搜索引擎技术基础技术报告

目录

一、 概述

互联网信息复杂多样，因此想要迅速、快捷的找到所需要的信息内容，就需要搜索引擎来帮忙实现。

第一个现在意义上的搜索引擎是 1994 年 Michael Mauldin 创建的 Lycos。是在网络爬虫的基础上加上索引程序，采用网络、数据库等关键技术来实现，检索速度也非常慢。

第二个阶段是 1996 年到 1998 年,这个期间,搜索引擎采用分布式检索方案,使用多个微型计算机来协同工作,其目的是为了提高数据规模和响应速度。一般可以响应千万次的用户检索请求。第三代搜索引擎,就当前所使用的搜索引擎,也是搜索引擎极为繁荣的时期。它拥有完整的索引数据库，除了一般的搜索，还有主题搜索和地域搜索。

本系统以西北工业大学的工大要闻为目标，使用 python 简单实现了搜索引擎的三个部分：网页搜集、对搜集网页的预处理、查询服务。并能满足其基本要求：在可以接受的时间内，输入一个查询词，输出符合条件的网页列表。

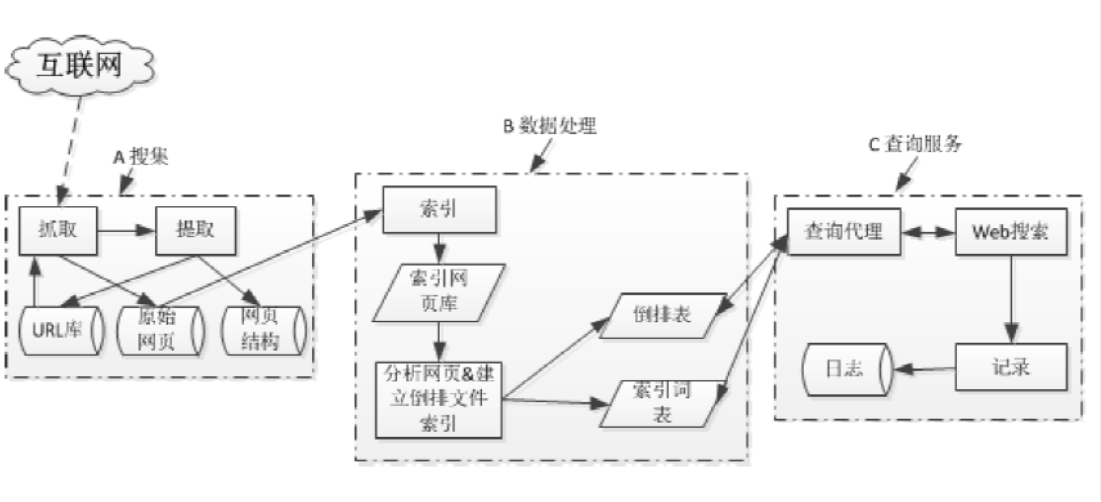

图 1 搜索引擎简单体系结构

项目地址：https://github.com/chen2511/SimpleSearchEngine

二、 页面获取与解析

2.1 理论基础

搜索引擎的页面获取流程如下：

1、首先选取一部分精心挑选的种子 URL，将这些 URL 放入待抓取 URL 队列

2、从待抓取 URL 队列中取出待抓取在 URL

3、解析 DNS，并且得到主机的 ip

456、将 URL 对应的网页下载下来，存储进已下载网页库中

79、分析页面中的其他 URL，并且将 URL 放入待抓取 URL 队列，从而进入下一个循环

8、将这些 URL 放进已抓取 URL 队列

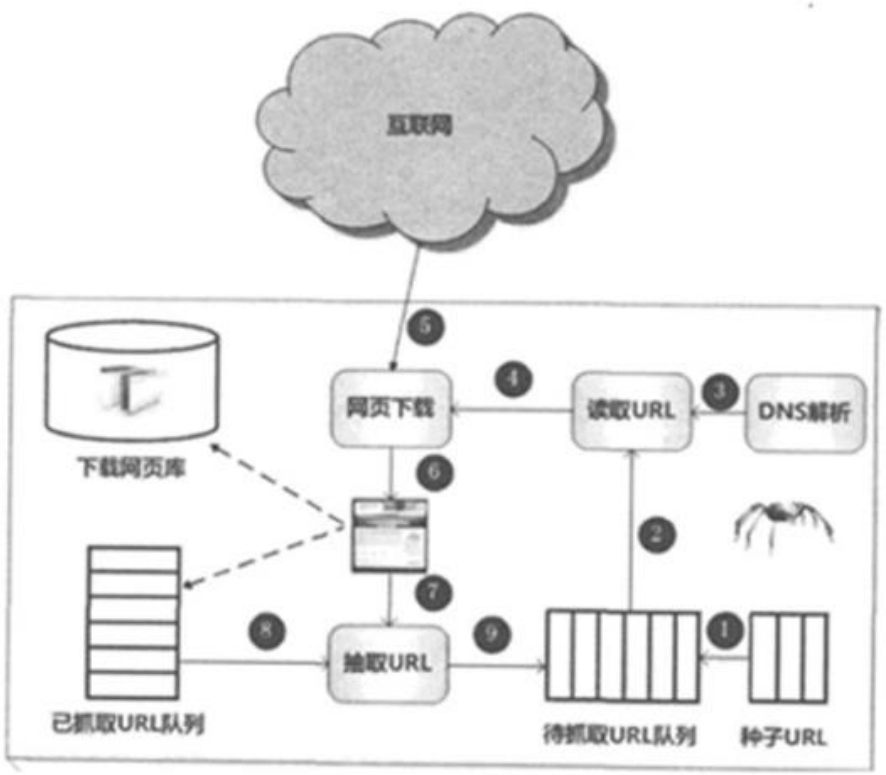

图 2 搜索引擎页面获取流程图

2.2 实践

相关代码位于：spider.py

本程序的搜集流程：挑选工大要闻的某些新闻列表页，作为种子；之后获取种子 url 的页面，分析其中存在的 url（新闻）并加入待爬取列表（考虑不爬取过多页面，只爬取两级），然后再爬取列表里的所有 url，之后再进行保存等。

图 3 种子 url 页面举例

