HW5 Web搜索引擎

- 姓名：管昀玫
- 学号：2013750
- 专业：计算机科学与技术

本次作业要求实现一个系统的Web搜索引擎（主题不限），为用户提供查询服务和个性化推荐。实现本次作业主要有网页**抓取、文本索引、链接分析、查询服务、个性化查询**几个步骤，个性化推荐为扩展内容。

文件结构为：

```
│  app.py                                  ————入口文件
│  requirements.txt
│
├─app
│  │  public_const.py                      ————公用函数库
│  │  __init__.py
│  │
│  ├─front                                 ————路由包
│  │      advanced_search.py                ————高级搜索路由
│  │      index.py                          ————主页路由
│  │      result_page.py                    ————普通搜索结果页路由
│  │      snapshot.py                       ————快照路由
│  │      suggest.py                        ————个性化建议路由
│  │      __init__.py
│  │
│  ├─static                                ————静态资源文件夹
│  │  └─js
│  │          jquery-3.6.3.slim.min.js
│  │          suggest.js
│  │
│  ├─templates                               ————前端html模板文件
│  │      advanced_search.html
│  │      base.html
│  │      index.html
│  │      no_result_page.html
│  │      result_page.html
│  │      snapshot.html
│  │
│  └─utils
│          advanced_search_func.py          ————高级检索方法
│          search_func.py                   ————搜索/高级检索第一次搜索方法
│          __init__.py                      ————闭包文件
│
└─tools
      1-worm.py                              ————爬虫
      2-text_index.py                        ————预处理文件
      3-term_frequency.py                    ————排序
      4-advanced_search_index.py              ————高级搜索用排序
     __init__.py
```

1. 网页抓取

Web采集是从Web中收集网页的过程，这些网页用于索引从而为搜索引擎提供支持。采集的目标是尽可能高效地采集更多数目的有用页面，并同时获得连接这些页面的链接结构。

任何超文本采集器（不论是面向 Web、内网还是其他的超文本文档集）的基本处理如下：首先，设定一个或者多个 URL 为采集的种子集合（seed set）。接着，从种子集合中选择一个 URL进行采集，然后对采集到的页面进行分析，并抽取出页面中的文本和链接（每个链接都链向其他的 URL）。抽取出的文本输给文本索引器（参见第 4 章和第 5 章的介绍），而抽取出的 URL则加入到待采集 URL 池（URL frontier，以下简称 URL 池）中，任何时候 URL 池中放的都是所有待采集网页的 URL。一开始，种子集合会放入 URL 池中，一旦某个 URL 被采集，那么就从池中删除这个地址。整个采集过程可以看成是 Web 图的遍历过程。当然，在连续式采集中，一个已采集的网页的 URL 还会被重新放到 URL 池中以等待下一次重新采集。

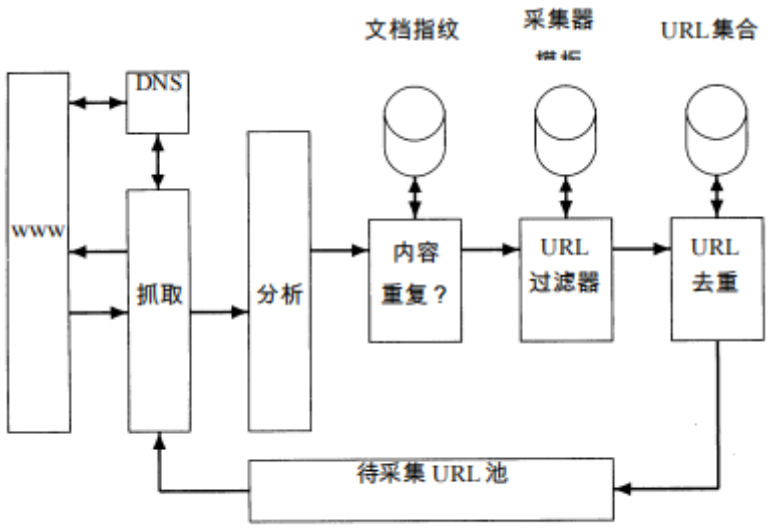


图 20-1　采集器的基本架构

我选择南开大学新闻网http://news.nankai.edu.cn/ywsd/index.shtml进行抓取，生成50页新闻列表页的url（每页有30篇新闻）

由于常用的requests库只支持同步阻塞爬取，而爬虫又是典型的IO密集型任务，可以使用异步优化，因此使用了支持异步的`httpx`库。调用`asyncio`标准库进行异步抓取，加快抓取速度。同时，为了在爬取过程中不给服务器带来负担，所以限制了协程数量（10）。

核心代码如下：

```
# 解析新闻列表页
async def parse_catalogs_page(url):
    async with sem:
        async with httpx.AsyncClient() as client:
            response = await client.get(url) # 异步请求
            selector = Selector(response.text) # 解析网页
            # 获取新闻列表页中的新闻标题和url
            temp_dict = zip(selector.css('a::attr(href)').getall(),
selector.css('a::text').getall())
            result_dict.update(temp_dict) # 将新闻标题和url添加到result_dict中

# 解析新闻详情页
async def parse_page(url):
    async with sem:
        try:
            # 异步请求
            async with httpx.AsyncClient(follow_redirects=True, timeout=10,
```

```
                                        headers={'User-Agent': 'Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/108.0.0.0 Safari/537.36'}) as client:
                if url.startswith('http') or url.startswith('https'):
                    response = await client.get(url)
                    selector = Selector(response.text)
                    title = selector.css('title::text').get() # 获取新闻详情页的标题
                    try:
                        if "/" in title:
                            title = title.replace("/", "_") # 将标题中的/替换为_,避
免文件名中出现/
                        async with aiofiles.open(f'./pages/{title}.html',
mode='w', encoding='utf-8') as f:
                            await f.write(response.text) # 异步写入文件
                        title_url_df.loc[title] = url  # 建立标题和url的映射关系

                    except Exception as e:
                        print(f'{e}: {url}|{title}')
        except:
            print(f'error: {url}')
```

