基于socket的简单的小说阅读器设计

总述：该项目要求使用socket编程，并且不使用第三方的拓展包来使用python开发一个简单的小说阅读器。

平台：Windows

编程语言：python2.7.13

可视化界面：Tkinter

项目目录：

Socket\_Project

-server.py

-func.py

-client.py

-log.json

-content.json

-novel

--bigchapters(multiple)

---chapters(multiple)

项目分为服务器端和客户端两个方面。

服务器和客户端之间考虑使用TCP连接。服务器处于监听状态，客户端向服务器发送请求才会建立连接。

首先，需要制定一个包传输协议。因为使用socket传输数据是可能会出现丢包或者其他情况的，所以，不能单纯使用recv和send函数来传输数据。由于该阅读器仅仅会传输较小的小说文件，所以，包传输协议可以设置为如下方式：

|  |  |  |
| --- | --- | --- |
| 头部 | 正文长度(int) 4 bytes | CMD(int) 4 bytes |
| 正文 | 最长为int可表示长度（4GB） | |

由于传输的文件大小可预知较小，不可能到达4GB，所以正文的可支持大小足够。并且，正文都采用json的格式来传递，并且当中会有中文的话，都直接传输未解码的str数据，并且在接收完数据后自行根据需要进行解码。

根据这个传输协议，写出数据传输的函数send\_all(socket, cmd, str)和recv\_all(socket)。

在需要进行数据发送时调用send\_all函数，该函数未设置返回值。数据接收的时候调用recv\_all函数，该函数返回一个dict类型的接收结果{‘cmd’: cmd, ‘bodypart’: bodypart}，其中cmd是传输的指令int，而bodypart是正文内容str。

其中send\_all是在函数中把头部和正文都打包好，然后使用sendall函数发送。

recv\_all是在循环中使用recv(2048)把数据接收到一个全局定义的BUFF中，根据BUFF和头部的长度（HEADERSIZE，8bytes）进行比较来判断头部是否已经全部接收到缓存中，如果头部都有了，那么就根据BUFF和正文长度大小来判断正文是否也都接收完毕。

服务器端和客户端设置的默认HOST是‘localhost’,PORT是9001，可根据需要在文件中修改。

服务器端采用过程式编程，运行一个循环函数处理逻辑，仅当客户端断开连接时候退出循环，等待下一个客户端接入。

客户端采用面向对象编程，定义了myTk的类，是客户端的主窗口界面，由于还有登陆的界面，所以就定义了两个Frame，在登陆之后就destroy登陆的Frame，然后把小说阅读界面的主Frame加载到myTk的窗口中。

服务器端：

server.py：服务器运行主程序。监听端口，如果客户端连接，那么就调用func.py中的函数进行处理。这是为之后多用户的一个模板，如果是要针对多用户的话，就在这里增加一个线程，然后在新线程中调用处理函数来进行并行处理。

func.py：服务器跟进客户端连接的处理函数定义。定义了不同的cmd操作以及对应的数据包中的code。

# cmd code  
# exit 0  
# exit with log 1  
# log in 101  
# last chapter 201  
# next chapter 202  
# download 301  
# jump to mark 401

客户端：

client.py，使用Tkinter包编写图形化界面，采用自定义类的方法，实现面向对象的编程。其中定义了signinFrame, mainframe, myTk类，在myTk中定义了start函数，客户端只需要运行myTk实例的start函数即可。登陆frame非常简洁，这里只是在服务器端判断用户信息是否是，用户名为123，密码也为123。

程序要求是要实现一个书签的功能，因此，在考虑之后，程序的逻辑采用如下策略。

在登陆成功后就接收一个带上次退出时候阅读状态的章节数据。详情参看log.json。这个文件存放的是用户的退出前的阅读状态，包括，第几册（bigchapter/int），第几章（chapter/int），第几页（page/int），书签列表（marks/list）。

如果登陆未成功的时候退出了客户端，那么传递的就是cmd=0，表明用户关闭客户端，断开连接。如果是登陆成功后退出客户端，那么传递cmd=1，此时还传递了当前的客户端阅读状态，然后修改log.json文件。所以，在阅读中用户修改书签都只是在本地修改，并未将数据同步，而是当退出时候才进行同步。

用户可进行的操作有：上一页，下一页，上一章，下一章，添加当页到书签并且添加备注，查看书签，清空书签，根据书签号删除书签，根据书签号跳转到对应页面，根据页号跳转到对应页面，下载本章，退出。其中，翻页绑定了键盘的方向键，可使用方向键或着界面按钮进行操作。退出是绑定了Esc键，阅读时可直接按该键退出。

由于Tkinter功能的单一，因此，此程序编写时只考虑仅仅适用于分辨率为1920x1080的电脑，同时无法修改界面，包括字体以及背景颜色等，但是背景颜色可以在代码中对应位置修改，可参见代码中myTk函数中start的mainFrame定义：

self.mainFrame = mainFrame(master=self.myTk, socketin=self.mySocket, packet=self.buff, bg='brown')

其中bg即为窗口的背景颜色。

用户在按键执行操作的时候，向服务器传输包头部中包含对应的cmd，服务器根据cmd码来选择对应的处理过程。翻页操作时，首先是判断是否还未到章首或章尾，如果需要换章，就利用翻章的函数，向服务器发请求。跳页的功能不需要与服务器进行通信，仅仅只需要显示出对应的部分即可。添加书签和删除书签，清空书签也都不需要通信，只需要在书签列表中选择用户输入的标签进行操作即可，如果用户输入不符合逻辑，那么就弹出提示框。而跳转到标签号就需要判断是否需要向服务器请求对应的章节文件了。而下载的操作，就只需要判断下载的默认目录（项目文件夹的download夹）是否存在，不存在就需要创建，然后向服务器发送请求，并且把返回的文件写入到download文件夹中的以接收数据中的文件名为文件名的文件中即可。但是需要注意的是，接收的数据是字符串类型，中文是未解码的，所以在打开对应文件名的文件时候，需要把文件的名字使用decode(‘utf-8’)进行解码，才能正确打开文件。同时，对于需要显示的中文，也需要注意这一点，必须对字符串进行解码再输出才能更好的避免在屏幕上显示时候出现乱码的问题。

整个项目可以使用下面的流程图来表达：

登陆

登陆是否成功

否

是

服务器发送上次退出时的阅读状态

下载

翻章

跳转书签

增删

查书签

翻页跳页

客户端进行判定以及操作

向服务器请求数据

退出

向服务器发送当前的状态，包括第几册，第几章，第几页，书签列表。服务器将其存放在log.json中，以便下次向客户端恢复现场。

等待客户端下次连接