首先是页面内容获取，本程序借助的是模拟浏览器。下载 chrome 相关驱动程序，配合 python `selenium` 包，即可完成页面内容获取。
部分代码如下：

```
# 模拟浏览器，使用谷歌浏览器，将 chromedriver.exe 复制到谷歌浏览器的文件夹内
chromedriver = r"C:\Program Files (x86)\Google\Chrome\Application\chromedriver.exe"
# 设置浏览器
os.environ["webdriver.chrome.driver"] = chromedriver
browser = webdriver.Chrome(chromedriver)
browser.get(url_seed)
```

这样就获得了相应 url_seed 的原始网页。

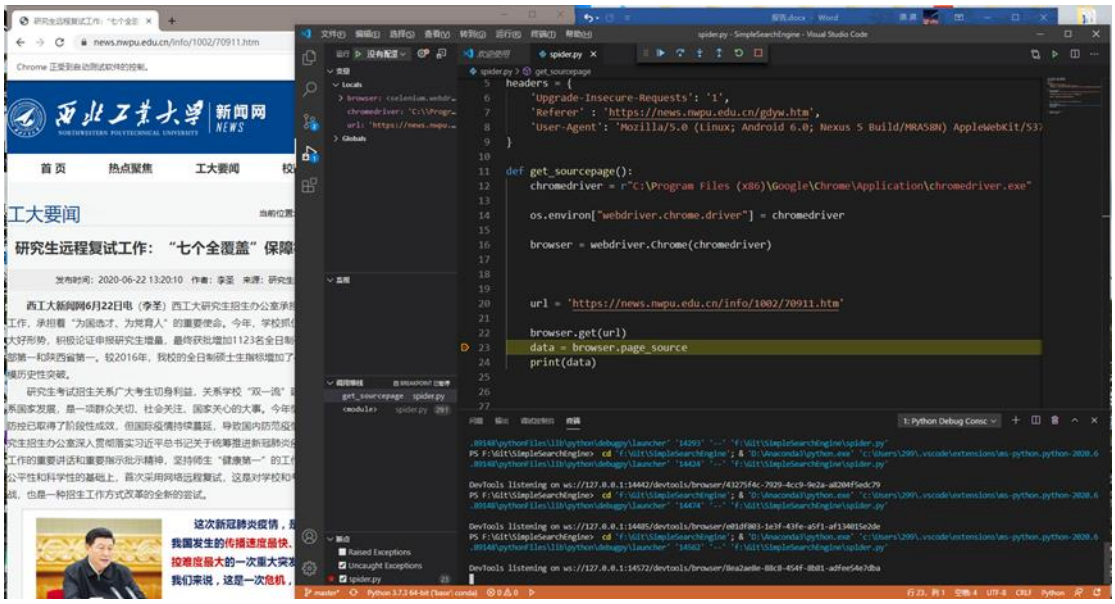

图 4 浏览器模拟器应用举例

在获取到种子页面的内容后，需要解析其中存在的 url（二级 url），并加入待爬取列表，以便下一遍爬取。

解析 html 页面的内容可以通过 xpath 来解析，从而获得其中的 url。

页面中某篇新闻对应的 url 所在位置如下图所示。

图 5 某篇新闻对应的 url 在 html 中的位置

实现方法如下所示：

```
for i in range(6):
    path = '//*[@id="line_u12_' + str(i) + '"]/div[2]/a'
    url = browser.find_element_by_xpath(path).get_attribute('href')
    # 如果未访问过，则加入 url 列表
    if(check_curent_visited(url)):
        urls.append(url)
```

首先可以通过在浏览器查看元素的界面中，右击相关元素可以获得其对应的 xpath 路径，通过 find_element_by_xpath 方法，即可在 python 代码中获得相关元素的内容，但是 url 是一个属性，则需要再次调用其 `get_attribute` 即可获得 url。仍需注意，这里的 url 在浏览器中显示的是相对路径，但 python 提供的方法返回的是绝对路径。

