

# 復旦大學



## 计算机网络课设报告

基于 Socket 的网络阅读器

姓名: 马逸君

学号: 17300180070

2020 年 1 月

## 项目目标

本项目要求设计一个网络协议（最好是应用层协议），并采用标准 Socket API 编程实现。

我的选题是：基于 Socket 的网络阅读器。

## 开发步骤

第一步, 写出不包含任何功能的服务器和客户端骨架代码, 使得可以建立 Socket 连接。

第二步, 向服务器添加 FTP 功能, 让客户端发送符合格式要求的请求给服务器, 使得可以通过 FTP 传输 pdf 文件。

第三步, 使用 Qt5 Designer 设计前端 (参照[1]), 及与后端连接, 得到有图形界面的 PDF 网络阅读器。

第四步, 设计 TXT 阅读器的窗口, 集成到 PDF 阅读器的前端中 (作为子窗口弹出) 及与后端连接, 得到最终功能完备的支持 PDF 和 TXT 双格式的网络阅读器。

## 重要模块描述

### 1. 服务器工作流程、协议的分组格式

服务器的工作流程：启动时创建套接字，绑定套接字到本地 IP 与端口，并开始监听连接。本服务器模型中的 TCP 服务器是一个**多线程**服务器，每有传入连接就启动一个子线程去处理。

```

144     # 轮询socket状态, 一旦有传入连接, 就启动一个子线程去处理
145     no = 0
146     while True:
147         conn, addr = sock.accept()
148         no += 1
149         thread.start_new_thread(child_connection, (no, conn))
150         # if index > 10: # 最多处理10个连接, 然后服务器退出
151         #     break
152
153     sock.close()
154

```

子线程启动后, 首先做一些初始化工作, 包括在运行信息中输出"begin connection [编号]"、声明文件指针 (用于处理 TXT 文件) 等, 然后开始循环处理当前连接发来的信息直至客户方关闭连接。规定的**请求格式**为"数字+请求内容", 数字为请求类型, 取值范围为{0, 1, 2, 3}。

"0 [文件名]"——请求从服务器下载文件, 我们实现了一个 FTP 过程, 来完成文件的传输, 这一过程基于**分组传输**思想, 相关细节将在下一节讨论。我们的阅读器阅读 PDF 图书时, 将通过该请求从服务器端下载 PDF 文件到本地, 在本地打开并处理。

```

# 用请求信息的首位判断请求类别
if rq[0] == '0': # 下载文件, 格式: '0 filename'
    # 获得文件名和文件大小
    filename = rq[2:]
    print("客户端请求文件", filename)
    filesize = os.path.getsize(filename) # 文件大小, 单位: 字节

    # 发送报头给客户端, 报头内容为文件名和文件大小
    head_dic = {'filename': filename, 'filesize': filesize} # 报头 (字典)
    head_raw = json.dumps(head_dic) # 字典转换成字符串 (当然也可以使用eval进行转换
    head_len_struct = struct.pack('i', len(head_raw)) # 报头的长度, 转换为字符串
    conn.sendall(head_len_struct) # 发送报头长度
    conn.sendall(head_raw.encode('utf-8')) # 发送报头 (字符串)

    # 发送文件内容
    with open(filename, 'rb') as fp:
        data = fp.read()
        conn.sendall(data)

    print('发送文件完毕')

```

"1 [TXT 文件名]"——请求让服务器打开服务器上的一个 TXT 文件, 准备向客户端提供 TXT 文件的内容。服务器将调用专门的 TXT 处理函数来探测文件编码并打开文件指针, 该函数基于 chardet 包的 detect()进行, 具体过程从略。

```
elif rq[0] == '1': # 打开txt, 格式: '1 filename_without_extension.txt'
    filename = rq[2:]
    print('客户端打开文档', filename)
    fpageum = (os.path.getsize(filename) + txtpagesz - 1) // txtpagesz # txt的总页数
    fp = open_text(filename)
    conn.sendall(str(fpageum).encode('utf-8')) # 把总页数返回给客户端
```

"2 [页码]"——请求服务器提供打开的 TXT 文件的指定页的内容。页码也可以为 0，表示请求下一页。服务器的行为是：若是跳页，则使用 seek()将文件指针跳转到对应的位置（若是下一页则不用）；然后使用 read()读取文件指针开始的一页的内容，该页的大小是一个固定的字节数，并发送给客户端。

```
elif rq[0] == '2': # txt跳页及翻页, 格式: '2 0' (下一页) 或 '2 pgnum' (跳页)
    pgnum = int(rq[2:])
    if pgnum != 0:
        fp.seek((pgnum - 1) * txtpagesz) # 跳页用文件指针的seek()实现
        content = fp.read(txtpagesz)
        conn.sendall(content.encode('utf-8'))
```

"3"——请求服务器关闭当前打开的 TXT 文件。服务器的行为是调用 close()关闭文件指针，并向客户端发送消息。

```
elif rq[0] == '3': # 关闭当前打开的txt, 格式: '3'
    print("客户端关闭文档")
    fp.close()
    conn.sendall('已关闭文件'.encode('utf-8'))
```

这里可以看到，对 TXT 文件的支持函数都放在服务器端，客户端只负责发请求、接收内容，是一种“瘦客户机-胖服务器”模型；而我们的服务器对 PDF 的支持则是通过将 PDF 文件下发到客户端完成的，服务器只负责下发文件，对 PDF 文件的支持逻辑由客户机完成，是一种“胖客户机-瘦服务器”模型。

我们的服务器还实现了简单的异常处理机制，若产生异常，则将异常信息直接发送给客户机，然后继续处理下一个请求。这样就确保服务器出现异常时不会输出红字崩溃。

## # 循环处理当前连接发来的信息

```
while True:
    try:
        rq = conn.recv(buffsize).decode('utf-8')
        print('request:', rq)
```

