸飞: 苛刻

李五飞 575589 2019-11-27 11:30:35 赞: 6

我觉得你可以

李五飞: 不你不行

李五飞: 你好

李四飞 654321 2019-11-27 11:30:11 赞: 0

你好

你不错

你还行

发送

显示更多记录

赞!

发送动态

评论

实现的功能有：

1. 用户账号注册登录
2. 多人即时聊天
3. 一对一私聊
4. 聊天记录保存
5. 朋友圈发送动态
6. 朋友圈点赞评论
7. 修改昵称等其它细节

二. 使用说明

1. 注册登录十分简单，注意用户名固定为 6 位数字，密码为 6~15 位任意字符，如下昵称

为违法昵称：空字符串，含有空格、'|'、'&&'、'*'，同时昵称也不能为"聊天大厅"，此外昵称最大长度为 10。

2. 登录后进入朴实无华的主界面，中间的框框为聊天列表，显示所有已注册的用户，并显示他们是否在线，因为考虑到注册的用户不会多，毕竟是自己玩。列表的第一项为聊天大厅，用于所有用户同时聊天，当然可以单击列表中个人用户的项，切换到和目标的单人聊天。在第一次载入时，默认加载最近 20 条聊天记录，点击"显示更多记录"按钮可以加载所有的聊天记录。

不同于微信，QQ 等需要加好友，我的聊天软件更像是一个简易版的"公司圈"，可以方便地联系到任何在公司里的同事。

3. 点击发送按钮发送消息，注意消息也含有同昵称一样的违法字符，这是由于协议涉及产生的缺陷，好在这些字符并不常用。在收到消息时，如果当前不在和消息发送方聊天，聊天目录的对应项的背景就会显示为粉色，需要切换到该聊天后恢复颜色。发送消息有 1 秒一次的限制。



4. 可以在聊天列表的左上方的文本框输入新昵称并点击修改，不违法的话立刻就能修改成功。聊天列表固定为 15 秒进行一次刷新，刷新后可以看到更新的昵称，当然在线离线、新注册用户也会在此时更新。

5. 右边一栏为朋友圈部分，通过双击聊天列表中的项切换到目标用户的朋友圈，如果是聊天大厅则会显示所有人的朋友圈。可以选中某一条朋友圈对齐点赞和评论，也可以直接发送一条动态，对点赞次数没有限制，但每秒仅能执行一次点赞操作。点赞和评论会即时更新。如果要刷新动态，再一次双击列表中对应该项即可。评论和发送动态有 3 秒一次的限制。

6. 当用户收到来自其他用户的点赞或评论时，聊天列表中该用户的文字会变绿，双击后切换查看，并恢复颜色。一条朋友圈的格式如第一部分所示，上方为发送者和点赞数，中间为内容，下方为评论部分。由于技术的限制依旧不够美观，我表示非常遗憾。



三. 文件目录

技术栈：python 的 socket API + python 的 GUI 库 pyqt5 + Mysql

主要文件如下：

- defmain.py
客户端的逻辑以及协议处理实现
- server.py
服务器的逻辑以及协议处理实现
- ui_login.py
客户端登录界面的 UI 实现
- ui.py
客户端主界面的 UI 实现
- socket_data.sql
数据库脚本

Python 的 socket API 为课程规定；Pyqt5 和 C++ 的 QT 使用方式非常相近，可以做成比较简洁的界面，当然相比 web 肯定会少许多视觉冲击力；由于需要保存聊天记录以及用户注册登录等数据，Mysql 是必不可少的。

四. 代码详解

1. socket 基本：

首先服务器采用的是多线程处理客户端 TCP 连接请求的基本模板，每当一个新客户端请求连接，服务器立马分出新线程来处理客户端的请求。因为是多人即时聊天的模式，当聊天人数较多时也需要尽快满足所有客户端的请求，不采用线程池的模式。采用了错误处理，在服务器与客户端的连接断开之后会立马更新其在线状态相应的数据结构。

2. 协议基本：

设计应用层协议方面，为了方便编程与理解就单纯地使用分隔符来确认是哪种类型的请求。

服务器发送给客户端的基本协议如下：

```
#处理服务器发来数据的协议
def handle_recv(s):
    params = s.split('||')
    switcher = {
        'ack_login': ack_login,
        'ack_sign': ack_sign,
        'ack_refresh_list': ack_refresh_list,
        'ack_change_nick': ack_change_nick,
        'ack_send_text': ack_send_text,
        'ack_acquire_chatrecord': ack_acquire_chatrecord,
        'ack_acquire_pyq': ack_acquire_pyq,
        'ack_pyq_gooded': ack_pyq_gooded,
    }
    func = switcher.get(params[0], lambda: "nothing")
    return func(params)
```