不能将上一步获得的 url 直接加入待爬取列表中，还有很重要的一步，是检查页面是否已经被爬取过。使用一个词典记录新闻页是否已经被爬取，词典的键为 url 的序号（本来应该是 url 的哈希值，但注意到页面不多，且有规律，键值仅仅是 url 末尾的编号，值则是 0 或 1，用来表示该 url 是否被访问过）。该词典通过读取 `visited-set.json` 文件初始化，再获取页面过程结束后，写回到 `visited-set.json` 文件。

具体见 spider.py 的 load.dic()、check_curent_visited(url)和 save_dic()方法。

最后再对获取到的所有 url 进行爬取，即完成了页面获取的全部内容。

三、 页面存储

3.1 网页信息存储的格式

将获取网页信息保存在磁盘中，需要按照规定的格式保存，便于后续处理和提供服务。这里的存储格式只是顺序保存网页信息，没有索引文件。

原始网页库（raw.txt）由若干记录组成，每个记录包含一个网页的原始数据，记录的存放是顺序追加的，记录之间没有分隔符；

每条记录由头部、数据和空行组成，顺序是：头部 + 空行 + 数据 + 空行；（html 文件中有空行）

具体效果下图所示：

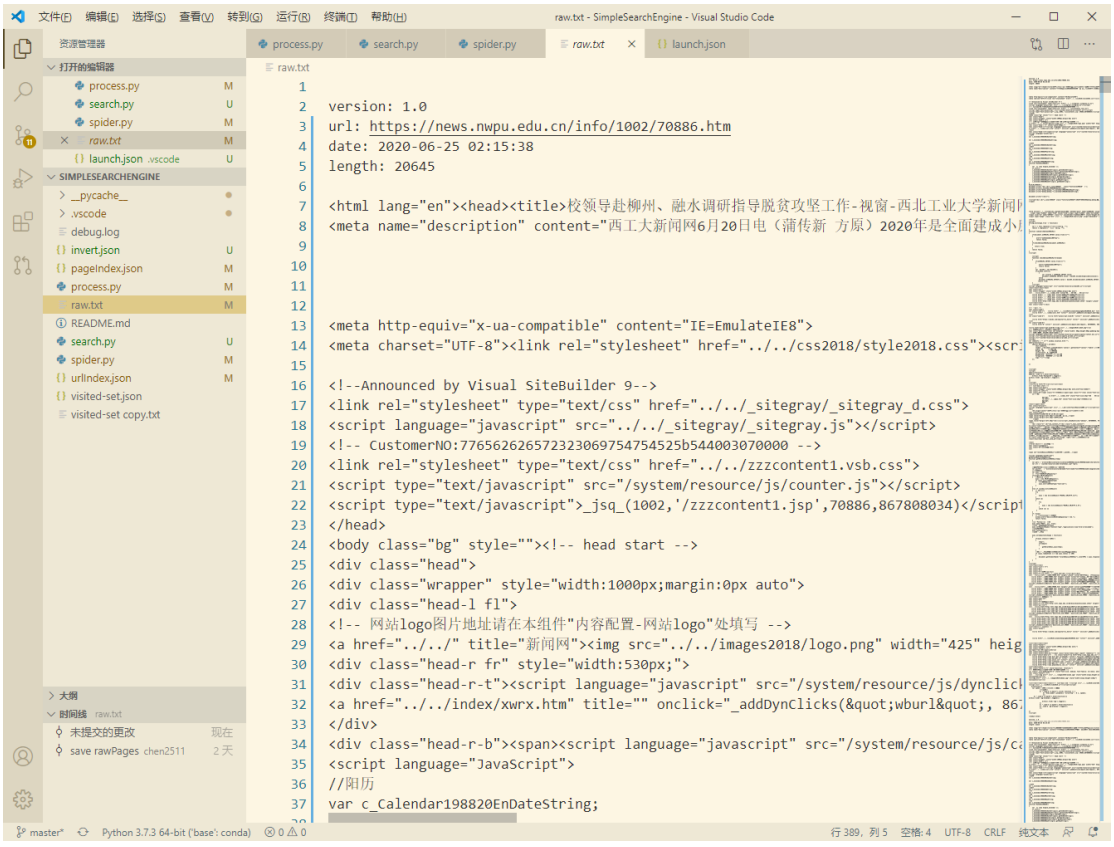

图 6 原始网页库的最终效果

具体见 spider.py 的 save_raw_page(data, rawfile, url)方法：

```
def save_raw_page(data, rawfile, url):
    # data.replace('\n', '')
    rawfile.write('\nversion: 1.0\n')
    rawfile.write('url: ' + url + '\n')
    rawfile.write('date: ' + time.strftime("%Y-%m-%d %H:%M:%S"
, time.localtime()) + '\n')
    rawfile.write('length: ' + str(len(data)) + '\n\n')
```

```
    rawfile.write(str(data))
    # 当前文件指针所在位置
    # print(rawfile.tell())
    rawfile.write('\n')
    return rawfile.tell()
```