以上是请求处理；在客户端调用 `close()` 关闭连接后，处理该连接的子线程也将退出，在退出前进行最后的清理工作，包括服务器端调用 `close()` 以双向关闭连接、关闭 TXT 文件指针（如果尚未关闭的话）、向运行信息中打印 "end connection [编号]"。然后子线程就调用 `_thread.exit_thread()` 退出。

以上就是服务器的工作流程和协议的分组格式。另外值得讨论的还有本服务器中应用的 **cookie 思想和握手协议**。

阅读器启动时，会向服务器请求下载一个包含书目信息的数据库，与本地缓存的数据库（如果有的话）合并，然后会在运行时按需根据本地的书目信息数据库向服务器请求下载相应的图书。这相当于一种不含过期机制的 cookie。相关细节将在“软件流程”部分的“维护元数据、下载封面”一节讨论。

在设计该服务器时，笔者曾思考过这样的问题：客户端调用 `close()` 是在本地单向关闭连接吗？如果是的话，应该如何保证服务器端也能随之关闭连接呢？为此笔者思考过写一个显式的握手协议来建立和关闭连接，但笔者查阅资料[2]得知，其实 socket 编程中的 `connect()`、`accept()`、`close()` 这些函数都已经原生包含了握手协议的过程。因此我们就不必实现显式的握手协议了。

## 2. FTP

为实现 PDF 支持，我们参照[3]实现了一个类 FTP 过程，来完成文件的传输。其工作流程如下。

当收到 "0 [文件名]" 请求时，服务器首先从请求字符串中截取文件名（第 2 位起至字符串末尾是文件名），然后用 `os.path.getsize()` 读取该文件的大小。这时会向运行信息中打印

```
# 获得文件名和文件大小
filename = rq[2:]
print("客户端请求文件", filename)
filesize = os.path.getsize(filename) # 文件大小，单位：字节
```

一行“客户端请求文件 [文件名]”。

获得文件大小以后，服务器将文件名和文件大小作为报头发送给客户端。报头的格式规定为：{'filename': [文件名], 'filesize': [文件大小]}（字典）。使用 json.dumps() 将字典转换成字符串（也可以用 eval() 进行转换，但使用 eval 有安全性问题[4]）。然后我们先发报头长度给客户端，再发报头。

```
# 发送报头给客户端，报头内容为文件名和文件大小
head_dic = {'filename': filename, 'filesize': filesize} # 报头（字典）
head_raw = json.dumps(head_dic) # 字典转换成字符串（当然也可以使用eval进行转换，
head_len_struct = struct.pack('i', len(head_raw)) # 报头的长度，转换为字符串
conn.sendall(head_len_struct) # 发送报头长度
conn.sendall(head_raw.encode('utf-8')) # 发送报头（字符串）
```

客户端接收报头长度字符串（因为报头长度往往不会超过 40 位，所以用 recv(40)），解码得到报头长度，用它作为参数去接收报头。

```
# 接收报头，得到文件大小
head_len_struct = self.tcp_sock.recv(40) # 报头长度的结构体
if not head_len_struct:
    return
print('[TCP]开始接收文件', filename + ('，保存名为 ' + savename if savename != filename else ''))
head_len = struct.unpack('i', head_len_struct)[0] # 报头长度
head_raw = self.tcp_sock.recv(head_len) # 报头的json编码
head = json.loads(head_raw.decode('utf-8')) # 报头
filesize = head['filesize'] # 文件大小
```

然后服务器端用 sendall() 发送所

有文件内容。但客户端因为有 buffer 大小限制，所以需分组接收，一次接收

```
# 发送文件内容
with open(filename, 'rb') as fp:
    data = fp.read()
    conn.sendall(data)
```

buffsize 个字节并写到本地，直至接收完毕。最后服务器和客户端再输出结束信息。

```
# 接收文件内容
recv_len = 0
start_time = time.time()
fp = open(savename, 'wb')
while recv_len < filesize:
    recv_cur = self.tcp_sock.recv(min(filesize - recv_len, self.buffsize))
    fp.write(recv_cur)
    recv_len += len(recv_cur)
fp.close()
```

