# 计算机网络 Project-小说阅读器

刘子牛

2019 年 12 月

## 目录

# 1 概述

本文为 2019 秋计算机网络课程 Socket 编程实验报告，我选择了小说阅读器作为实现对象。本节主要阐述任务要求、项目概述及文章框架。

## 1.1 任务要求

本任务的要求分为两个部分: 总体要求和具体项目要求。

- **总体要求：**设计一个具体的协议（建议是应用层协议），采用标准 Socket API 编程来实现协议的功能。
- **简单的小说阅读器设计：**
  1. 服务器端保存小说文本（txt 格式的即可）
  2. 客户可以打开对应的文本，翻页，翻章，跳页，书签，下载，关闭等
  3. 建议最好有图形界面，因为是 txt 格式，所谓的“页”可以通过规定每次内容包含的字节来规定

## 1.2 项目特点

本小说阅读器使用 Python 编写，图形界面使用 Tkinter 库，信息传输依赖标准的 Socket API。提供的功能有：用户注册、用户登录、小说阅读、小说下载。对所有使用 Socket 传输的信息都实现了加密，保证信息安全。在服务器端采用多线程设计，可长时间运行并支持同时支持多个客户端。

## 1.3 文章结构

本文第二节会说明小说阅读器的实现细节，包含传输协议（及加密算法）、服务器、客户端、重点功能实现和错误处理五个部分，第三部分则展示小说阅读器的最终运行效果，最后一部分进行总结，阐述还可以改进的地方以及本次实验的收获。

# 2 实现细节

本节会详细说明小说阅读器实现各个部分的关键点。

## 2.1 传输协议

由于传输协议为客户端与服务器通用，所以我将所有协议内容放在单独的文件夹“protocol”下。鉴于小说阅读器自身的特点——既需要实现消息传输（登陆、发送一页等），又需要实现文件传输（小说下载功能），我针对这两个不同的需求分别定义了发送协议。

### 2.1.1 消息传输

由于 Socket API 只支持 byte 型数据传输，所以在发送时先要将各种数据类型转换为 byte 类型。Python 已经为我们提供了许多转换的函数，但是由于各个类型的转换函数不同，使用起来不太方便。因此我写了两个**通用转换函数**（在 protocol/-data_conversion 文件夹下），负责各个类型与 byte 类型之间的相互转换，其首先识别原数据类型，之后再调用相应的函数进行转换，给下面的编程带来了方便。

对每条消息我共做了三**层封装**，第一层封装的作用是将各种类型的**数据转化为 byte 类型**，正如前文所述，为了做到统一的数据转换，传输时需要指明原始数据类型及转换后 byte 数据的长度。第一层封装后数据格式如下：

| Type of Data (1 Byte) | Length of Data (4 Bytes) | Data |
|---|---|---|

第二层封装的作用是**组合消息头与消息体**。类似于 HTTP 协议，每条发送的消息都可以分为两个部分：消息种类（消息头）和消息内容（消息体）。为了方便识别消息头，我给每个种类的消息定义了一个“int”型编号（定义在 message_type.py 文件中），映射关系如下。发送消息时转换为数字，接收消息时再转换回消息类型。

```
class MessageType(enum.IntEnum):
    """里面规定了各个信息类型对应的标号以及处理指令对应的标号"""
    # === Client Action 1-100
    login = 1
    register = 2
    require_list = 3
    download = 4
    start_read = 5
    require_page = 6
    update_bookmark = 7

    # === Server Action 101-200
    login_successful = 101
    register_successful = 102
    book_list = 103
    file_size = 104
```

```
    send_page = 105
    send_chapter = 106 # 发送章节列表
    page_num = 107
    total_page = 108

    # === Failure 201-300
    login_failed = 201 # 登陆失败
    username_taken = 202 # 用户名被占用
    no_book = 203 # 查无此书
```

经过第二层封装后消息的格式如下。注意，由于消息头的长度和类型固定，所以无需使用第一层封装，而消息体则已经经过第一层封装。

| Message Type (4 Byte) | Message Body |
|---|---|

第三层封装为**加密封装**，将第二层输出的结果作为一个整体加密。根据 AES 加密的规则（见2.1.3节），除了加密后的数据外，发送方还需要告知接收方消息总长度、填充位数以及初始化向量（Initialization Vector，IV）。我们将这三个信息依次封装在消息头部，最后是经过 AES 加密的第二层封装数据：

| Length of Message (4 Bytes) | Length of Padding (1 Byte) | AES IV (16 Bytes) | Message |
|---|---|---|---|