值得一提的是，为了便于生成索引网页库，写完内容之后会返回当前的文件指针。

3.2 建立索引网页库

至此，我们体系结构中的第一部分全部完成；下面将开始数据处理部分。首先是建立索引网页库：

索引网页库的任务就是完成给定一个 URL，在原始网页库中定位到该 URL 所指向的记录。如果不建立索引信息，只能通过顺序查找的方法，查找指定的记录，这样会消耗大量的 I/O，数据量增大的时候不能够满足搜索引擎的快速响应要求。

所以我们一共建立两个文件：网页索引文件和 URL 索引文件。

网页索引文件：以 ISAM（索引顺序访问模式）存储，其中每一行不记录文档长度，因为文档长度可以通过后续文档起始位置的偏移和当前文档其位置的偏移的差获得。为了保证对最大文档号数据读取的一致性，在最后一行增加“哨兵”，它不对应实际的文档数据，其中文档摘要为空，表示文档索引的结束。如下图所示：

图 7 网页索引文件：pageIndex.json

URL 索引文件：以 ISAM 存储，包含了 URL 的摘要和文档编号。为了能够快速的对给定 url 找到对应的文档编号，按照 URL 摘要排序。

图 8 URL 索引文件：urlIndex.json

代码实现：

首先是哈希值的计算：见 spider.py 的 get_md5()方法：

```
import hashlib
def get_md5(data):
    # 应用MD5 算法
    md5 = hashlib.md5()
    md5.update(data.encode('utf-8'))
    return md5.hexdigest()
```

输入字符串返回 hash 值

网页索引：

```
# 原始网页索引列表添加新项
def append_pageIndex(data, cnt, index):
    data_md5 = get_md5(data)

    pageIndex[-1]['md5'] = data_md5
    #pageIndex[-1]['pagelen'] = len(data)
    pageIndex.append({'No': cnt, 'offset':index, 'md5': None})
```

url 索引：

```
# 原始网页 url 索引列表添加新项
def append_urlIndex(url, cnt):
    url_md5 = get_md5(url)
    urlIndex.append({'url':url_md5, 'No':cnt})
```

最后要对 url 索引列表根据 url 的 hash 值排序，加速查找速度。

```
    save_json(pageIndex, 'pageIndex.json')
    save_json(urlIndex, 'urlIndex.json')
```

最近结果保存到 json 格式的文件中。

四、 页面解析

部分页面解析的内容在前面已经提到过，如获取网页中存在的 URL。所以该部分内容还包括：网页编码识别和获取标题。

切词前需要识别网页的编码，编码格式不对，导致切出来的词没有意义。

我们可以在网页内容中得到网页的编码格式，即从 http 返回的 head 信息中的 charset 域得到，如下图所示：

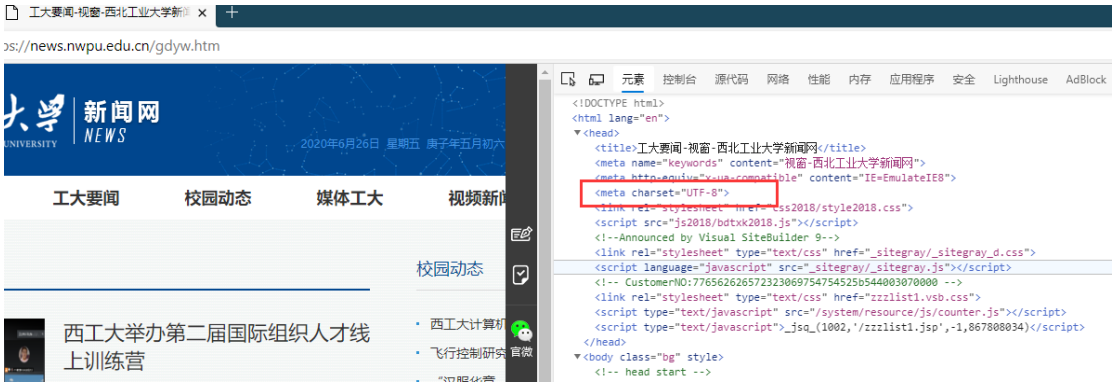

图 9 网页编码识别

获取标题是显示结果时需要的必须步骤，本程序使用的是 BeautifulSoup 包来解析。

```
soup = BeautifulSoup(data, 'html.parser')
title = soup.find("title")
```

五、 切词

生成了所以网页库之后，需要分析网页，并进行分词。分词是网页分析的前提。

分词方法是按照一定的策略将待分析的汉字串与一个充分大的词典中的词条进行匹配，若在词典中找到某个字符串，则匹配成功。按照扫描方向不同，串匹配分词方法可以分为：正向匹配和逆向匹配。按照匹配长度可以分为最长/最大匹配和最短/最小匹配。