我们可以看到，这一 FTP 过程体现了**分组传输**的思想。

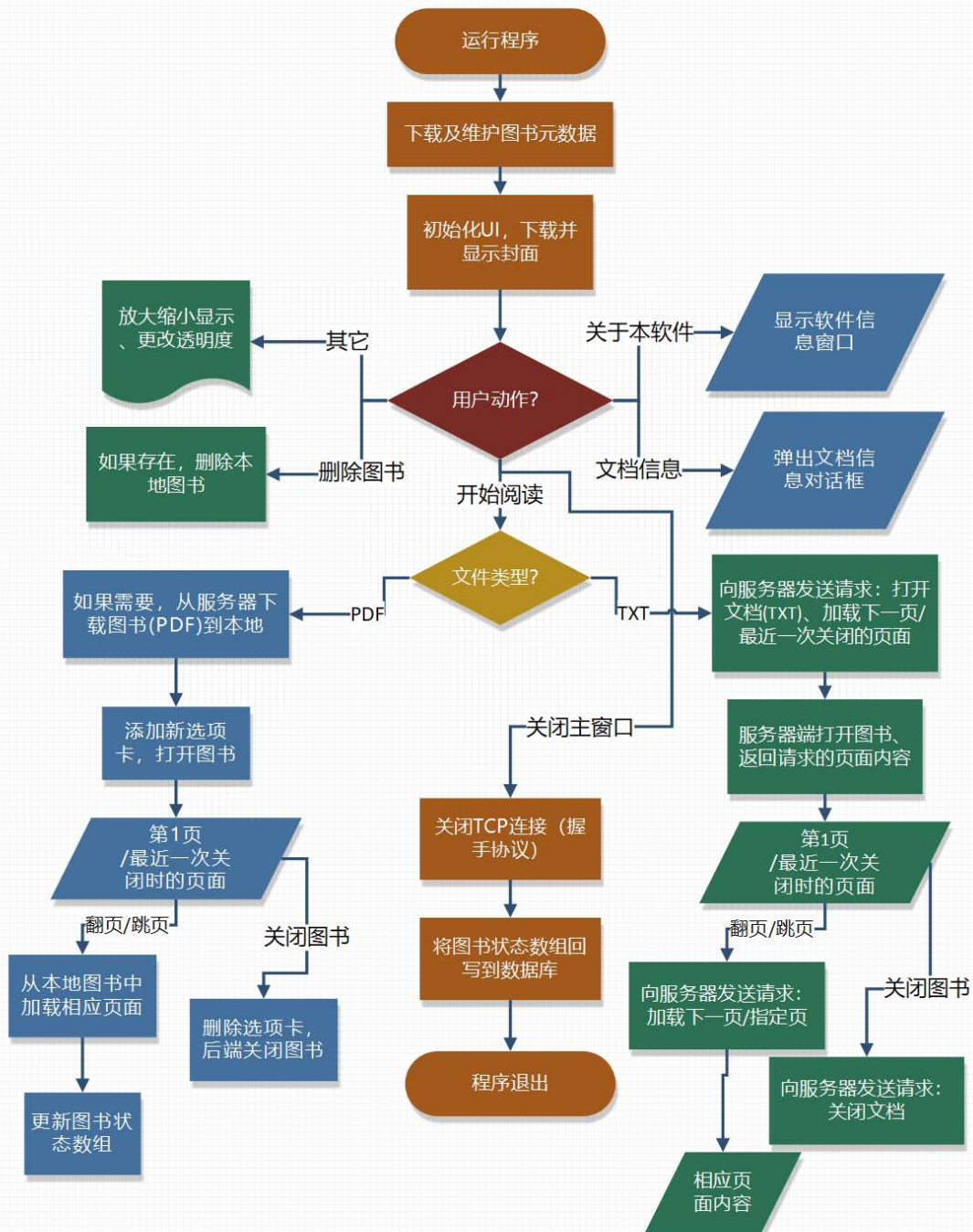
### 3. PDF 支持

前面已经说过，我们的阅读器阅读 PDF 图书时，将通过 FTP 请求从服务器端下载 PDF 文件到本地，在本地打开并处理。具体处理过程是调用 PyMuPDF 包，利用它的 open() 打开接收到的 PDF 文件，利用它的 loadPage() 完成翻页和跳页功能，该函数返回的 Page 对象还可以转换成 Qt（前端设计工具）的 QImage 对象，从而支持 PDF 缩放功能。最后再调用它的 close() 关闭 PDF 文件。因为不属于我们这门课程的主要内容，就不展开叙述了。



## 软件流程

















程序逻辑图





## 1. 启动服务器

在开始使用本阅读器前，先搭建 TCP 服务器。我们需要将相关的图书文件（支持 PDF 和 TXT 两种格式）和封面（与图书同名，扩展名为 jpg）与服务器代码放在同一文件夹下，并创建一个数据库 PDF.db，数据库中存储图书的文件名、已读到的页数、服务器端当前图书文件的 md5 值。

名称	修改日期	类型	大小
 Abraham-Silberschatz-Operating-Sys...	2019/12/17 21:07	JPG 文件	125 KB
 Abraham-Silberschatz-Operating-Sys...	2019/11/27 23:12	Adobe Acrobat ...	6,874 KB
 alice.jpg	2019/12/22 0:13	JPG 文件	39 KB
 alice.txt	2019/11/9 17:13	文本文档	149 KB
 Packet_Analysis_with_Wireshark.jpg	2019/12/17 21:07	JPG 文件	154 KB
 Packet_Analysis_with_Wireshark.pdf	2019/10/23 17:22	Adobe Acrobat ...	11,940 KB
 PDF.db	2019/12/23 20:12	Data Base File	12 KB
 RECIPE Converting Concurrent DRA...	2019/12/17 21:06	JPG 文件	173 KB
 RECIPE Converting Concurrent DRA...	2019/11/28 12:40	Adobe Acrobat ...	900 KB
 Speech and Language Processing 3r...	2019/12/17 21:07	JPG 文件	58 KB
 Speech and Language Processing 3r...	2019/12/12 10:14	Adobe Acrobat ...	18,323 KB
 tcp_server.py	2019/12/24 21:21	Python File	6 KB
 兼容ARM9的软核处理器设计——基于F...	2019/12/17 21:54	JPG 文件	38 KB
 兼容ARM9的软核处理器设计——基于F...	2019/11/29 22:05	文本文档	37 KB
 挽歌.jpg	2019/12/17 21:07	JPG 文件	206 KB
 挽歌.pdf	2019/12/16 19:21	Adobe Acrobat ...	819 KB

path	page	flag	md5
1 Abraham-Silberschatz-Operating-System-Con...	0	<null>	376e69c9ca2d90bd5748c3c16bda69
2 Packet_Analysis_with_Wireshark.pdf	0	<null>	a54793a1b4d008dae5b6f483ac52e5da
3 RECIPE Converting Concurrent DRAM Indexes..	0	<null>	3b1a1e1efa06c0a95209a43f8b8d7f9e
4 Speech and Language Processing 3rd ed. dr..	0	<null>	8c384dd81778828381dec83f811a1f50
5 挽歌.pdf	0	<null>	f62bcd76758d4e66ff83a7f175761ad
6 兼容ARM9的软核处理器设计——基于FPGA.txt	1	<null>	<null>
7 alice.txt	1	<null>	<null>