存储的title_url.csv文件格式如下：

	nchu	url
2	媒体南开	http://news.nankai.edu.cn/mtnk/index.shtml
3	南开要闻	http://news.nankai.edu.cn/ywsd/index.shtml
4	扬州校友会反哺母校 颁	http://news.nankai.edu.cn/ywsd/system/2021/10/18/030048375.shtml
5	光影南开	http://news.nankai.edu.cn/gynk/index.shtml
6	南开故事	http://news.nankai.edu.cn/nkrw/index.shtml
7	南开大学：一片紫杉林	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048364.shtml
8	充满挑战的环己六酮（(	http://news.nankai.edu.cn/ywsd/system/2021/10/18/030048368.shtml
9	统计学科育人主题项目/	http://news.nankai.edu.cn/ywsd/system/2021/10/18/030048372.shtml
10	南开大学“享趣运动联盟	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048366.shtml
11	南开西藏校友会向我校	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048355.shtml
12	南开大学成立新闻与传	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048367.shtml
13	“天津论坛2021”在津举	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048358.shtml
14	校领导为我校年轻干部	http://news.nankai.edu.cn/ywsd/system/2021/09/17/030047966.shtml
15	南开大学报	http://news.nankai.edu.cn/nkdxb/index.shtml
16	张全兴院士受聘南开大	http://news.nankai.edu.cn/ywsd/system/2021/10/15/030048350.shtml
17	我校在“2021年中俄大学	http://news.nankai.edu.cn/ywsd/system/2021/10/18/030048369.shtml
18	视频	http://news.nankai.edu.cn/sp/index.shtml
19	广播-南开大学	http://news.nankai.edu.cn/gb/index.shtml
20	专家学者聚南开探讨化	http://news.nankai.edu.cn/ywsd/system/2021/10/18/030048371.shtml
21	【党史学习教育】曹雪	http://news.nankai.edu.cn/ywsd/system/2021/10/15/030048349.shtml
22	南开大学项目入选全国	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048362.shtml
23	南开大学物理学科创建	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048357.shtml
24	张全兴院士与南开学子	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048361.shtml
25	学者嘉宾齐聚天津 聚焦	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048359.shtml
26	数智时代的网络空间治	http://news.nankai.edu.cn/ywsd/system/2021/10/18/030048374.shtml
27	南开大学-南开大学	http://news.nankai.edu.cn/index.shtml
28	1.6亿！校友献礼南开化	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048360.shtml
29	南开大学建校102周年暨	http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048365.shtml
30	南开大学化学学科创建	http://news.nankai.edu.cn/ywsd/system/2021/10/16/030048354.shtml

2. 文本索引

对网页及其锚文本构建索引，可以按锚文本、网页标题、URL 等域构建索引。

2.1 预处理

这里首先需要建立一个dataframe，其中以url作为索引，具体格式为：`url | title | description | date_timestamp | content | editor`。

建立一个dict，用于存储该URL的html文本内容中所含有的URL，以便后续PageRank的计算。

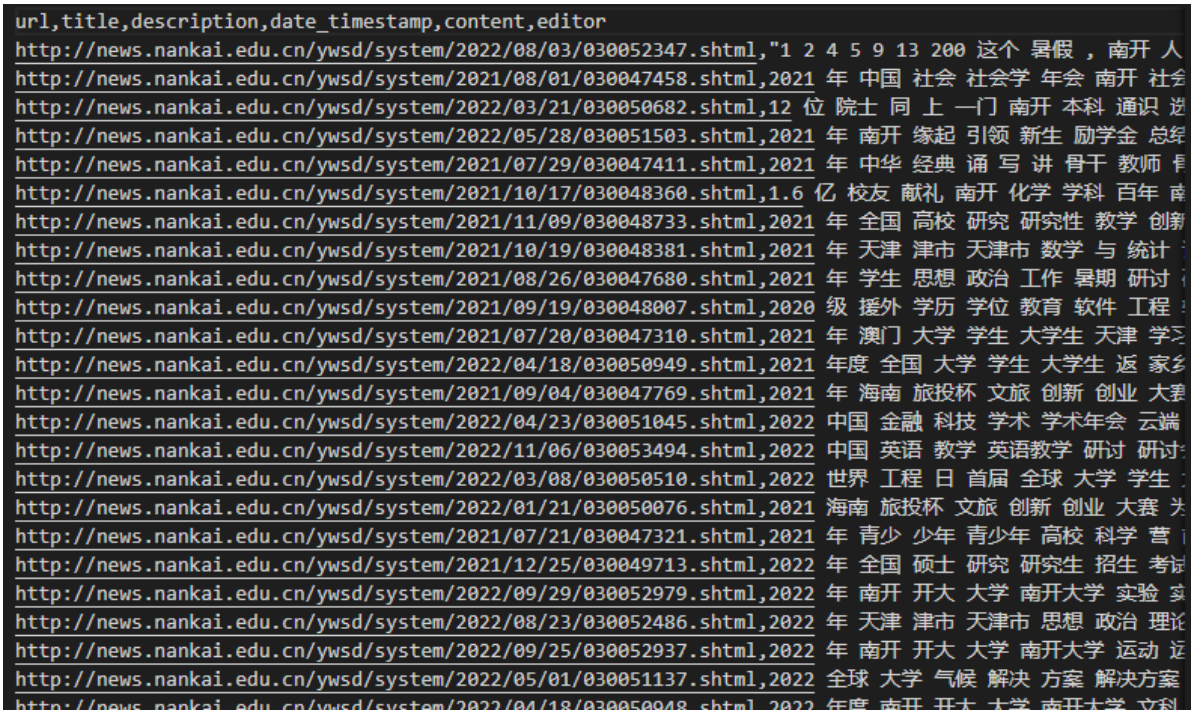
同时还需要对爬取的网页进行**预处理**：

1. 获取title、description、content，并将title将/替换为_，因为在windows系统下/是文件保留词，无法使用。而中文标题中一般不会使用_符号。
2. 处理content：将正文内容和编辑信息分离，编辑信息单独存储，并将`\r`、`\n`、`\t`、` （空格的html转义）`等特殊字符去除。
3. 观察新闻页发布时间格式，为YYYY/MM/DD，使用正则匹配后，转换为时间戳存储
4. 对title、description、content使用`jieba`库`cut_for_search`，也就是搜索引擎模式进行分词
5. 对title、description、content进行处理：list转str，使用特殊符号分隔，并在删除标点后，删除原有空格并将分隔符号改为空格
6. 提取页面中含有的URL，存储到以该页面URL为索引的dict中。