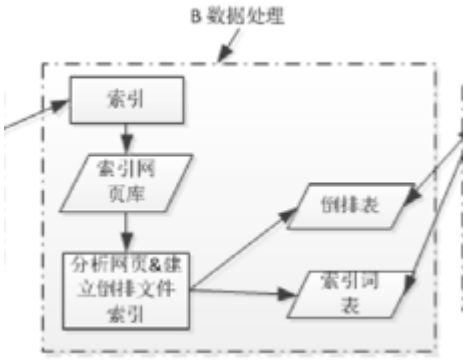

图 10 数据处理流程

本程序使用的是结巴分词。jieba 是一个基于 Python 的中文分词工具。对于一长段文字，其分词原理大体可分为三步：

1.首先用正则表达式将中文段落粗略的分成一个个句子。

2.将每个句子构造成有向无环图，之后寻找最佳切分方案。

3.最后对于连续的单字，采用 HMM 模型将其再次划分。

程序实现：

首先结合网页索引从原始网页库中读取一篇篇原始网页，逐网页进行分析。读取到一篇网页之后，根据正则表达式找出其中的中文内容，之后对这些内容进行切词，根据得到的切词结果，进一步构造倒排文件。

部分代码如下：

```
url, data = read_raw_info(rawfile, pageIndex['offset'])
            #data = data.decode('utf-8')
data_list = re.findall('[\u4e00-\u9fa5]', data)
data_str = "".join(data_list)
token = jieba.tokenize(data_str, mode='search')
```

其中，jieba.tokenize()方法可以指定分词模式为搜索引擎模式，该模式可以把“中国科学院”分成中国，科学，学院，科学院，中国科学院多个词，更适合

搜索引擎的模糊查询。同时还会返回一个分词结果列表，不仅存储分词结果，还存储分的词的起始位置和结束位置。如下图所示：

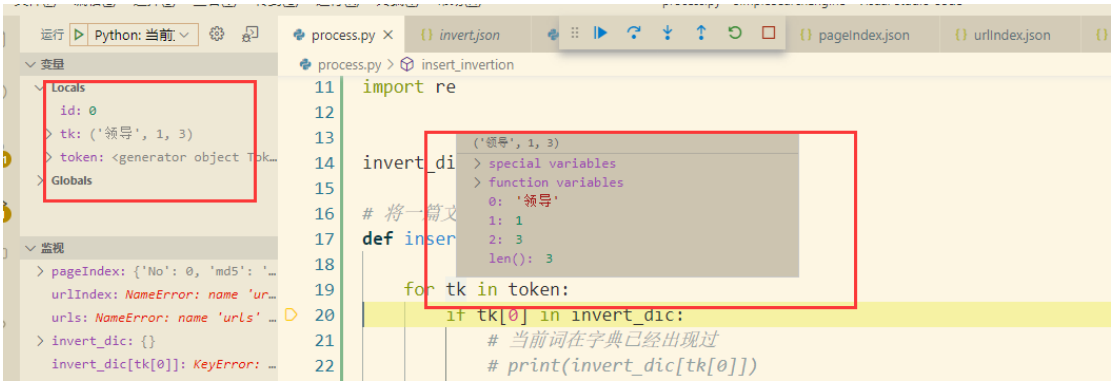

图 11 分词结果

六、 倒排文件

我们切词之后得到的结果是正向索引，为了提高查找效率，我们需要生成倒排文件。

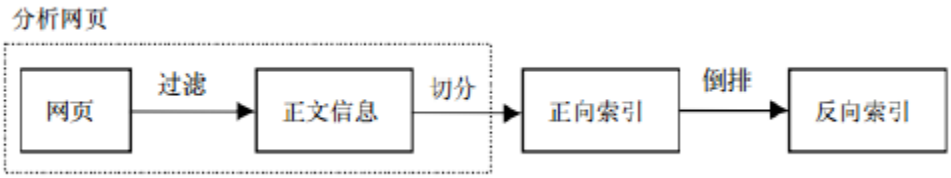

图 12 分析网页流程

关键词	倒排表（所在文档编号，出现次数， 出现位置）
KMP	（ #3307， 2， 5， 43）（ #4615， 5， 0， 19， 34， 70， 143）
最小支撑树	（ #2519， 1， 267）（ #6742， 3， 19， 322， 526）……
贪心算法	（ #2948， 3， 45， 267， 587）（ #3693， 5， 39， 423， 765, 809, 1024）……

图 13 倒排文件格式举例

本程序中使用的是字典存储倒排索引，键是一个个关键词，值对应的是一个列表，列表中的每一项都是一个列表（列表的元素按顺序是：文档编号、关键词在该篇文档中出现的次数，最后是每次出现的位置），其对应一个网页。如上图所示。这样一个二维列表就存储了该关键词在所有文档中出现的次数与位置。

编程实现：见 process.py

```
# 将一篇文档的所有 token 加入倒排文件
def insert_invertion(token, id):
    for tk in token:
        if tk[0] in invert_dic:
            # 当前词在字典已经出现过
            # print(invert_dic[tk[0]])
            if id == invert_dic[tk[0]][-1][0]:
                invert_dic[tk[0]][-1][1] += 1
                invert_dic[tk[0]][-1].append(tk[1])
```

```
            else:
                invert_dic[tk[0]].append([id, 1, tk[1]])
        else:
            invert_dic[tk[0]] = []
            invert_dic[tk[0]].append([id, 1, tk[1]])
```