由于消息格式的统一，所以在服务器和客户端可以采用相同的函数来接收所有类型的消息。服务器在开启之后便一直处于监听状态，运行着接收消息的函数。而客户端则在有需要的时候运行，接收想要接收的消息。

### 2.1.2 文件传输

文件传输相比于普通的消息传输有其自身的特殊性：

- 小说文件（txt 格式）本身可以用二进制方式读取直接得到 byte 格式的信息，无需调用函数转化。
- 小说的传输本身有连续性。一本小说需要分多次传输，而每次传输的长度和格式都相同（最后一次长度不同）。

所以对于文件传输只需要一层加密封装即可，加密后的数据格式为：

| Length of Padding (1 Byte) | AES IV (16 Bytes) | One File's Part |
|---|---|---|

我们在发送的数据里并没有包含长度，那么客户端该如何进行接收和解密呢？其实这涉及到文件发送的整体流程。在发送时，首先通过消息发送文件的总大小，再循

环发送文件内容，这里我们规定每次发送的大小为定值（最后一次发送剩余值）；接收时则相应先接受文件大小，然后客户端自动算出每次应接受的文件的大小，循环接收。

### 2.1.3 加密算法

加密是传输协议的重点。我通过设计“secure_channel”类（文件 protocol/secure_transmission/secure_channel.py 中定义）构建了一个数据传输的 “安全通道”，上文所述的加密与解密、发送与接收其实都是定义在该类里的函数。

服务器持续运行，当一个新的客户端建立时，第一件要做的事便是与服务器建立安全通道。这个建立过程包括两步：(I) 客户端与服务器交换密钥 (II) 二者分别计算出共同密钥。建立完成之后，客户端与服务器各自保存密钥，接下来所有的信息交换都将依赖该安全通道传输。

在加密协议方面我选择了比较流行的 **AES 的 CBC 加密**，这是一种加密后长度不变的算法，不过需要被加密信息的长度为 16 的整数倍（因此才需要前文中的填充位）。加密算法如图1。可以看到，该算法无论是加密还是解密都需要初始化向量和密钥两个信息，密钥在建立安全通道时计算出，之后就不再变化，初始化向量则每条消息都不相同，作为加密消息的头部一起发送。

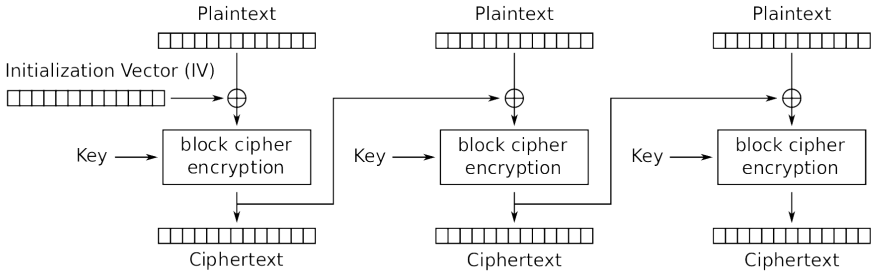


图 1: AES 的 CBC 加密过程图

## 2.2 服务器

服务器端的内容存放在 server 文件夹中。根据设计思路，所有的小说都应存放在服务器端，我把它们放在 server/books 文件夹下。服务器代码最主要的功能就是不断监听从客户端发来的请求，然后针对该请求发送相应的回复。为了能让服务器同时支持多个客户端并长时间运行，我采用了**多线程**的设计，每当客户端发来一个信息时便给它分配一个线程，使用该线程进行读取。

我们把每种请求称为一个事件(event)，而处理这些请求的函数统一放到 server/event 文件夹中。根据前文对 “消息” 的定义，一条消息的请求可以通过其消息头的代码来

得知，所以我在 server/event/___init.py___ 文件中定义了一个事件集中处理函数，其作用是根据消息头所属的类型分配对应的函数。

## 2.3 客户端

客户端的所有代码保存在 client 文件夹中。与服务器不同，客户端无需写事件处理函数，它只要发送请求然后接收服务器反馈即可，因而代码结构也比较简单。GUI 界面的代码全部在 client/forms 文件夹下。此外，我还用 memory.py 文件保存当前客户端的一些必要信息，比如安全通道的信息，登陆用户的信息等等。