重新保存带有description的title_url.csv，方便后续使用。更新后的title_url.csv格式如下：

title	url	description
媒体南开	http://news.nankai.edu.cn/mtnk/index.shtml	
南开要闻	http://news.nankai.edu.cn/ywsd/index.shtml	
扬州校友会	http://new	南开新闻网讯10月17日，适逢南开大学迎来建校102周年华诞，2021年扬
光影南开	http://news.nankai.edu.cn/gynk/index.shtml	
南开故事	http://news.nankai.edu.cn/nkrw/index.shtml	
南开大学：	http://new	南开新闻网记者蓝芳吴军辉摄影宗琪琪看着一颗颗娇艳的红豆从这片紫杉
充满挑战的	http://new	陈军院士和张新星研究员课题组联合攻关取得新突破南开新闻网讯（通讯
统计学科育	http://new	南开新闻网讯（通讯员宋辰王晓雅）10月16日，南开大学统计学科育人主
南开大学“	http://new	南开新闻网讯（通讯员阎宝岩）10月17日，南开大学享趣运动联盟运动宝
南开西藏校	http://new	南开新闻网讯（通讯员刘晓彤李佳音杨蕊）10月12日，南开西藏校友会联
南开大学成	http://new	
“天津论坛2	http://new	南开新闻网讯（记者 付坤吴军辉摄影吴军辉）10月16日至17日，天津论

将分词后的结果存储到index.csv中。



```
url,title,description,date_timestamp,content,editor
http://news.nankai.edu.cn/ywsd/system/2022/08/03/030052347.shtml,"1 2 4 5 9 13 200 这个 暑假 ， 南开 人
http://news.nankai.edu.cn/ywsd/system/2021/08/01/030047458.shtml,2021 年 中国 社会 社会学 年会 南开 社会
http://news.nankai.edu.cn/ywsd/system/2022/03/21/030050682.shtml,12 位 院士 同 上 一门 南开 本科 通识 选
http://news.nankai.edu.cn/ywsd/system/2022/05/28/030051503.shtml,2021 年 南开 缘起 引领 新生 励学金 总结
http://news.nankai.edu.cn/ywsd/system/2021/07/29/030047411.shtml,2021 年 中华 经典 诵 写 讲 骨干 教师 作
http://news.nankai.edu.cn/ywsd/system/2021/10/17/030048360.shtml,1.6 亿 校友 献礼 南开 化学 学科 百年 南
http://news.nankai.edu.cn/ywsd/system/2021/11/09/030048733.shtml,2021 年 全国 高校 研究 研究性 教学 创新
http://news.nankai.edu.cn/ywsd/system/2021/10/19/030048381.shtml,2021 年 天津 津市 天津市 数学 与 统计
http://news.nankai.edu.cn/ywsd/system/2021/08/26/030047680.shtml,2021 年 学生 思想 政治 工作 暑期 研讨
http://news.nankai.edu.cn/ywsd/system/2021/09/19/030048007.shtml,2020 级 援外 学历 学位 教育 软件 工程
http://news.nankai.edu.cn/ywsd/system/2021/07/20/030047310.shtml,2021 年 澳门 大学 学生 大学生 天津 学习
http://news.nankai.edu.cn/ywsd/system/2022/04/18/030050949.shtml,2021 年度 全国 大学 学生 大学生 返 家乡
http://news.nankai.edu.cn/ywsd/system/2021/09/04/030047769.shtml,2021 年 海南 旅投杯 文旅 创新 创业 大赛
http://news.nankai.edu.cn/ywsd/system/2022/04/23/030051045.shtml,2022 中国 金融 科技 学术 学术年会 云端
http://news.nankai.edu.cn/ywsd/system/2022/11/06/030053494.shtml,2022 中国 英语 教学 英语教学 研讨 研讨
http://news.nankai.edu.cn/ywsd/system/2022/03/08/030050510.shtml,2022 世界 工程 日 首届 全球 大学 学生
http://news.nankai.edu.cn/ywsd/system/2022/01/21/030050076.shtml,2021 海南 旅投杯 文旅 创新 创业 大赛 关
http://news.nankai.edu.cn/ywsd/system/2021/07/21/030047321.shtml,2021 年 青少 少年 青少年 高校 科学 营
http://news.nankai.edu.cn/ywsd/system/2021/12/25/030049713.shtml,2022 年 全国 硕士 研究 研究生 招生 考试
http://news.nankai.edu.cn/ywsd/system/2022/09/29/030052979.shtml,2022 年 南开 开大 大学 南开大学 实验 实
http://news.nankai.edu.cn/ywsd/system/2022/08/23/030052486.shtml,2022 年 天津 津市 天津市 思想 政治 理论
http://news.nankai.edu.cn/ywsd/system/2022/09/25/030052937.shtml,2022 年 南开 开大 大学 南开大学 运动 运
http://news.nankai.edu.cn/ywsd/system/2022/05/01/030051137.shtml,2022 全球 大学 气候 解决 方案 解决方案
http://news.nankai.edu.cn/ywsd/system/2022/04/18/030050948.shtml,2022 年度 南开 开大 大学 南开大学 文科
```

2.1.2 设置排除停用词

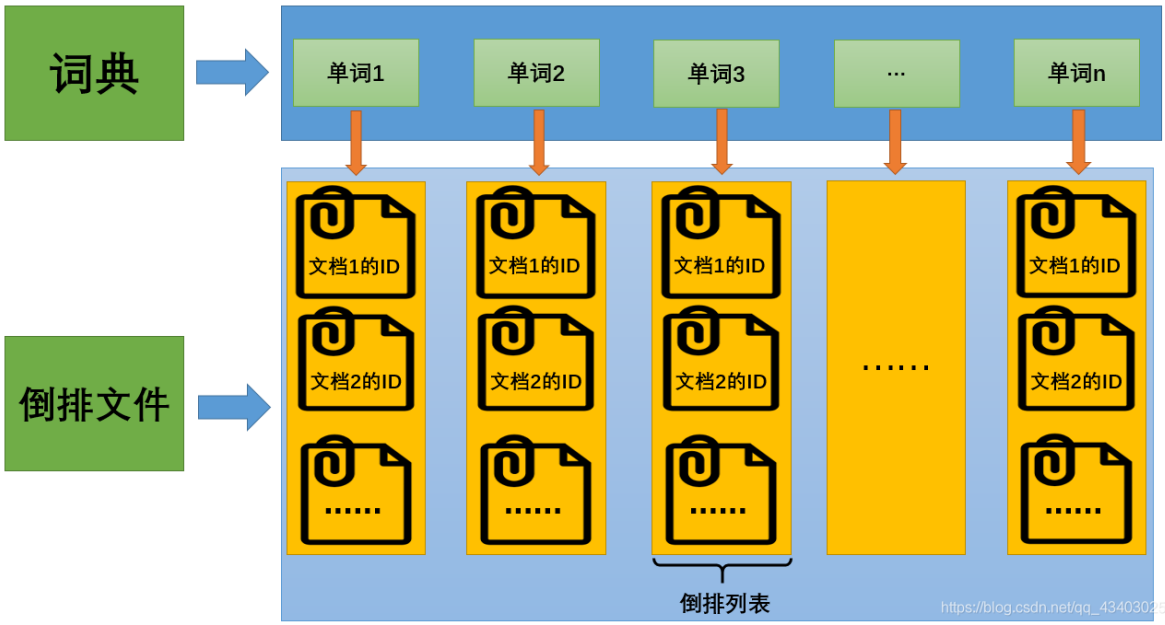
这里使用的是scu_stopwords，同时还要添加以下的词：'要闻', '新闻', '新闻网', '讯','...'