上述算法是将一篇文档建立倒排文件，之后遍历所有文档即可建立所有的倒排文件。

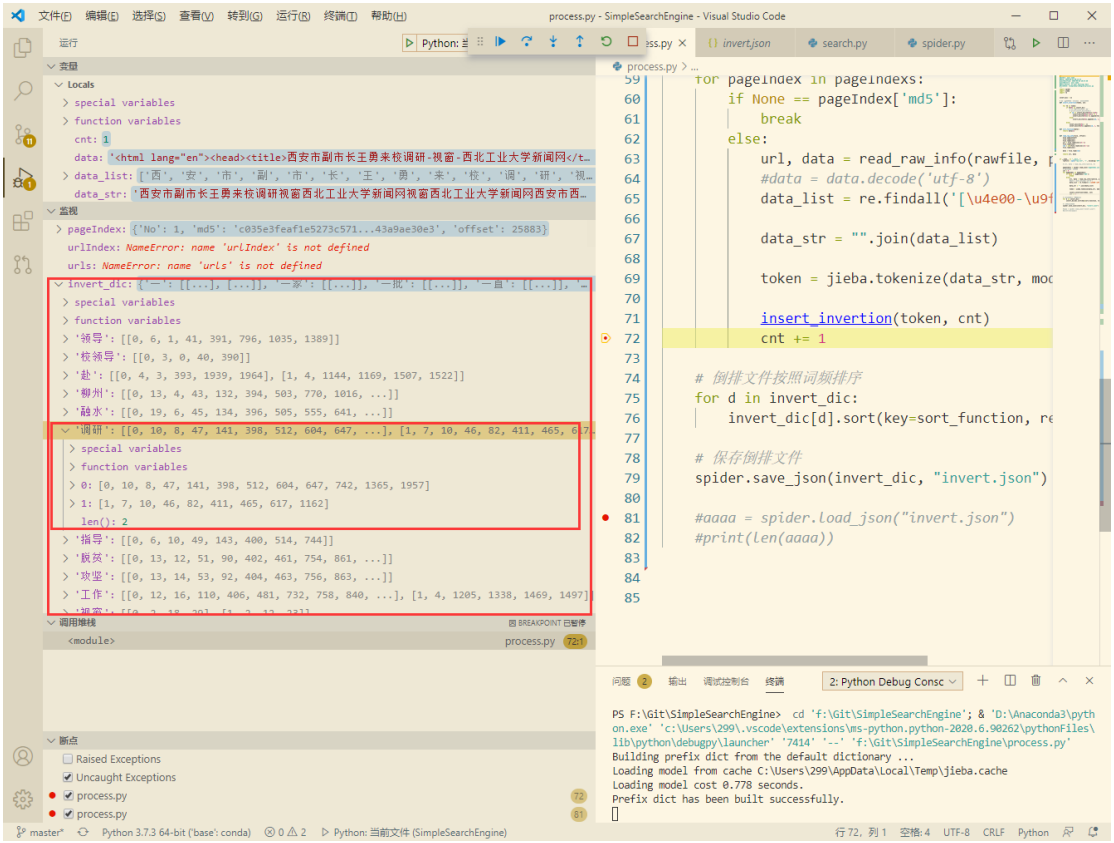

图 14 倒排索引示例

还要将倒排索引进行排序（这样便于显示搜索结果），遍历每个关键词，按照每个文档中当前关键词出现的频次进行排序，这样在搜索该关键词时，显示的结果将会是提前排好序的。

最后调用之前写好的方法，将字典写如 json 格式的文件中：invert.json。

七、 查询服务

查询服务包括接受用户输入的查询短语、检索、获得相应的匹配结果并显示给用户。搜索引擎三段式工作流程的最后一个环节。

实现流程：首先接受输入的词汇，之后读取倒排文件，找到相关词汇的文档序号。由于生成倒排文件时已经按照词频排好序，又因为查询结果排序我使用的就是词频，所以得到相关文档的序号的列表之后即是我们需要显示的内容。有时相关文章太多，那么将显示部分网页。

根据要显示的网页的序号，我们还需要读取原始网页库，获取 url 和标题和摘要。获取 url 和标题较为简单，摘要的生成是截取关键词前后的一定词，词的数量通过程序中 abstract_ahead_offset 和 abstract_len 参数来实现，其表示关键词前多少个字和一共取得长度。

程序实现：search.py

general_pageinfo（）：

输入：倒排索引关键词对应的列表数组里的一个列表，也就是一篇文档的倒排索引：[文档 id，词频，索引词每次出现的位置……]

输出：标题、url，摘要

```
def general_pageinfo(page):
    no = page[0]
    # time = page[1]
    pos = page[2:]
    raw_offset = pageIndexs[no]['offset']
    rawfile = open('raw.txt', 'r', encoding='UTF-8')
    url, data = process.read_raw_info(rawfile, raw_offset)
    soup = BeautifulSoup(data, 'html.parser')
    title = soup.find("title")
    data_list = re.findall('[\u4e00-\u9fa5]', data)
    data_str = "".join(data_list)
    abstract = get_abstract(data_str, pos[0])
    rawfile.close()
    return title.text, url, abstract
```