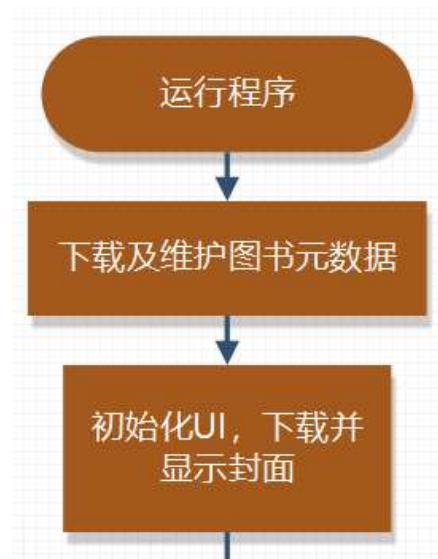
在准备好图书及封面文件和图书信息数据库后，我们启动服务器 tcp\_server.py。启动服务器之后，服务器就开始监听我们设定好的地址和端口，等待连接。

```
*Python 3.7.5 Shell*
File Edit Shell Debug Options Window Help
Python 3.7.5 (tags/v3.7.5:5c02a39a0b,
Oct 15 2019, 00:11:34) [MSC v.1916 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or
"license()" for more information.
>>>
= RESTART: C:\Users\jasha\Desktop\NetE
BookReader_server\tcp_server.py
Server start
Server is listening 127.0.0.1:8998
```

## 2. 维护元数据、下载封面

现在我们可以启动阅读器了。在阅读器启动之初，程序进行两项初始化工作：下载及维护图书元数据、下载并显示封面。（右侧是前面展示过的程序逻辑图的一部分，是程序刚启动时的初始化工作）

所谓的图书“元数据”指的就是前面提到的图书信息数据库。建立连接后，客户端立即向服务器发送请求"0 PDF.db"来从服务器下载 PDF.db 这个文件。我们可以从服务器的运行输出和阅读器的命令行输出中看到这一步骤，如下页顶端图片所示。



```
= RESTART: C:\Users\jasha\Desktop\l
_server.py
Server start
Server is listening 127.0.0.1:8998
begin connection 1
request: 0 PDF.db
客户端请求文件 PDF.db
发送文件完毕
```



在取得服务器端的 PDF.db 文件后 (本地暂存名为 PDF\_1.db), 阅读器后端会运行 SQL 语句将服务器端的 PDF.db (本地暂存名为 PDF\_1.db) 与本地缓存的 PDF.db 进行对比, 若发现服务器端的数据库有文件增删, 则本地数据库也作相应的改动; 若发现服务器端数据库中有文件的 md5 值相对本地有变化, 说明服务器端的图书文件已经更新, 则本地数据库的 md5 值也作相应的改动, 并删除该图书的缓存 (如果有的话)。下面的三张图片分别表示了服务器端数据库删除图书、增加图书、有图书 md5 值更新时的情况。(其中第三幅图片中的新 md5 值是我手动更改旧 md5 的最后一位得到的, 仅做展示用, 但与实际场景原理相同)

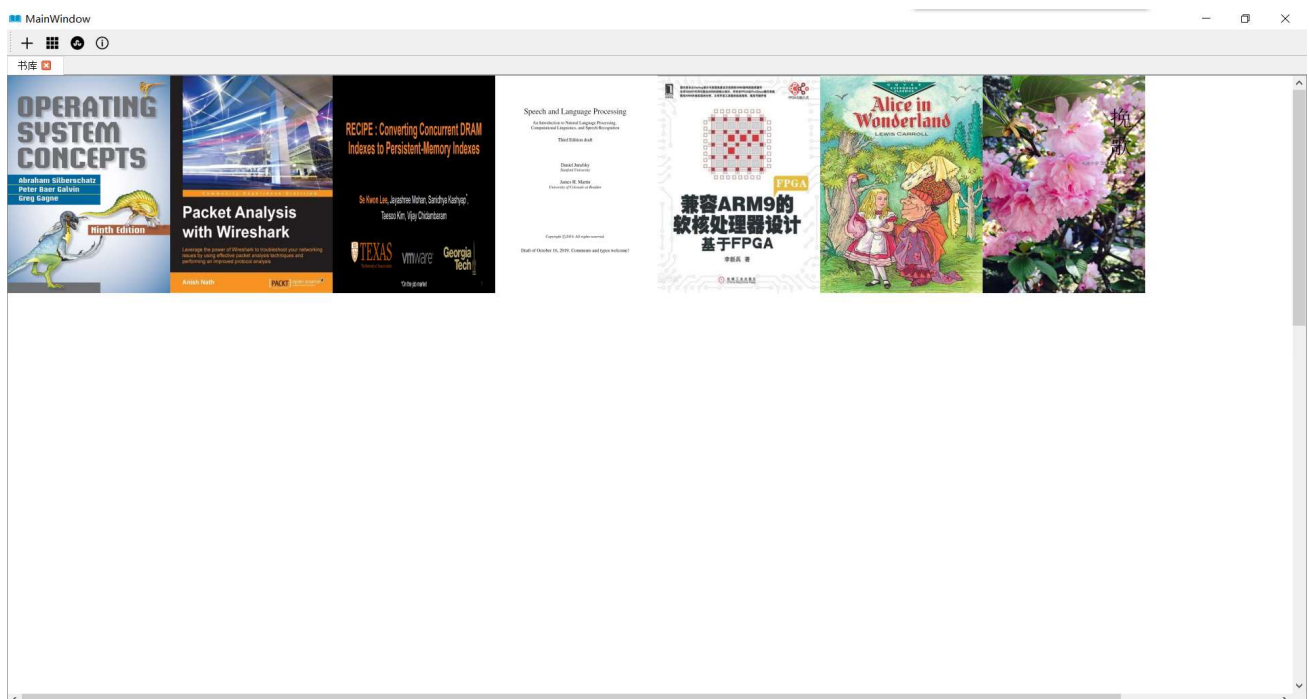




```
Run: _main_ x
C:\Users\jasha\AppData\Local\Programs\Python\Python37\python.exe "D:/Py
[Reader]从服务器端请求图书元数据.....
[TCP]开始接收文件 PDF.db, 保存名为 PDF_1.db
[TCP]接收长度: 12288 , 文件大小: 12288 , 总共用时0.0009968280792236328秒
[database]delete 挽歌.pdf md5: f62bcdb76758d4e66ff83a7f175761ad
[database]insert ('挽歌.pdf', 'f62bcdb76758d4e66ff83a7f175761ac')
[Reader]请求及更新元数据成功
```