2.2 倒排索引与文本频率

2.2.1 倒排索引

倒排索引(Inverted Index)：倒排索引是实现"单词-文档矩阵"的一种具体存储形式，通过倒排索引，可以根据单词快速获取包含这个单词的文档列表。倒排索引主要由两个部分组成：**"单词词典"和"倒排文件"**。

- **单词词典(Lexicon)**：搜索引擎的通常索引单位是单词，单词词典是由文档集合中出现过的所有单词构成的字符串集合，单词词典内每条索引项记载单词本身的一些信息以及指向"倒排列表"的指针。
- **倒排列表(PostingList)**：倒排列表记载了出现过某个单词的所有文档的文档列表及单词在该文档中出现的位置信息，每条记录称为一个倒排项(Posting)。根据倒排列表，即可获知哪些文档包含某个单词。
- **倒排文件(Inverted File)**：所有单词的倒排列表往往顺序地存储在磁盘的某个文件里，这个文件即被称之为倒排文件，倒排文件是存储倒排索引的物理文件。



在我的实现中，为了简单，我直接使用了一个dict来构建倒排索引：索引值为word，其中word又是一个dict，其索引值为url，value值为频率。因为python的dict本质是哈希表，由C作为底层实现，因此性能尚可，并且代码简便方便调试；但若索引内容增多，空间复杂度将急剧增大，因此这种写法仅适用于网页不多的情况下（本次适用的网页数量在1500左右）

```
# 从df构建文本索引
index = {}
for url, row in df.iterrows():
    index[url] = {} # index存储每个url里的具体内容的词频
    for word in row['title'].split(" "):
        if word not in stopwords:
            if word not in index[url]:
                index[url][word] = 1
            else:
                index[url][word] += 1
    for word in row['description'].split(" "):
        if word not in stopwords:
            if word not in index[url]:
```

```
                index[url][word] = 1
            else:
                index[url][word] += 1
    for word in row['content'].split(" "):
        if word not in stopwords:
            if word not in index[url]:
                index[url][word] = 1
            else:
                index[url][word] += 1
    for word in row['editor'].split(" "):
        if word not in stopwords:
            if word not in index[url]:
                index[url][word] = 1
            else:
                index[url][word] += 1
    if index[url].get('') != None:
        del index[url]['']

# 构建倒排字典
inverted_index = {}
for url, words in index.items():
    for word, frequency in words.items():
        if word not in inverted_index:
            inverted_index[word] = {}
            inverted_index[word][url] = frequency
        else:
            inverted_index[word][url] = frequency
```

2.2.2 tf-idf

- 有一个文档集合C，文档集合C里面一共有N篇文档
- 有一个词典D，或者叫词库（Vocabulary），词库里面一共有M个词

文档到向量的转化

向量是有长度的，向量中的每个元素是数值。

首先将文档通过词袋模型转化成一个个的词，一般地，由于文档中的词都会存在于词典D中，定义一个M维向量（M等于词典大小），若文档中的某个词在词典中出现了，就给这个词赋一个实数值。若未出现，则在相应位置处赋值为0。

使用TF-IDF来衡量每个词的权重

TF值：tf(term frequency)，是指 term 在某篇文档中出现的频率。tf 是针对单篇文档而言的，即：**某个词在这篇文档中出现了多少次。**词频是计算文档得分的一个因子，因此为了计算某篇文档的得分，使用的词频指的就是term在这篇文档中出现的次数，而不是term在所有文档中出现的次数。