## 2.4 功能实现

上面笼统介绍了小说阅读器代码的组织结构，接下来我将就一些比较关键和复杂功能的实现进行说明。

### 2.4.1 页和章

小说阅读器最重要的就是阅读体验，根据要求，书籍应该保存在服务器中，客户端申请一页服务器发送一页的内容；除此之外还应实现翻页、跳页，翻章、跳章的功能。也就是说我们对于一本书一定要有固定的定义页的方式，内容与页数对应关系是不变的。如果仅仅分页的话这很容易，只需要固定每次发送字符串的长度即可。但是如果还想分章的话就不是那么容易了，我们需要在每章的开头新起一页，这必然涉及到对章节的识别。经过思考，我决定采用在每章前加标记的方法，读入书籍时逐行阅读，如果某一行的开头为“#”，那么这行就是章节标题，应在此处新起一页。这样的设计也使获得页数相对复杂，下面是获得总页数的算法示例：

```
with open('./server/books/' + bkname + '.txt', 'r', encoding='utf-8')
    as f:
    if f.readline() == 'C\n':
        page_words = C_ONE_PAGE_WORDS
    else:
        page_words = E_ONE_PAGE_WORDS
    line = f.readline()
    while line:
        s = ''
        s += line
        line = f.readline()
        while line:
            if line[0] == '#':
                break
```

```
            s += line
            line = f.readline()
        total_page += math.ceil(len(s) / page_words)
        chapter.append([line[1:-1], total_page])
sc.send_message(MessageType.total_page, total_page-1) # 发送总页数
sc.send_message(MessageType.send_chapter, chapter[:-1]) # 发送章数列表
```

### 2.4.2 书签

书签功能的意思是用户如果阅读过一本小说，那么当其再次打开该小说时，服务器发送的应该是其关闭时的那页。为了实现这个功能，我们需要为每个用户维护一个列表，这个列表中存放了小说名和其上次阅读的页数。由于用户信息保存在 server/users.txt 文件中，每个用户占文件中的一行，所以我设计将小说名和书签页数直接保存在用户信息后面，每个信息用“|”分隔，保存格式如下：

```
Jack|123456|Little Prince|27|老残游记|0|万历十五年|110
```

每次打开书籍，首先在该文件中查找是否存在书签，如果存在则发送书签页，如果不存在则创建新书签，页数置为 0。

```
# 查找书签
page_num = 0 # 初始化书签为0
with open('./server/users.txt', 'r', encoding='utf-8') as f:
    users = f.read().splitlines() # 转化为列表
    for user in users:
        user = user.split('|')
        if user[0] == user_name: # 找到该用户
            index = user.index(bkname) if (bkname in user) else -1
            if index != -1: # 找到该书的书签
                page_num = int(user[index+1])
            break
```

我将书签更新设定在阅读窗口关闭的时候，当点击关闭按钮时，客户端发送当前的页数，服务器接收后作为书签页数保存。

### 2.4.3 中英文支持

如前文所述，我们通过设定 Python 自带的 read 函数读取的大小来定义“一页”的字数。为了保证中英文图书全面支持，这两种图书统一使用 UTF-8 格式保存。然而这就带来了一个问题：中英文对 UTF-8 格式的编码规则不同，通过实验发现，读取相同的位数时中文的字数要多于英文一倍以上，这将导致中文刚好填满阅读器的一页时英文却只有不到半页。

为了解决这个问题，只能中英文使用不同的读取位数。为了区分当前书的语言，我在每本书的第一行单独使用一个英文字母来表示该书语言，“C”表示中文，“E”表示英文。服务器会首先识别语言再计算页数。

## 2.5 错误处理机制

本程序设计了一些错误处理机制以应对例外情况，大部分例外情况会通过弹窗通知用户。对本小说阅读器来说，由于功能相对简单，可能发生错误的地方主要在于（I）登陆和注册（II）客户端书架（书籍列表）界面内容与服务器不匹配（III）页面跳转时超过了上下界限（IV）服务器发生了其他错误，对此我分别写了相应的错误处理机制。

**登陆与注册：**登陆时可能发生用户名或密码输入错误的情况，注册时可能发生用户名已被占用或两次输入密码不一致的情况，这里我们展示当用户名被占用时服务器的检测和处理代码（运行示例如图4）：

```
with open('./server/users.txt', 'r', encoding='utf-8') as f:
    users = f.read().splitlines()
    for user in users:
        user = user.split('|')
        if parameters[0] == user[0]: # 用户名已被占用
            sc.send_message(MessageType.username_taken)
            return
```