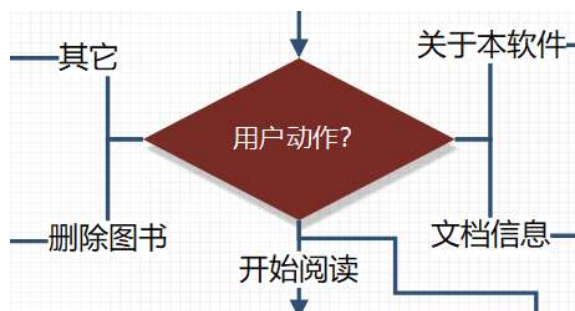
这样就维护了本地的图书信息数据库。此后阅读器再对数据库中的每本图书都向服务器发送"0 [图书名].jpg"下载封面，并显示在前端窗格中。

```
Run: _main_ x
[TCP]开始接收文件 RECIPE Converting Concurrent DRAM Indexes to Persistent Memory Indexes.jpg
[TCP]接收长度: 176738 , 文件大小: 176738 , 总共用时0.002990673065185547秒
[TCP]开始接收文件 Speech and Language Processing 3rd ed. draft 10-16-2019 (with note).jpg
[TCP]接收长度: 59242 , 文件大小: 59242 , 总共用时0.0008242130279541016秒
[TCP]开始接收文件 兼容ARM9的软核处理器设计—基于FPGA.jpg
[TCP]接收长度: 38879 , 文件大小: 38879 , 总共用时0.001993417739868164秒
[TCP]开始接收文件 alice.jpg
[TCP]接收长度: 39699 , 文件大小: 39699 , 总共用时0.0009851455688476562秒
[TCP]开始接收文件 挽歌.jpg
[TCP]接收长度: 210622 , 文件大小: 210622 , 总共用时0.0022361278533935547秒
[Reader]请求封面成功
```



### 3. 等待用户事件

GUI 的运行逻辑是，不断监听用户操作，用户的行为（包括鼠标点击、键盘按下等）会触发对应的事件（在本例中就是鼠标点击开始阅读、删除图书、文档信息、关于本软件等菜单），再启动对应的事件处理函数去处理它。因此我们的程序逻辑图的中心是等待用户动作。



### 4. 阅读 PDF

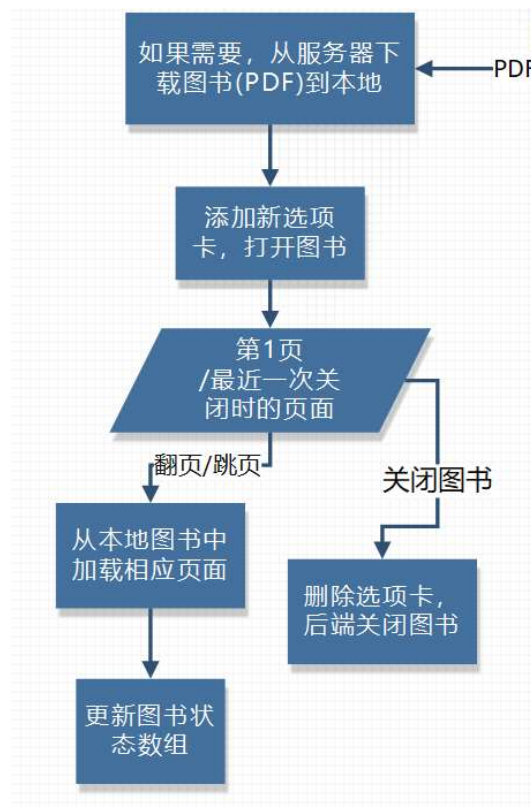
现在，我们点击阅读一本 PDF 图书《挽歌》。阅读器首先检查本地是否有该图书缓存，若无，则从服务器请求下载该图书，如下图所示，然后将打开该图书的第一页。

[Reader]从服务器端请求图书.....