$$TF = log_{10}(N+1)$$

IDF值：idf值 由 词(term) 出现在各个文档中数目来决定。idf 值是针对所有文档(文档集合)而言的，即：**数一数这个词都出现在哪些文档中。**

$$idf_t = log\frac{N}{df_t}$$

TF-IDF值

$$TF-IDF=tf*idf$$

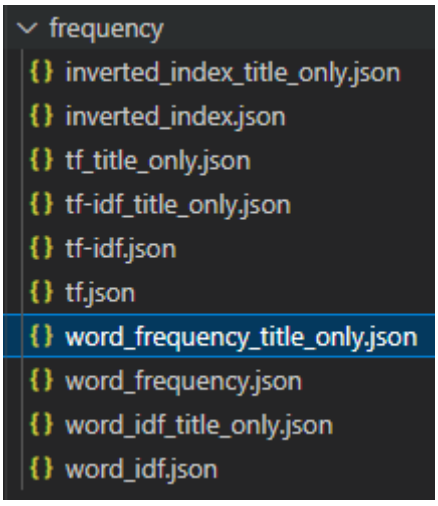
具体代码如下：

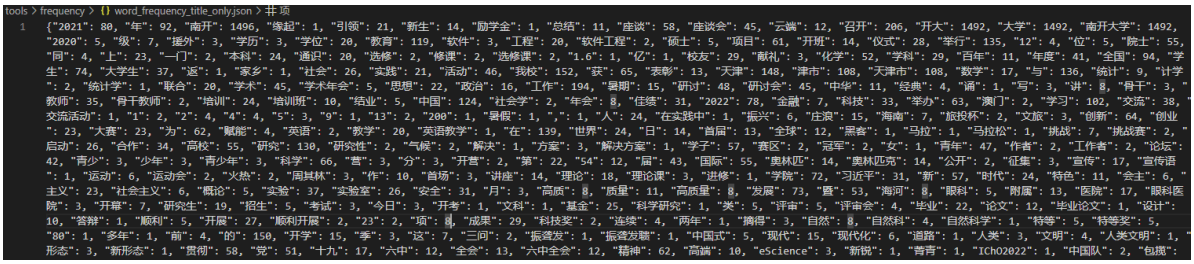
```
# 计算词频
word_frequency = {} # word_frequency存储每个词的词频
for url, words in index.items():
    for word, frequency in words.items():
        if word not in word_frequency:
            word_frequency[word] = 1
        else:
            word_frequency[word] += 1

# 计算逆文档频率
word_idf = {}
for url, frequency_dict in index.items():
    for word, frequency in frequency_dict.items():
        word_idf[word] = math.log(len(index) / frequency)

# 计算tf
tf = {}
for url, words in index.items():
    temp_dict = {}
    for word, frequency in words.items():
        if word not in temp_dict:
            temp_dict[word] = 1
        else:
            temp_dict[word] += 1
    tf[url] = temp_dict

# 计算tf-idf
tf_idf = {}
for url, words in index.items():
    temp_dict = {}
    for word, frequency in words.items():
        temp_dict[word] = frequency * word_idf[word]
    tf_idf[url] = temp_dict

############
# 具体查询实现
############
# 统计词项tj在文档Di中出现的次数，也就是词频。
def computeTF(word_set, split):
    tf = dict.fromkeys(word_set, 0)
    for word in split:
        if word in word_set:
            tf[word] += 1
    for word, cnt in tf.items():
        tf[word] = math.log10(cnt + 1)  # TF = log10(N + 1) 减少文本长度带来的影响
    return tf

# 计算逆文档频率IDF
def computeIDF(tf_list):
    idf_dict = dict.fromkeys(tf_list[0], 0)  # 词为key，初始值为0
    N = len(tf_list)  # 总文档数量
    for tf in tf_list:  # 遍历字典中每一篇文章
```

```
        for word, count in tf.items():  # 遍历当前文章的每一个词
            if count > 0:  # 当前遍历的词语在当前遍历到的文章中出现
                idf_dict[word] += 1  # 包含词项tj的文档的篇数df+1
    for word, Ni in idf_dict.items():  # 利用公式将df替换为逆文档频率idf
        idf_dict[word] = round(math.log10(N / Ni), 4)  # N,Ni均不会为0 IDF = log10(N / df_t)
    return idf_dict  # 返回逆文档频率IDF字典

# 计算tf-idf(term frequency-inverse document frequency)
def computeTFIDF(tf, idfs):  # tf词频,idf逆文档频率
    tfidf = {}
    for word, tfval in tf.items():
        tfidf[word] = tfval * idfs[word]
    return tfidf

def length(key_list):
    num = 0
    for i in range(len(key_list)):
        num = num + key_list[i][1] ** 2
    return round(math.sqrt(num), 2)
```

注意，在构建完整体文本的倒排索引与tf-idf之后，还要构建一版title-only索引，以便用于仅检索标题相关的功能。

处理过后的json文件示例：





3. 链接分析

使用PageRank进行链接分析，评估网页权重。

搜索引擎是如何进行网页的相关性排序的：除了看网页本身的关键词密度和关键词位置外，还要看一个更重要的要素，就是链接流行度（或称之为链接分析），几个方面结合起来就能让排序更加精确。链接流行度的原理是，一个网页拥有的反向链接越多，就越有可能是高质量网页，不然也不会有更多人愿意为其做链接。因此，在其他因数相同的条件下，反向链接越多的网页排名更靠前。

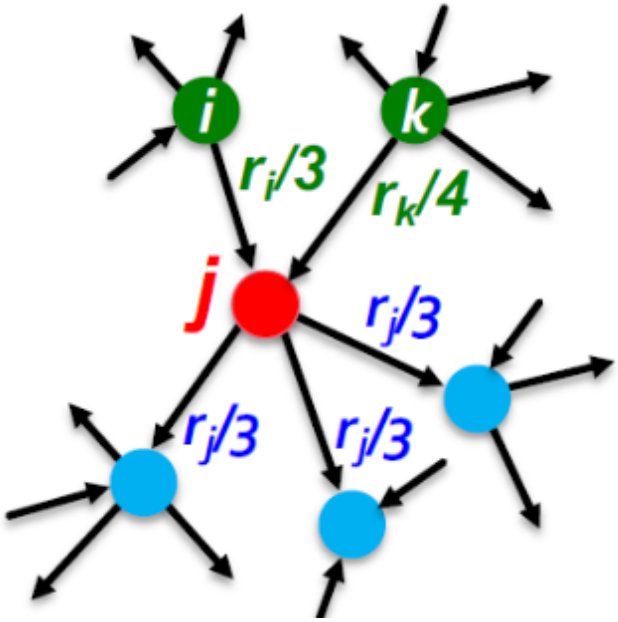
可以看到，在抓取的网页中会包含一些html链接，使用这一信息我们就能计算PageRank：

```
<a href="http://news.nankai.edu.cn/ywsd/system/2022/11/28/030053859.shtml" target="_blank">南开大学召开党委常委会（扩...</a><br/>
<a href="http://news.nankai.edu.cn/ywsd/system/2022/11/28/030053838.shtml" target="_blank">【爱国奋斗南开人】中国大学...</a><br/>
<a href="http://news.nankai.edu.cn/ywsd/system/2022/11/29/030053867.shtml" target="_blank">【学习贯彻二十大】南开大学...</a><br/>
<a href="http://news.nankai.edu.cn/ywsd/system/2022/11/26/030053830.shtml" target="_blank">【学习贯彻二十大】天津高校...</a><br/>
<a href="http://news.nankai.edu.cn/ywsd/system/2022/11/29/030053860.shtml" target="_blank">【学习贯彻二十大】学校召开...</a><br/>
<a href="http://news.nankai.edu.cn/ywsd/system/2022/11/30/030053874.shtml" target="_blank">我校2门课程入选2022年天津市...</a><br/>
<a href="http://news.nankai.edu.cn/ywsd/system/2022/12/01/030053886.shtml" target="_blank">专家学者南开论道 研究阐释中...</a><br/>
<a href="http://news.nankai.edu.cn/ywsd/system/2022/12/01/030053894.shtml" target="_blank">有机新物质创造前沿科学中心...</a><br/>
<a href="http://news.nankai.edu.cn/ywsd/system/2022/12/02/030053897.shtml" target="_blank">南开大学科技成果获天津市专...</a><br/>
<a href="http://news.nankai.edu.cn/ywsd/system/2022/11/26/030053831.shtml" target="_blank">“南开大学—中汽数据有限...</a><br/>
```

Page Rank的简单递推公式：

1. 所有链接的投票权重与其源网页的**权重**成比例
2. 页面j的权重为r_j，拥有n个出链，则每个出链有的投票权重为$r_{j_{out}} = \frac{r_j}{n}$
3. 页面j自身的权重r_j为其入链权重之和
4. Page Rank网络局部示意:

$$r_j = r_i/3 + r_k/4$$



- 一个网页如果入链越多，则越重要（PageRank越高）
- 一个网页如果被越重要的网页所指向，则越重要（PageRank越高）

核心代码如下：

```
# 计算PageRank
    digraph = networkx.DiGraph()
    for url, url_list in url_url_list_dict.items():
        for _url in url_list:
            if _url in title_url_df.url.values:
                digraph.add_edge(url, _url)
    result = networkx.pagerank(digraph, alpha=0.85)
    page_rank_df = pd.Series(result, name='page_rank')
    page_rank_df = page_rank_df.apply(lambda x: math.log(x * 10000, 10) + 1)  # 将page_rank列所有数值*10000后，取10的对数，再+1保证数值大于1（用于缩小PageRank极差，避免权重过大）
```