**书架与服务器不匹配：**这种情况会发生在客户端书架已接收书籍列表，之后服务器删除某一书籍，而书架尚未更新导致的错误。为此我定义了消息类型“查无此书”，并为书架界面设定了刷新按钮，可以再次申请获得书籍列表。运行展示中有这种意外检测的情况，运行效果见图6，检测代码如下：

```
# 检查该书是否在服务器中
bklist = os.listdir('./server/books')
for i in range(len(bklist)):
    bklist[i] = bklist[i].strip('.txt')
if bkname not in bklist: # 如果这本书不在书籍列表里
    sc.send_message(MessageType.no_book)
    return
```

**页面跳转超过界限：**输入页面需要新的窗口，我使用了 Tkinter 库自带的简单对话框以输入页码，在其中可以方便的设置上下界，这里不再赘述。运行效果如图9。

**服务器发生其他错误：**除了以上情况外，服务器还可能发生一些其他错误，对于这种错误我们也应在客户端予以提示。经设定服务器如果不能正常处理请求会发送一条错误代码，所以我们可以弹出错误窗口并打印该错误代码，这里以接收书签所在页数为例，代码如下：

```
# 接收书签所处页数
message = self.sc.recv_message()
if message['type'] == MessageType.page_num:
    self.page_num = message['parameters']
    print('《{}》书签位于第{}页'.format(self.bkname, message['
        parameters']))
elif message['type'] == MessageType.no_book:
    messagebox.showerror('请求失败', '查无此书，请返回刷新书籍列表！')
    return
else:
    print('未能成功接收到书签页数！错误：{}'.format(message['type']))
    messagebox.showerror('请求失败', '未能成功接收到书签页数！错误：{}'
        .format(message['type']))
    return
```