[TCP]开始接收文件 挽歌.pdf

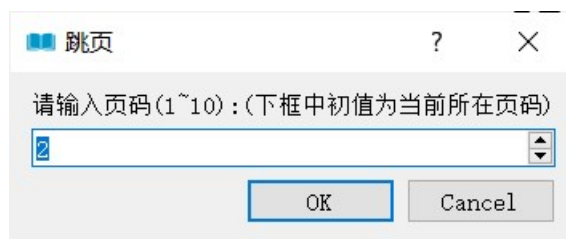
[TCP]接收长度：838311 ，文件大小：838311 ，总共用时0.01

[Reader]请求图书成功

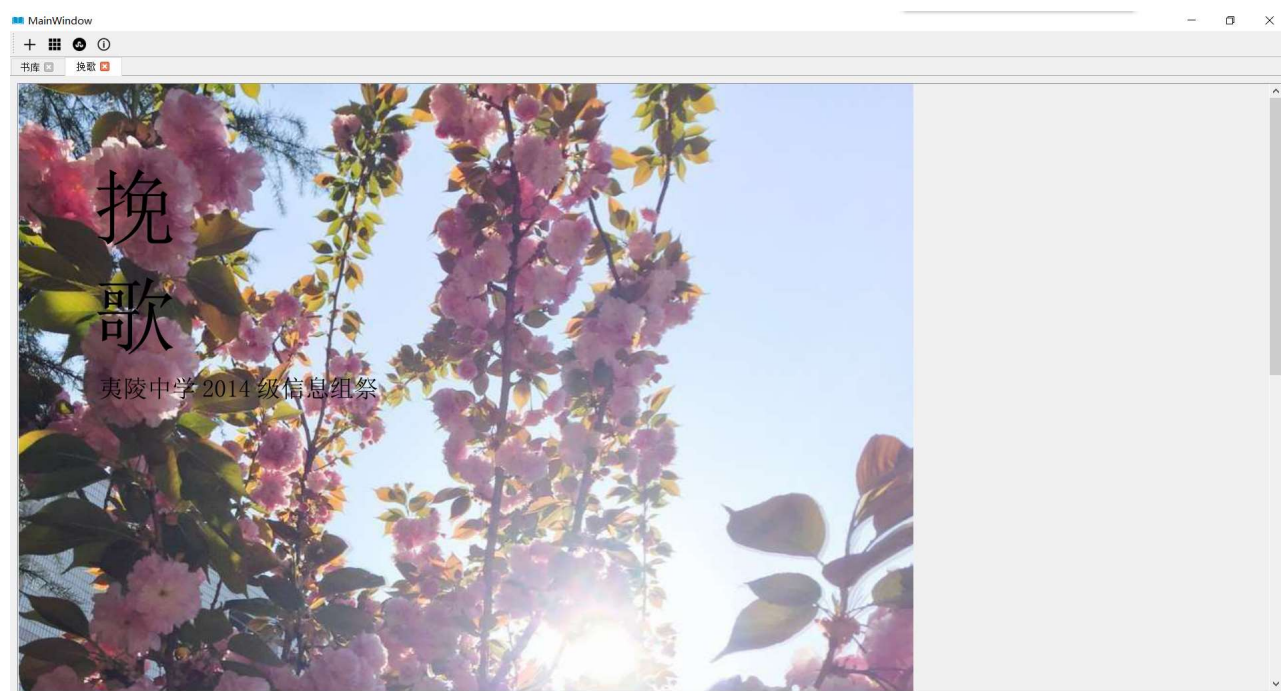




翻页的按键是左右方向键，也可以用鼠标点击屏幕的左三分之一区域或右三分之一区域。按下 Ctrl+G 将弹出跳页窗口，在此窗口我们也可以看到总页数（"1~10"）和当前所在页码（输入框中初值）。我们输入"10"并点击确定，



图书就跳到最后一页。值得一提的是，如果输入不在合法范围内，则阅读器什么都不做。



前端进行翻页和跳页动作时，一个图书状态数组也会记录图书当前所在页面。作为验证，我们可以关闭图书选项卡并重新打开，我们将看到上次关闭前最后阅读的页面。

## 5. 文档信息、删除图书

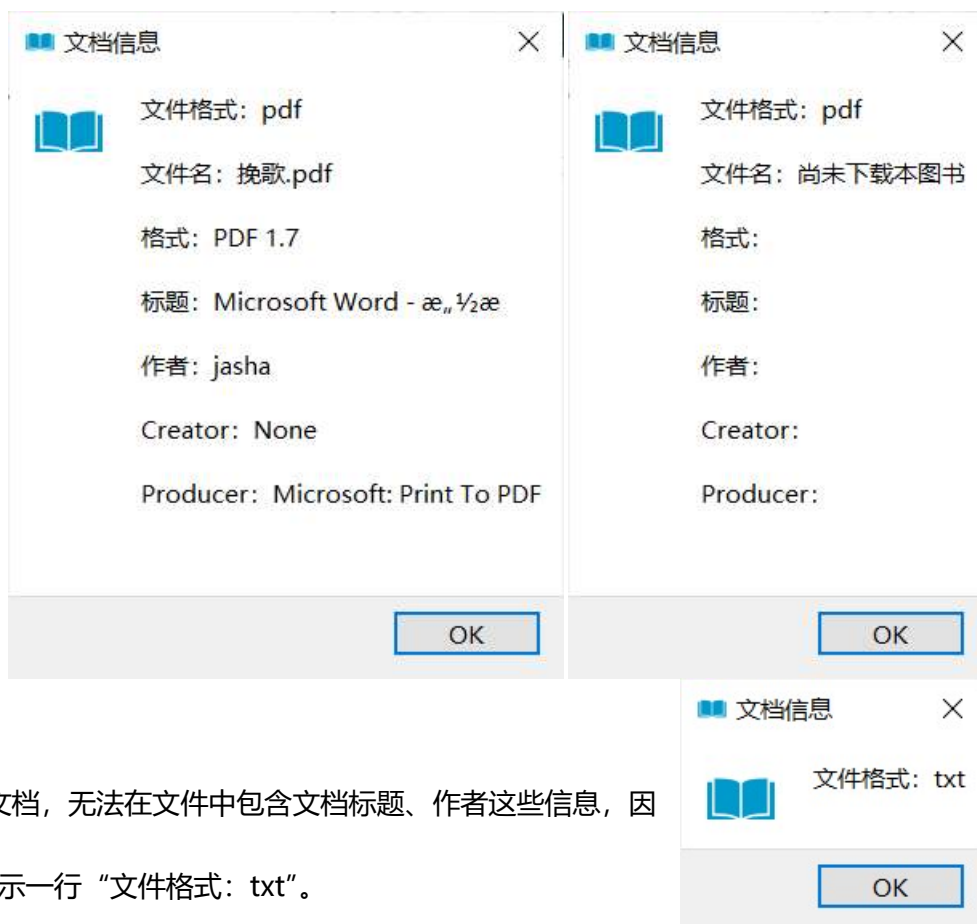


在主界面的图书封面上点击右键，我们将看到三项功能，如左图所示。其中“开始阅读”已经在上面展示过。接下来展示“文档信息”和“删除图书”。



对已下载到本地的 PDF 文件, 点击“文档信息”, 我们将看到该 PDF 文件中包含的文件名、PDF 格式版本、文档标题、作者等信息。对未下载的 PDF 文件, 则这些项都显示为“尚未下载本图书”。若是 TXT 文件,