在这里还需要注意一个细节：需要控制PageRank的数值范围，如代码块中注释所写，否则会造成PageRank和余弦相似度相乘后极差过大的问题。经过如上处理后，可以将PageRank控制在1-6之间，近似类比百度的0-9权重。

将PageRank计算结果存储到pagerank.csv中

```
url,page_rank
http://news.nankai.edu.cn/ywsd/system/2022/08/03/030052347.shtml,1.00057944569
http://news.nankai.edu.cn/index.shtml,3.59579684997601
http://news.nankai.edu.cn/ywsd/index.shtml,3.59579684997601
http://news.nankai.edu.cn/mtnk/index.shtml,3.6280993687224004
http://news.nankai.edu.cn/gynk/index.shtml,3.59579684997601
http://news.nankai.edu.cn/nkrw/index.shtml,3.59579684997601
http://news.nankai.edu.cn/nkdxb/index.shtml,3.59579684997601
http://news.nankai.edu.cn/sp/index.shtml,3.6280993687224004
http://news.nankai.edu.cn/gb/index.shtml,3.59579684997601
http://news.nankai.edu.cn/wx/system/2019/06/03/030033750.shtml,3.5767133801556
http://news.nankai.edu.cn/wx/index.shtml,3.576713380155614
http://news.nankai.edu.cn/ywsd/system/2022/11/28/030053838.shtml,3.57671338015
http://news.nankai.edu.cn/ywsd/system/2022/11/29/030053867.shtml,3.57671338015
http://news.nankai.edu.cn/ywsd/system/2022/11/26/030053830.shtml,3.57671338015
http://news.nankai.edu.cn/ywsd/system/2022/11/29/030053860.shtml,3.57671338015
http://news.nankai.edu.cn/ywsd/system/2022/11/30/030053874.shtml,3.57671338015
```

4. 查询服务

完成上述模块后，为用户提供**站内查询、短语查询、通配查询、查询日志、**网页**快照**等高级搜索功能。

我选择完成的功能有：**通配查询、站内查询、时间查询、快照服务、标题检索、搜索历史**

基础搜索页面如下：

高级搜索页面如下：

在我的实现中，算法会先做一轮基础查询，筛选出有用的信息，然后再进行高级查询。搜索历史以cookie的形式存储。

cookie作为现在http通讯中必备的组成部分，不需要额外的代码即可在用户和后端之间传输，兼顾简便性和用户无感原则。同时cookie还可以设置过期时间，用户在一段时间不使用的情况下可能需要搜索的侧重点有所变化，cookie的过期可以贴合用户的使用场景。

4.0 基础查询

即使用向量空间模型进行基础的搜索。

- 输入：关键词、查询历史（从cookie）中读取、查询模式（标题索引or全文索引）
- 输出：形式为`(url, similarity)`的一个list

具体实现步骤为：

1. 对输入关键词和历史记录进行分词，并处理空格
2. 判断搜索模式：是全文搜索还是标题搜索
3. 计算每一篇文章、关键词、历史记录的tf-idf
4. 进行余弦相似度的计算

```
def main(input_word: str, history_words: list, is_title_only: bool = False) ->
list[tuple[str, float]]:
    """
    Args:
        input_word: keyword
        history_words: history words
        is_title_only: search mode

    Returns:
        [(url, similarity), ...]
    """
    # 对输入的关键词进行分词
    split_input = list(cut_for_search(input_word))
    split_input.sort()
    if '' in split_input:
        split_input.remove('')
    if ' ' in split_input:
        split_input.remove(' ')

    # 对历史记录进行分词
    split_history = []
    for i in range(len(history_words)):
        temp_split = list(cut_for_search(history_words[i]))
        for i in temp_split:
            if i in ['', ' ']:
                pass
            elif i not in split_history:
                split_history.append(i)

    # 判断搜索模式，全文搜索或标题搜索
    if not is_title_only:
        tf_dict = tf
        idfs = idf
        word_sets = word_set
    else:
        tf_dict = tf_title_only
        idfs = idf_title_only
        word_sets = word_set_title_only

    tfidf_dict = {}  # 存储每一篇文档的向量(tf-idf)
    for k, v in tf_dict.items():
        tfidf_dict[k] = computeTFIDF(v, idfs)
```

```
    key_tfidf_dict = {}  # 存储关键词的tfidf。筛选出tf-idf最大的前key_valid_number个词，降序排列
    for k, v in tfidf_dict.items():
        key_tfidf_dict[k] = sorted(tfidf_dict[k].items(), key=lambda d: d[1], reverse=True)[:key_valid_number]  # d.items() 以列表的形式返回可遍历的元组数组
    key_tfidf_list = list(key_tfidf_dict.values())  # 将结果转化为list，方便后续调用
    key_tfidf_url_list = list(key_tfidf_dict.keys())  # 将结果转化为list，方便后续调用
    len_key_tfidf_url_list = len(key_tfidf_url_list)

    tf_input = computeTF(word_sets, split_input)  # 查询的tf
    tfidf_input = computeTFIDF(tf_input, idfs)  # 查询的tf-idf
    key_input = sorted(tfidf_input.items(), key=lambda d: d[1], reverse=True)[:key_valid_number]  # 查询的前100个关键词
    len_key_input = length(key_input)

    tf_history = computeTF(word_sets, split_history)  # 历史记录的tf
    tfidf_history = computeTFIDF(tf_history, idfs)  # 历史记录的tf-idf
    key_history = sorted(tfidf_history.items(), key=lambda d: d[1], reverse=True)[:key_valid_number]  # 历史记录的前100个关键词
    len_key_history = length(key_history)

    # 余弦相似度计算
    key_results = []
    key_results_index = []  # 用于存储index，方便history_words的调用
    for i in range(len_key_tfidf_url_list):  # 遍历每个文档
        num = 0
        _key_tfidf_list = key_tfidf_list[i]
        for _key_input in key_input:  # 遍历每个关键输入词
            if _key_input[1] != 0:  # 如果输入词的tf-idf不为0，才计算
                for __key_tfidf_list in _key_tfidf_list:  # 遍历每个文档内的每个关键词
                    if _key_input[0] == __key_tfidf_list[0]:  # 若为相同单词
                        num = num + _key_input[1] * __key_tfidf_list[1]
        cos = round(num / (len_key_input * length(_key_tfidf_list)), 4)
        key_results.append((key_tfidf_url_list[i], cos))  # 存储第i个文档的余弦相似度
        if cos > 0:
            key_results_index.append(i)

    if len(history_words) > 0:  # 没有历史记录时不计算历史记录的相似度
        history_results_dict = {}
        for i in key_results_index:  # 遍历每个文档
            num = 0
            _key_tfidf_list = key_tfidf_list[i]
            for _key_history in key_history:  # 遍历每个关键输入词
                if _key_history[1] != 0:  # 如果输入词的tf-idf不为0，才计算
                    for __key_tfidf_list in _key_tfidf_list:  # 遍历每个文档内的每个关键词
                        if _key_history[0] == __key_tfidf_list[0]:  # 若为相同单词
                            num = num + _key_history[1] * __key_tfidf_list[1]
            history_results_dict[i] = ((key_tfidf_url_list[i], (round(num / (len_key_history * length(_key_tfidf_list)), 4))))  # 存储第i个文档的余弦相似度

        results = []
        for i in range(len_key_tfidf_url_list):
            if j := history_results_dict.get(i):  # 海象运算符，若j不为空，则执行下面的语句
```

```
                results.append((key_results[i][0], key_results[i][1] + j[1] /
10))  # 历史记录的权重为1/10
            else:
                results.append((key_results[i][0], key_results[i][1]))
        results = sorted(results, key=lambda d: d[1], reverse=True)
    else:
        results = []
        for i in range(len_key_tfidf_url_list):
            results.append((key_results[i][0], key_results[i][1]))
        results = sorted(results, key=lambda d: d[1], reverse=True)

    return_list = []
    for result in results:
        if result[1] > 0:
            return_list.append((result[0], result[1]))
    return return_list
```