生成摘要：get_abstract（）

注意：生成的摘要的内容仅包括关键词第一次出现位置的前后文。

输入：原始网页正文，关键词偏移

输出：关键词前后的一定长度的上下文

```
def get_abstract(data, offset):
    if(offset > abstract_ahead_offset):
        offset -= abstract_ahead_offset
    else:
        offset = 0
    return data[offset : offset + abstract_len]
```

八、 使用说明:

由于未了解相关 web 编程，所有搜索结果的显示部分，将通过终端输出。结果中的 url 也可以通过 ctrl+鼠标点击即可打开浏览器访问。输入的查询词也只接受一个（考虑到分词时是按照搜索引擎模式分词分的，可以实现模糊查询的功能）

首先在终端（推荐 vscode 执行，因为 url 可以点击）执行：python .\search.py

之后会提示输入查询词，若有查询结果，则显示相关结果，否则提示未找到相关词汇。

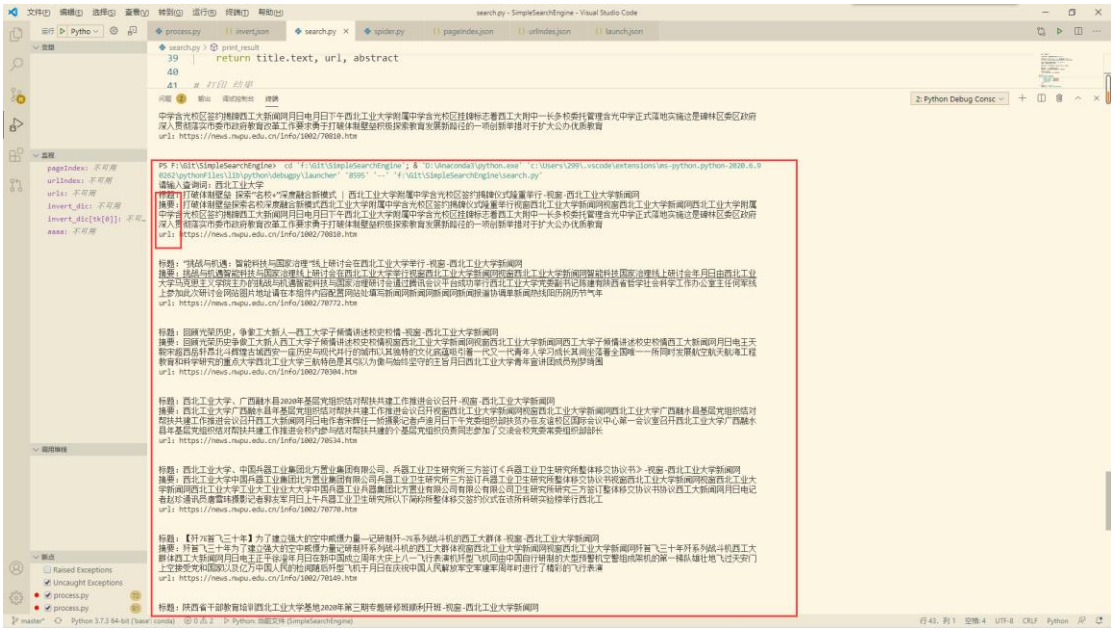

图 15 搜索“西北工业大学”示例

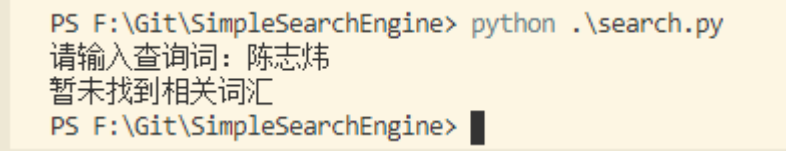

图 16 查找失败示例

九、 遇到的问题

9.1 代理错误

本来计划使用 request 的包，但是始终遇到错误：