因为 TXT 文件是纯文本文档, 无法在文件中包含文档标题、作者这些信息, 因此其文档信息窗口中只显示一行“文件格式: txt”。



“删除图书”将删除图书在本地的缓存 (如果有的话)。阅读器首先检查本地缓存是否存在, 若存在则将其删除, 否则什么都不做。作为验证, 我们可以对刚才下载的图书《挽歌》点击“删除图书”, 再打开“文档信息”, 将变成“尚未下载本图书”; 然后我们再点击“开始阅读”, 此时阅读器会再次从服务器下载本图书, 于是就又可以看得到它的文档信息了。

## 6. 阅读 TXT

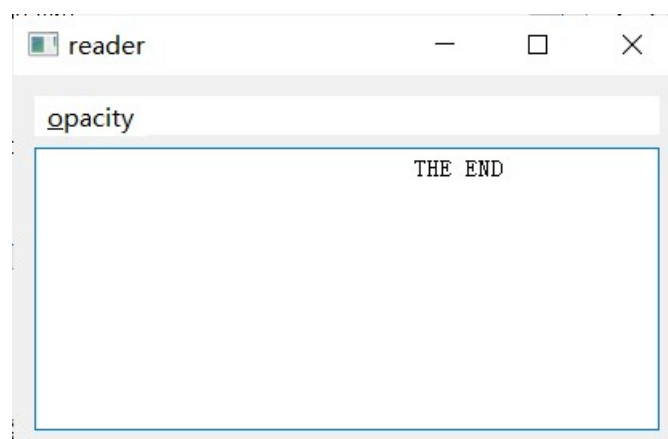
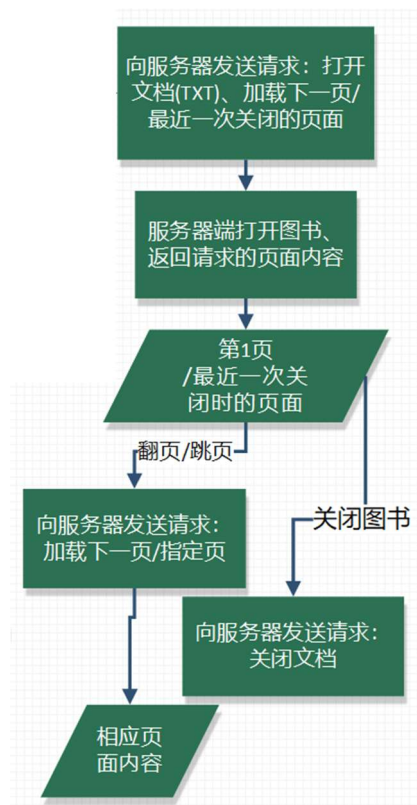
以上就是本阅读器阅读 PDF 文件的全流程。接下来，我们开始阅读文件格式为 TXT 的图书（以《Alice in Wonderland》为例）。

点击“开始阅读”后，阅读器后端将向服务器发送"1 Alice.txt"。我们可以从服务器端的运行输出中看到这一点。服务器使用前面介绍过的编码探测函数打开 alice.txt，并向客户端发送文件总页数。

然后阅读器将从图书状态数组中读取最近一次关闭该图书时所在页面，并向服务器请求"2 [最近一次关

闭时所在页]"，此处为 846。服务器返回第 846 页的内容，随后阅读器弹出 TXT 阅读器子窗口显示该内容。如果是从未打开过的图书，则发送"2 0"请求第一页；我们知道"2 0"的含义是请求下一页，此时文件指针刚刚打开，指向的是文件头，则"2 0"请求的就是第一页的内容。

```
request: 1 alice.txt
客户端打开文档 alice.txt
alice.txt
ascii
ascii
request: 2 846
```



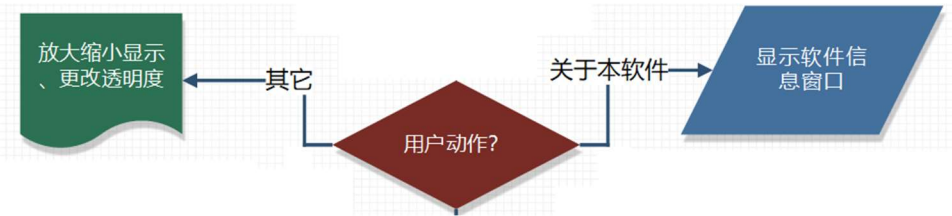
在该子窗口中，我们可以按下 Page Up / Page Down 键翻页，也可以按下 Ctrl+G 打开跳页对话框。此处我们随意按下若干次 PgUp/PgDn，服务器端的运行输出如右图所示。可以看出，对 TXT 文件翻下一页时，阅读器向服务器发送"2 0"；翻上一页时则通过"2 [上一页页码]"来进行。与此同时阅读器也会更新后端的图书状态数组。

```
request: 2 0
request: 2 0
request: 2 0
request: 2 0
request: 2 42
request: 2 41
request: 2 40
request: 2 39
request: 2 38
request: 2 37
request: 2 36
```

最后我们点击子窗口右上角的叉关闭之。服务器的运行输出如右图所示，由此可见，关闭 TXT 图书时，阅读器向服务器发送了一个"3"，符合我们在“重要模块描述”中所说的请求格式。

```
request: 3
客户端关闭文档
```