# 3 运行展示

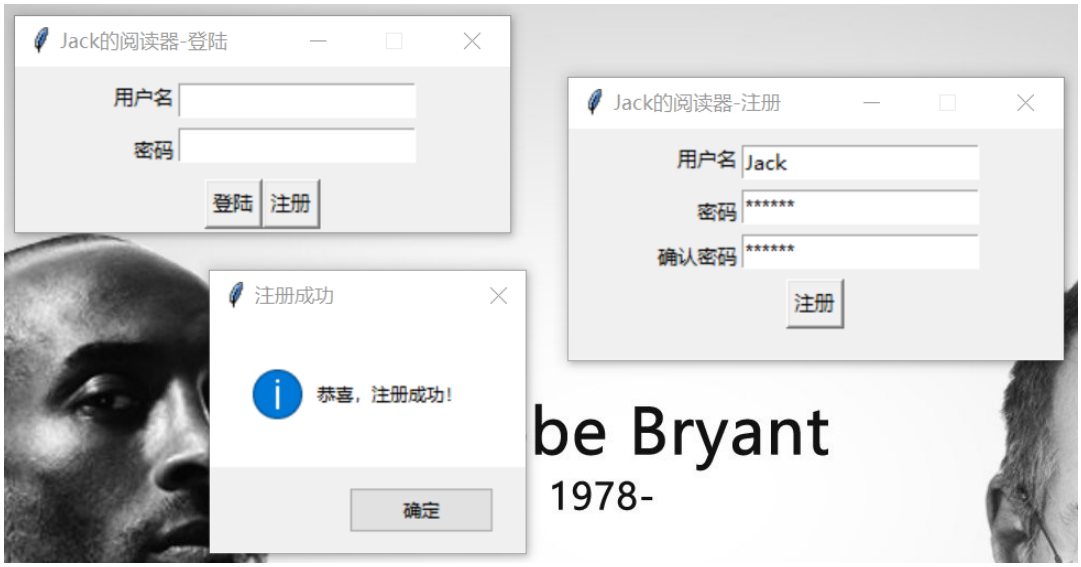


图 2: 登陆与注册

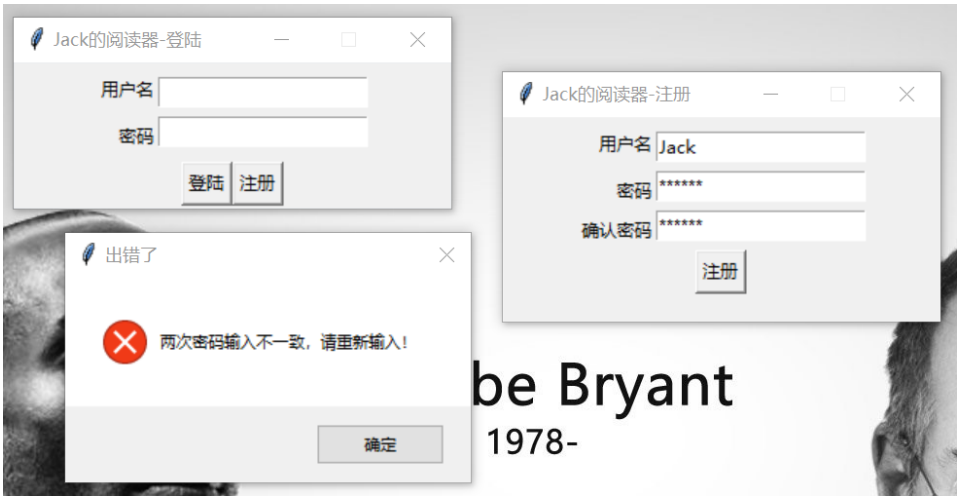


图 3: 错误处理：注册时输入的两次密码不一致

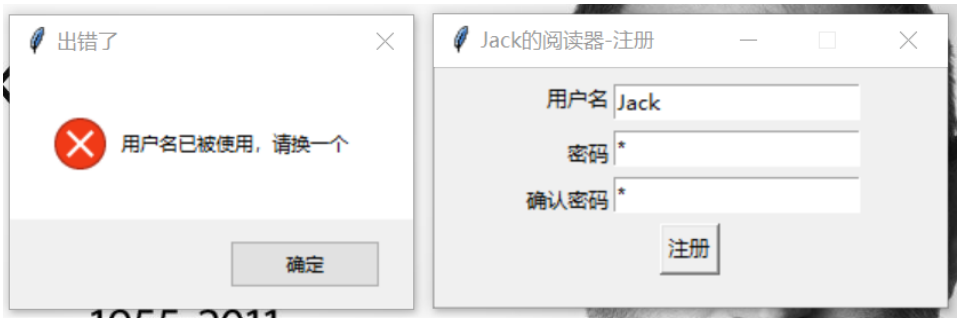


图 4: 错误处理：用户名已存在


图 5: 书架（书籍列表）

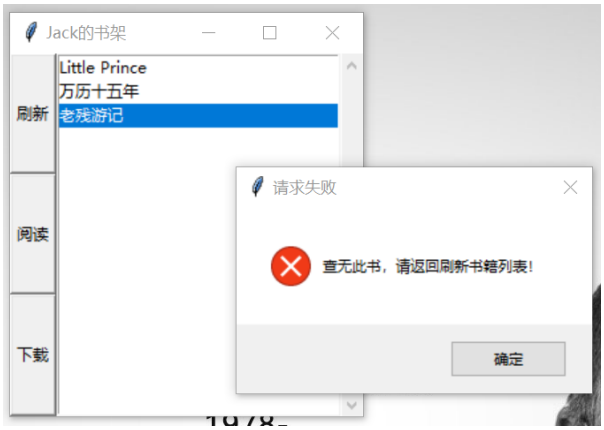


图 6: 错误处理：服务器无对应书籍

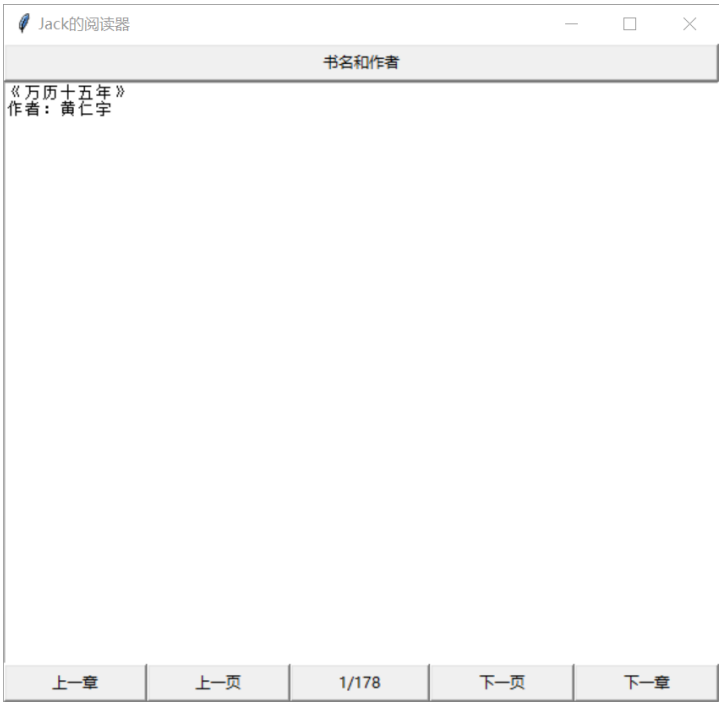


图 7: 阅读书籍第一页