客户端发送给服务器的基本协议如下：

#来自客户端的请求列表

```
def req_handler(params, connection):
    switcher = {
        'req_login': req_login,
        'req_sign': req_sign,
        'req_refresh_list': req_refresh_list,
        'req_change_nick': req_change_nick,
        'req_send_text': req_send_text,
        'req_acquire_chatrecord': req_acquire_chatrecord,
        'req_acquire_more_chatrecord': req_acquire_more_chatrecord,
        'req_send_pyq': req_send_pyq,
        'req_comment_pyq': req_comment_pyq,
        'req_acquire_pyq': req_acquire_pyq,
        'req_pyq_good': req_pyq_good,
    }
    try:
        func = switcher.get(params[0], lambda: "nothing")
    except Exception:
        print('proto error!')
        return
    return func(params, connection)
```

协议名称会在发送的内容前添加，并以分隔符 '||' 与发送的数据加以区分。那么在客户端或者服务器收到数据时，读取分隔符 '||' 前的第一段字符串的内容就是协议名称，python 的字符串操作 join 和 split 能非常方便地达到这些目的。

通过字典映射的方式就能调用协议名称相对应的接口并执行，当然同样拥有错误处理，对于没有在字典中的协议，会单纯地什么都不做。

这样添加协议的话，只需要在字典里添加映射，在实现相应的接口即可，方便了很多。

此外，所有协议的数据内容部分也都有分隔符加以区分，这样就能顺利地读到正确的数据。当然这么做的话，额外的分隔符会增加不必要的流量，这也是一个缺点。

3. 基本架构

1) 客户端

客户端的代码主要分为两块，发送协议部分和接受协议部分。接受协议部分额外分出一个线程来单独处理，由于主界面 ui 有不能在子线程更新的限制，需要采取信号的机制，这是在开发时遇到的一个小困难。发送协议部分则根据用户的操作来决定协议内容并发送给服务器。

2) 服务器

服务端的架构相对简单，分为 socket 部分和接发协议部分。Socket 部分处理客户端的连接和断开，在这里会维护一个字典 id_conn 在储存在线的用户的 id 对 connection 的映射，这样就能解决广播的问题。接发协议部分会根据接收到的来自客户端的协议，处理若干个逻辑或数据库等操作之后发送给客户端数据以表示回应。

3) 数据库

数据库方面，建立如下五个表：

users: 保存用户信息，元素有用户名，密码，昵称，主键为用户名

chat_records: 保存私聊信息，元素有发送者用户名，接收者用户名，发送时间，发送内容。主键为发送接受者用户名和发送时间，意味着同一对用户每秒钟只能有一条消息，这个限制在客户端中已经被保证了。

chat_global: 保存聊天大厅信息，元素有发送者用户名，发送时间，发送内容。主键为发送者用户名和发送时间

pyq_main: 保存朋友圈主要信息，元素有发送者用户名，发送时间，发送内容，点赞数。主键为发送者用户名和发送时间，意味着同一用户每秒只有一条朋友圈，这个限制同样在客户端被保证。

pyq_comment: 保存朋友圈评论信息，元素有发送者用户名，发送时间，评论者用户名和评论内容。没有主键。可以通过发送者用户名和发送时间唯一确认一条朋友圈。

若要配置数据库，先创建一个名为'socket_project'的模式，在其中执行 socket_data.sql 脚本。

对于服务器的几乎每一个数据库操作都有相应的 try-except 错误处理以防止崩溃。

4. 详细协议

1) 注册登录

客户端对应的发送协议为'req_login'、'req_sign'，服务器对应的发送协议为'ack_login'、'ack_sign'。

客户端进行注册，会先判断内容是否合法，合法后将'req_sign'、用户 id、密码、昵称以分隔符'|'连接的形式发送给服务器，这也是为什么分隔符为违法字符的原因。

服务器接收到后先判断该 id 是否已被注册，判断方式是维护了一个字典 user_list 来储存所有用户的 id 对昵称的映射，这个字典是在服务器初始化时从数据库读入的，这样相对每次请求都读取数据库会快上许多，若已被注册，则返回'ack_sign'协议中的内容为注册失败，若未被注册则注册成功，返回的'ack_sign'协议为注册成功。

客户端收到注册成功的消息则会登录并显示主界面 ui。

登录操作更简单，客户端发送'req_login'协议，这里服务器需要读取数据库来判断是否密码正确，因为把密码时刻维护在服务器内存中不仅浪费空间还并不安全。返回'ack_login'的协议包含是否登录成功。

2) 刷新列表

对应的客户端协议为'req_refresh_list'，服务器为'ack_refersh_list'。

客户端在登录成功之后会额外分出一个线程，每隔一定时间（暂时设置为 15 秒）发送'req_refresh_list'的请求。用于更新用户的在线离线状态以及昵称更改。

服务器接收到请求后，将所有用户的在线状态以及昵称账号分隔符形式发送给客户端，协议头为'ack_refresh_list'。先前有提到，服务器在内存维护着一个在线的字典 id_conn 和所有用户的字典 user_list，这样就可以轻松区分用户是否在线。

客户端收到对应的 ack 协议就会更新列表内容。

3) 修改昵称

'req_change_nick'、'ack_change_nick'。

客户端读取昵称框中内容判定合法后发送 req 协议，包含 id 和修改后的昵称。
服务器收到请求后更新维护的 user_list，并进行数据库更新。返回修改成功的 ack。
客户端收到对应的 ack 协议就会更新昵称。

4) 接发消息

'req_send_text'、'ack_send_text'