4.1 时间查询

用于时间的限制，即时间范围。可选的时间范围有：

- 一天内
- 一周内
- 一个月内
- 一年内

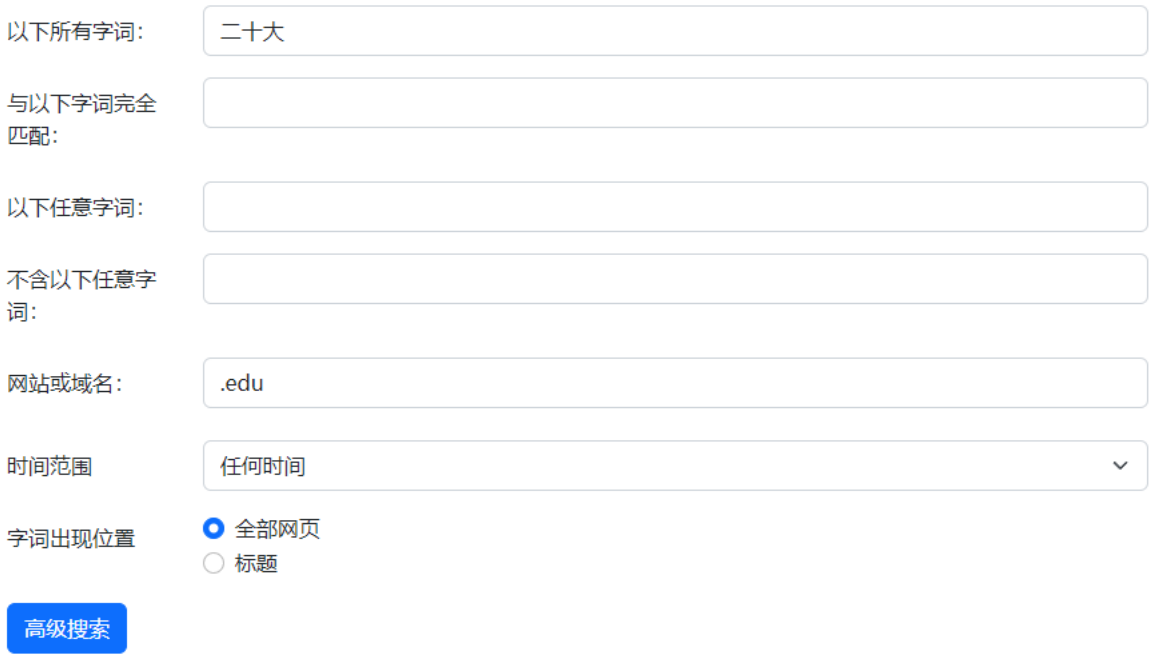
这里利用了python的datetime标准库进行时间判断。如果超出时间限制，则直接return，并跳出循环。

```
# 0. 时间限制
    if text_line['date_timestamp'] != '':
        result_datetime =
datetime.fromtimestamp(int(text_line['date_timestamp']))  # 时间戳转换为datetime
        if time_limit == '一天内':
            if datetime.now() - result_datetime > timedelta(days=1):
                return result  # 如果超过时间限制，就返回这个结果方便删除，并跳出循环
        elif time_limit == '一周内':
            if datetime.now() - result_datetime > timedelta(days=7):
                return result  # 如果超过时间限制，就返回这个结果方便删除，并跳出循环
        elif time_limit == '一个月内':
            if datetime.now() - result_datetime > timedelta(days=30):
                return result  # 如果超过时间限制，就返回这个结果方便删除，并跳出循环
        elif time_limit == '一年内':
            if datetime.now() - result_datetime > timedelta(days=365):
                return result  # 如果超过时间限制，就返回这个结果方便删除，并跳出循环
    else:
        if time_limit != '任何时间':  # 如果没有时间戳，那么默认超过时间限制
            return result  # 如果超过时间限制，就返回这个结果方便删除，并跳出循环
```

4.2 站内查询

用于搜索某个网站，或将搜索结果限制为特定域名类型

例如限定域名类型为edu.