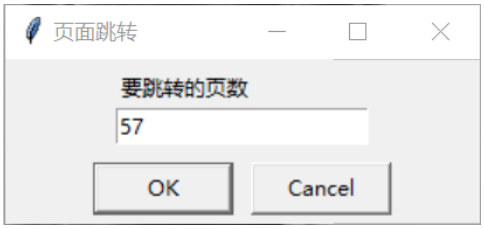


图 8: 页面跳转（点击底部中间的页码按钮即可选择页数跳转）

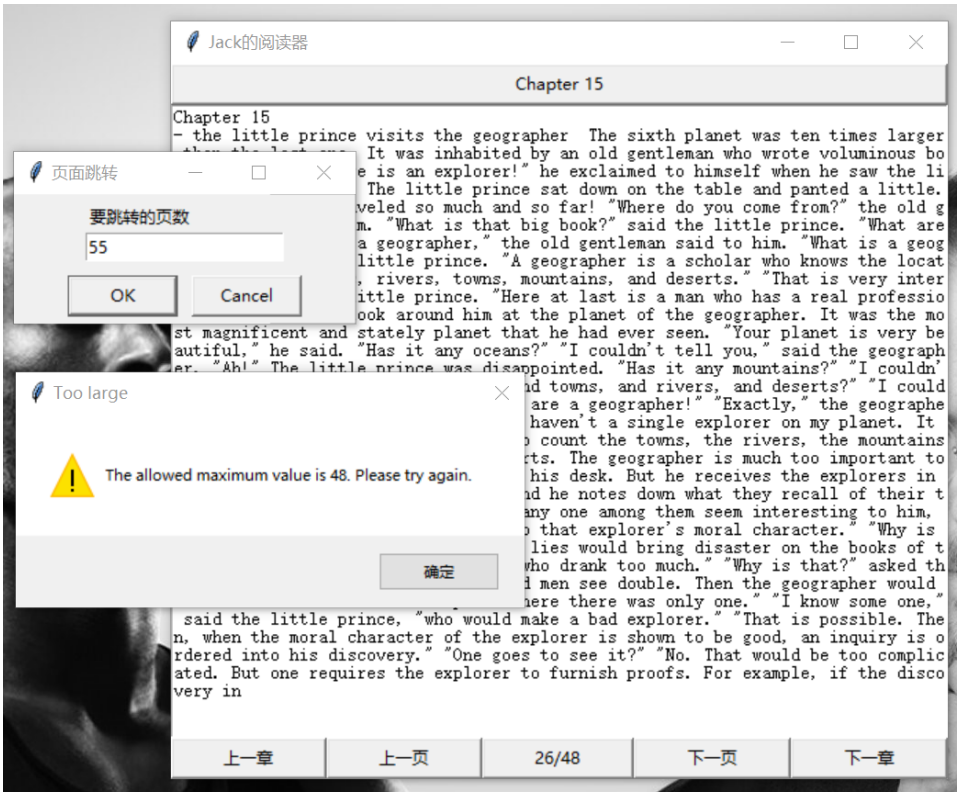


图 9: 错误处理：页面跳转超出范围

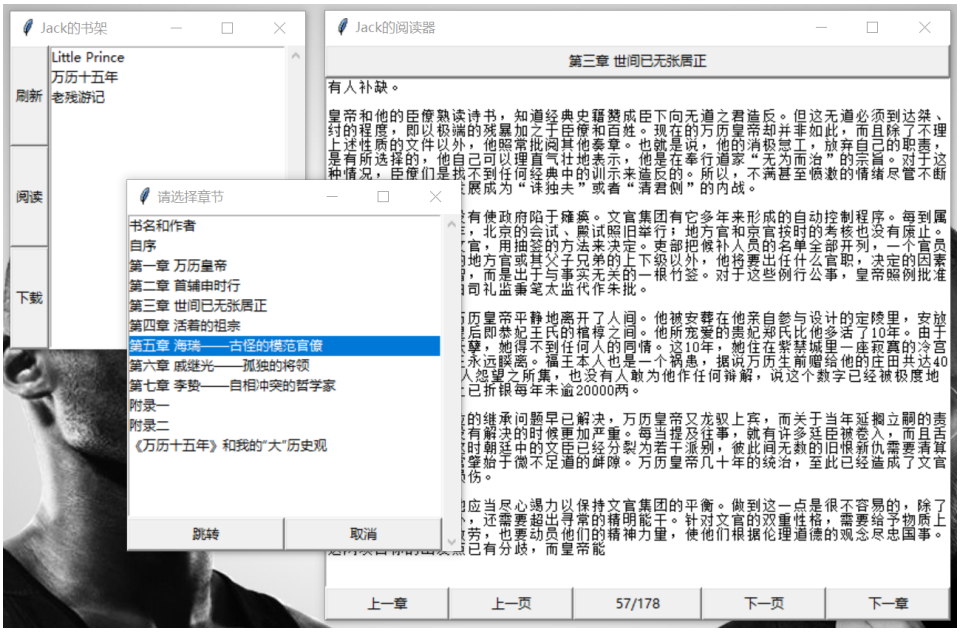


图 10: 章节跳转（点击页面上方章节栏即可弹出此界面）

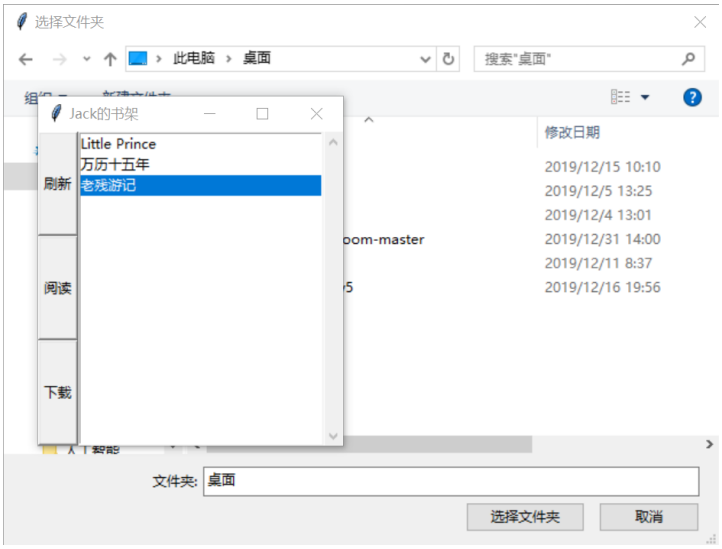


图 11: 下载书籍，下载时支持自由选择路径

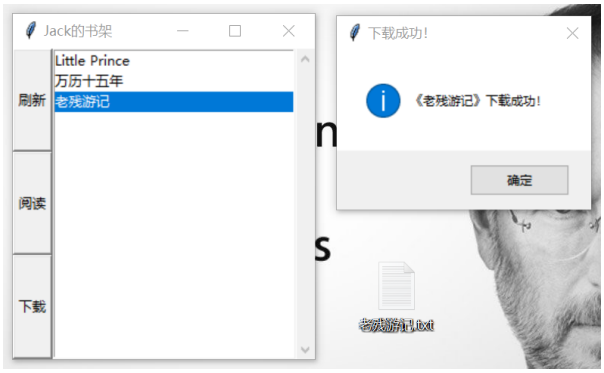


图 12: 下载成功

# 4 总结

本次编程实验的领域比较陌生，从零开始一步步做出完整的程序让我收获了许多。在这一过程中，我不仅掌握了了 Socket API 的用法，对其他如 Tkinter GUI 编程、Python 文件管理、加密算法等知识都有所了解。

但是整个开发经历还是有些遗憾的，如一开始规划的书籍评论功能最终因为时间的原因没能实现，在错误检测方面做的也还有漏洞。关于本小说阅读器还有许多工作要做，希望将来有时间可以逐一完善。

由于本项目的开发时间比较长和零散，所以查阅的一些参考资料没能一一记录，**在此一并致谢**。项目代码已开源在我的Github 仓库，整个项目详细的文件结构说明可以在 README.md 中看到。