Failed to establish a new connection: [WinError 10061] 由于目标计算机积极拒绝，无法连接。

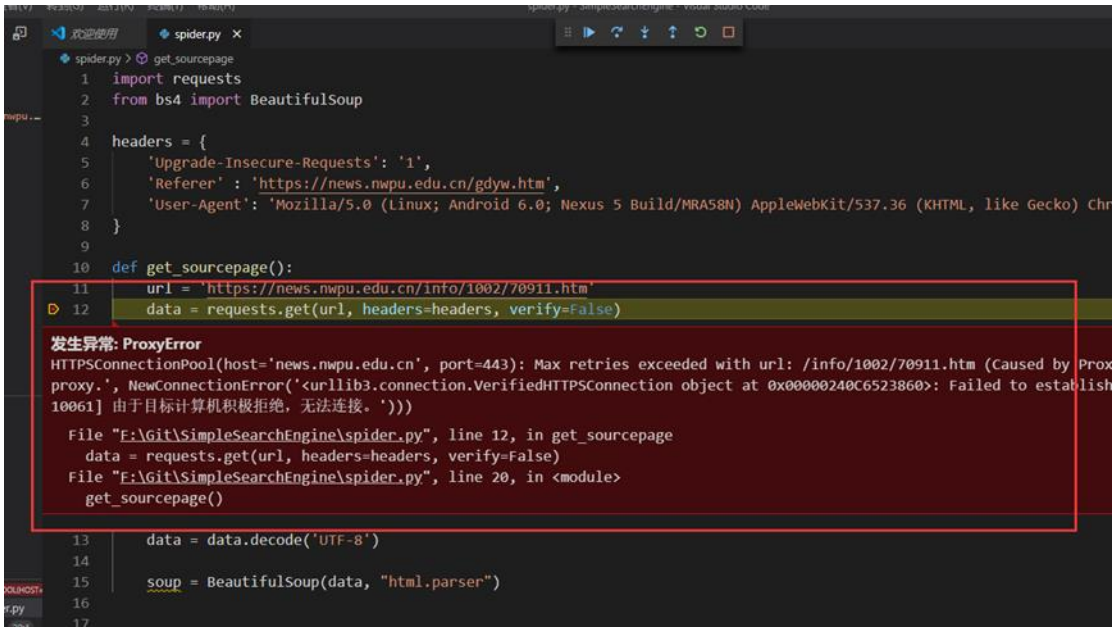

图 17 连接错误

一开始认为是反爬虫的问题，我加了 header 之后还是报错，而且其他网站也是这样。后来觉得可能是代理的问题，去改系统的设置，重启也还是错误。之后使用浏览器的模拟器来获取网页内容。

但在室友的帮助下最后解决了，原来是之前使用代理的时候直接关闭程序或者关机，导致系统出错。

9.2 \n 计数方法不一致

在读取原始网页时，本来向通过网页索引文件中的前后偏移位置的差来作为读取的长度，如：file.read(len)。

但是发现总是会多读取内容，后来通过试验发现，对于'\n'文件指针会把它当作两个字节，而 file.read()方法只把它读作一个字节，这样通过指针的差来确定长度总是会读更多的东西。

所以最终，我在原始网页库中新增一个网页长度的字段，从而确定要读取的

网页的长度。

9.3 全局变量未声明

如下图所示，未声明全局变量 Dic 时，在 77 行代码执行之后，返回的值只会赋给一个局部变量 DIc，而不是全局变量 Dic，在函数返回之后，Dic 始终是空的。

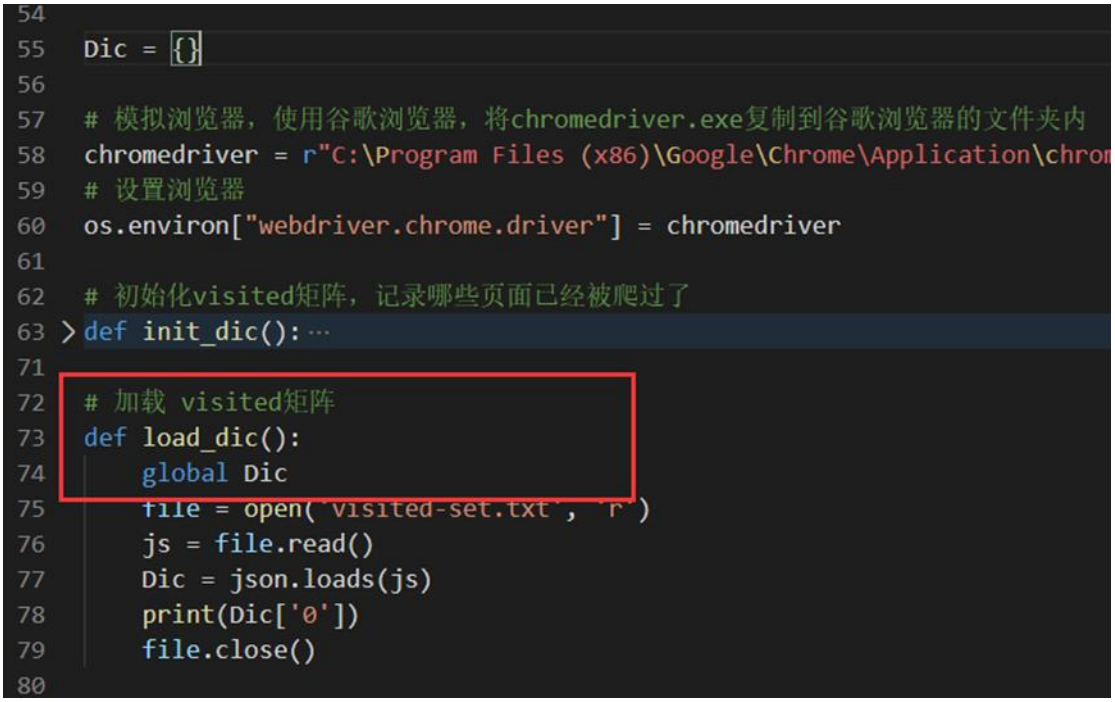

图 18 全局变量未声明

十、 参考文献

1、老师的 PPT 和课本《搜索引擎原理，技术与系统》

2、使用 python 实现简单的搜索引擎

3、jieba 分词使用方法

4、python 文件的基本操作之控制文件指针