7. 其它功能

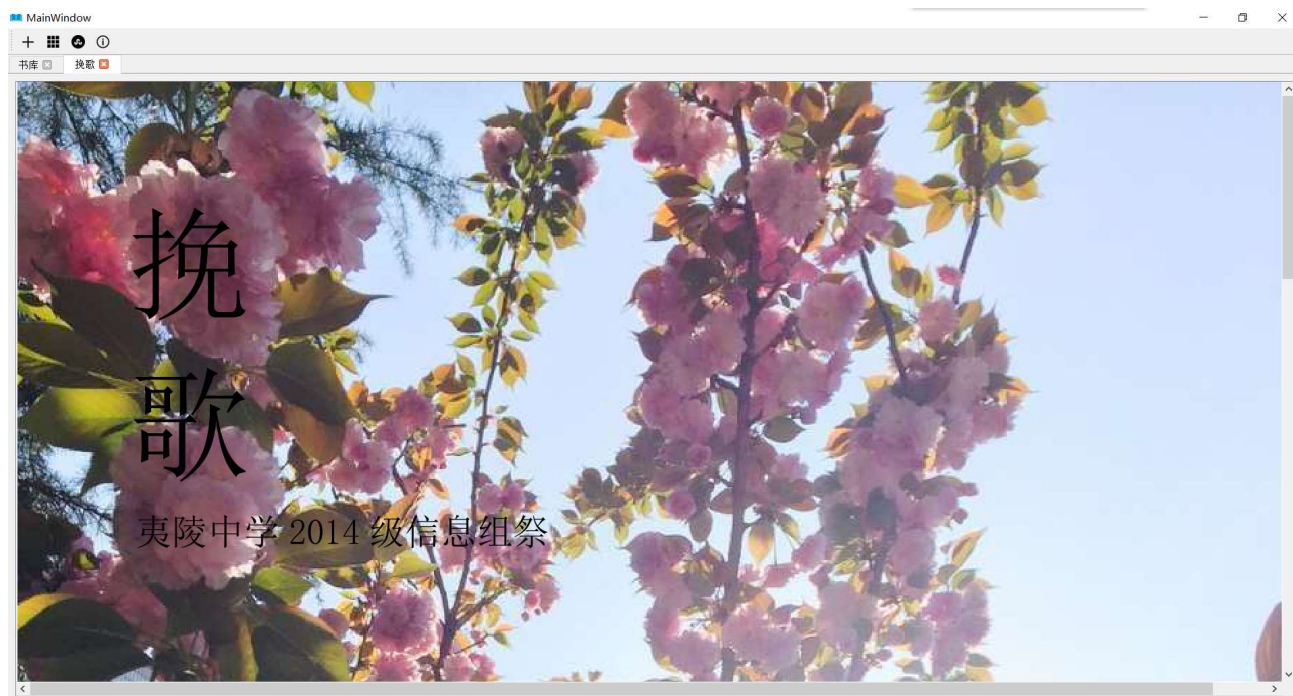


以上是本软件的主线，接下来展示本软件的一些附加功能。

更改透明度。TXT 阅读器支持更改透明度的功能，只需点击菜单栏的"Opacity"，并在 25%、50%、75%、100%四项中选择一项即可。其中 25%的效果如图所示。



放大/缩小显示。PDF 可以放大和缩小显示, 只需在阅读界面按下 Ctrl+"+"/"- "即可。下图是将《挽歌》放大显示三次后的显示效果。



“关于”窗口。点击主窗口工具栏的第四个图标, 可弹出“关于”子窗口, 内容如图。



## 8. 书签功能

最后介绍阅读器关闭时的流程和与之相关的书签功能。

当我们关闭主窗口时，后端将调用 `close()` 关闭 TCP 连接，根据[2]，`close()` 的执行过程是包含握手协议的，客户端会给服务器发送一个 FIN 分组，服务器端也就可以随之调用 `close()` 关闭连接了，这样就不会出现半开连接。

阅读器还会将前面提到的图书状态数组回写到数据库，从而可以实现书签功能，下次启动时再从数据库中读取图书状态数组（其中包含上次读到的页面的页码）即可。

作为验证，我们可以直接读取客户端的 `PDF.db` 这个文件，发现我们刚才阅读过的《挽歌》和《Alice in Wonderland》的页码已经发生了更改；我们还可以重新运行阅读器并打开相关图书，将回到上一次关闭阅读器时读到的页面。



Output		main.book_info	
<div>7 rows</div>			
path		page	
1	Abraham Silberchatz Operating System Concepts 5th2012.12.pdf		0
2	Packet_Analysis_with_Wireshark.pdf		0
3	RECIPE Converting Concurrent DRAM Indexes to Persistent Memory Index		0
4	Speech and Language Processing 3rd ed. draft 10-16-2019 (with note).		0
5	兼容ARM9的软核处理器设计—基于FPGA.txt		1
6	alice.txt		846
7	挽歌.pdf		9



## 总结

我们设计了一款支持 PDF 和 TXT 双格式的网络阅读器，实现了图书的打开、关闭、下载、翻页、跳页、书签功能。在 Socket 编程过程中，我们应用了多线程、胖客户机-瘦服务器、分组传输、cookie、握手协议等思想。除此之外的工作还有前端的调试、前后端的连接、数据库的配置等。

## 参考资料

[1] <https://www.jianshu.com/p/bfd8043e81f2>

PyQt5 从零开始制作 PDF 阅读器

[2] <https://www.cnblogs.com/cy568searchx/p/4211124.html>

Linux Socket 过程详细解释（包括三次握手建立连接，四次握手断开连接）

[3] <https://www.cnblogs.com/xiaokang01/p/9069048.html>

python socket 传输文件

[4] <https://www.cnblogs.com/OnlyDreams/p/7850920.html>

Python 如何将字符串转为字典