客户端判断发送消息是否合法后发送 req 协议，包含 id 和接收者 id 和内容。如果为聊天大厅，那么接收者 id 设置为“0”。

服务器接收到 req 后，对数据库进行插入操作，成功后对接收者以及发送者都发送 ack 消息，内容为发送者 id、时间和消息内容。如果为聊天大厅，则改为对所有在线用户都发送 ack 消息。

客户端在接收到 ack 后，更新对应的缓冲池（稍后会提及），如果当前聊天恰为 ack 中的发送者，那么会更新消息界面，否则将聊天列表中对应用户的项背景置为粉红，如先前的图中所示。

5) 聊天记录获取

'req_acquire_chatrecord'、'req_acquire_more_chatrecord'、'ack_acquire_chatrecord'

'req_acquire_chatrecord'协议仅在客户端第一次切换到与目标用户的聊天时才会发送。在收到 ack 之后就会更新本地一个消息缓冲 CHAT_BUFF，这是一个目标用户 id 和聊天记录的映射，之后的切换都是直接读取消息缓冲区的内容。CHAT_BUFF 也会根据收发消息实时更新。这么做是为了尽可能减少服务器读取数据库的负担。

'req_acquire_more_chatrecord'协议在点击“获取更多消息”按钮后会发送。在收到 ack 之后是同样的更新操作。

服务器收到第一个 req 协议后，会从数据库读取 20 条对应的聊天记录返回给客户端。第二个协议则是 80 条。统一采用的是'ack_acquire_chatrecord'协议。

6) 发送朋友圈动态

'req_send_pyq'

客户端在判断合法后发送该协议，包含 id，时间和内容。之后不等服务器的反馈，直接本地更新朋友圈界面的内容，这是出于节省流量的考虑。当然发送朋友圈是设定为 5 秒限制的，这样就避免数据库的主键冲突，理论上是必然能够发送成功的。如果发送仍然失败则更新动态后就会消失。

服务器接收到 req 后会直接执行插入数据库的操作。这条协议并没有 ack。

7) 刷新/切换朋友圈界面

'req_acquire_pyq'、'ack_acquire_pyq'

在双击聊天列表的某一项后，客户端就切换朋友圈到目标用户，发送 req 协议。

服务器在接收到 req 后从数据库读取对应的内容，并将最近的 25 条朋友圈组成 ack 协议发送给客户端。

客户端根据 ack 内的朋友圈内容以固定格式更新朋友圈界面。

在这里说的比较简单，但实际上考虑到传输的数量可能相当的大，于是将发送的朋友圈最多限制到 25 条。此外朋友圈更新的格式、协议的分隔方式也花了一定时间才得以解决。

8) 朋友圈点赞和评论

'req_comment_pyq'、'req_pyq_good'、'ack_pyq_gooded'。第一个 req 为评论，第二个为点赞。

客户端在执行点赞或评论操作后，发送包含对象 id 和自己 id 和内容（如果为评论）的对应 req 协议给服务器。点赞的间隔被限制为 1 秒，评论的间隔被限制为 3 秒。在发送之后会直接本地更新内容，而不是等待 ack 后再更新，同样是为了节省流量。

服务器收到 req 协议后更新数据库，之后为**通知被点赞或评论的在线用户**，向他发送 'ack_pyq_gooded' 协议。

被点赞或评论的在线用户收到这条 ack 协议后，将聊天列表中自己的项字体颜色置为绿色，如第一部分的图所示。

这部分也有小难点就是客户端朋友圈的读取方式，为了顺利从文本中读取到朋友圈的关键信息，如发送者，评论者，点赞数等，需要对朋友圈的格式仔细的设计，但又不能让它显得太丑，显然我在这方面做得并不好，这也是我不太满意的地方。

五. 总结

在暑假的实习时我就接触到了服务器编程，也了解了一个程序员在开发服务器时的编程习惯，如封装和框架等的设计等。socket 编程也给了我一次机会去尽可能的模仿再现当初我老大的编程习惯，当然我做得并不好。我认为不太满意的地方如下：

第一点是在数据库的读取方面，作为服务器应当尽可能减少琐碎数据库的读写操作以增加效率，为此就需要设计大量的缓冲区，因为公司服务器的强大配置而无需太担心内存问题，这方面虽然我也有用到缓冲区，但在朋友圈的切换读取方面依旧琐碎地操作了数据库，这点是我比较不满意的地方。

第二点是在朋友圈的 ui 设计，说白了就是太丑，可能是 pyqt 的限制，也可能是我第一次碰 Python 技术太差，没有了解更好的控件。这点相信我在更熟练使用 python 之后能有所改善。

当然还有我比较满意的地方。

首先是程序的架构方面，我认为我的代码框架相比于去年的项目有了显著进步，至少在代码的功能明确、函数和协议的命名、面向对象接口的封装方面在源码中都有所体现，这对于程序员来说是个必须要养成的好习惯。

还有就是总算学了 python，在大三才正式接触 python 的我可能已经很晚了，python 语言的重要性不需多言，反正学了就是好事。

最后总之就是项目对我的意义很大，效果也很好。