搜索结果：

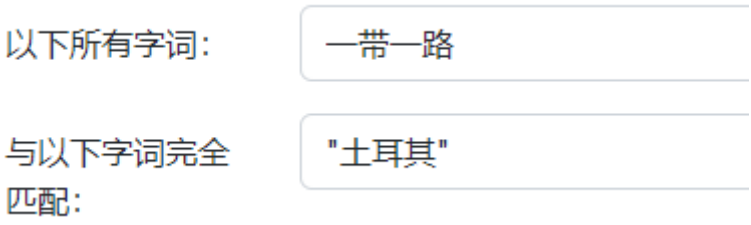
其主要思路是先做基础查询，然后再根据网站或域名进行筛选。如果不是指定的网站或域名，就返回这个结果方便删除，并跳出循环。

```
    # 1. 网站或域名
    if site_or_domain:
        if site_or_domain not in result[1]:
            return result  # 如果不是指定的网站或域名，就返回这个结果方便删除，并跳出循环
```

4.3 通配查询

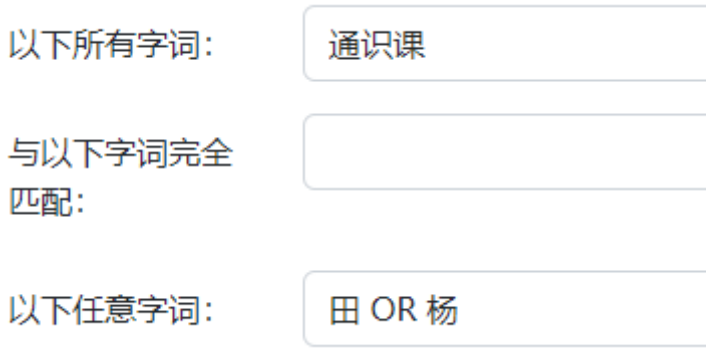
如查询：**与某个字符完全匹配、包含以下任意字符、不含以下任意字符**

- 与某个字符完全匹配



“一带一路”中土语言文化经贸交流学术研讨会云端举办-南开要闻-南开大学
南开新闻网讯（通讯员王柳青）9月20日，为庆祝土耳其眸迪特派大学孔子学院成立五周年，由南开大学、眸迪特派大学和天津市人民对外友好协会共同举办的一带一路中土语言文化经贸交流学术研讨会于线上举办。中国驻土耳其大使馆公参李勤、中国驻伊斯坦布尔代总领事吴健、土耳其驻华使馆教育参赞蒋苏、天津市人民对外友好...
http://news.nankai.edu.cn/ywsd/system/2022/09/21/030052895.shtml 网页快照

- 包含以下任意字符



找到约 3 条结果 （用时 0.46 秒） 高级搜索

南开师生纪念杰出校友周恩来总理诞辰124周年-南开要闻-南开大学
南开新闻网讯（通讯员杨奇张宏思毕昕悦摄影姜志凯朱轩）3月5日是南开杰出校友周恩来总理诞辰124周年纪念日。当天上午，南开大学师生代表在八里台校区、津南校区周恩来总理塑像前开展传承恩来精神，谱写时代新篇纪念活动。校学生会、研究生会代表，第十一届周恩来班师生代表，部分周恩来奖学金获得者，《周恩来...
http://news.nankai.edu.cn/ywsd/system/2022/03/05/030050491.shtml 网页快照

我校3位教师荣获霍英东教育基金会第18届高等院校青年科学奖和教育教学奖-南开要闻-南开大学
南开新闻网讯（通讯员杨丽新）近日，庆祝香港回归祖国25周年霍英东教育基金会第18届高等院校青年科学奖及教育教学奖颁奖活动在北京和香港连线举行。教育部党组书记、部长怀进鹏和香港特别行政区行政长官李家超在大会上讲话。教育部副部长田学军总结致辞并为获奖代表颁奖。香港教育局局长蔡若莲，霍英东教育基金会理...
http://news.nankai.edu.cn/ywsd/system/2022/08/02/030052336.shtml 网页快照

南开大学举办纪念“一二•九”爱国运动86周年系列活动-南开要闻-南开大学
南开新闻网讯（通讯员杨奇记者郝静秋）为纪念一二九运动86周年，弘扬爱国主义精神，厚植爱国主义情怀，12月9日，南开大学举办了答题接力跑、师生同讲党史学习教育思政微课、纪念一二九主题团日活动、征集百个团支部代言家乡产品等系列纪念活动。当天上午，由化学学院举办的纪念一二九爱国运动86周年线上答题接...
http://news.nankai.edu.cn/ywsd/system/2021/12/09/030049378.shtml 网页快照

- 不包含以下任意字符

【专题】学习贯彻党的二十大精神-系列专题-南开大学

★《光明日报》整版聚焦南开：在新征程上勇攀高峰争创一流 ★《光明日报》刊发杨庆山陈雨露署名文章：为中国式现代化贡献南开力量 ★校领导为新传学院师生讲授形势与政策课 ★杨庆山为全校中层干部宣讲党的二十大精神 ★校领导为三学院师生讲授形势与政策课 ★南开大学党...

http://news.nankai.edu.cn/xlzt/system/2022/10/04/030053022.shtml 网页快照

【学习贯彻二十大】校领导为三学院讲授“形势与政策”课-南开要闻-南开大学

南开新闻网讯（通讯员吕沛繁韩佳穆摄影邹风博）11月24日，副校长李靖以深入学习贯彻党的二十大精神，与党同心同德、与时代同向同行为题，为金融学院、环境科学与工程学院和材料科学与工程学院的全体本科生讲授形势与政策课，近两千名师生以线下和线上直播的形式同时参加。李靖指出，党的二十大是在关键历史时...

http://news.nankai.edu.cn/ywsd/system/2022/12/02/030053896.shtml 网页快照

【学习贯彻二十大】校领导为两院师生讲授“形势与政策”课-南开要闻-南开大学

南开新闻网讯（通讯员邱辰霄崔丽月）12月1日，副校长李靖以深入学习贯彻党的二十大精神，与党同心同德、与时代同向同行为主题，为外国语学院和物理科学学院全体本科生讲授形势与政策课。李靖深入浅出地阐述了党的二十大主题、取得的主要成果、具有的深远意义和重大影响。他强调，党的二十大报告主题高度凝练，...

http://news.nankai.edu.cn/ywsd/system/2022/12/02/030053903.shtml 网页快照

十位政治学名家同上一门“名师引领”通识课-南开要闻-南开大学

南开新闻网讯（通讯员郭道久于慧）这个学期，南开大学首开的名师引领政治学通识通识选修课，邀请了10位来自国内一流高校政治学领域的名师围绕学科前沿问题授课，成为了师生热议的网红课。南开大学讲席教授朱光磊、南开大学副校长王新生、四川大学公共管理学院教授姜晓萍、华中师范大学资深教授徐勇、吉林大学...

http://news.nankai.edu.cn/ywsd/system/2021/12/24/030049695.shtml 网页快照

“名师引领”通识课：阎锡蕴院士讲述发现纳米酶背后的故事-南开要闻-南开大学

南开新闻网讯（通讯员万金鹏）5月25日，中国科学院院士、中国科学院生物物理所、中国科学院纳米酶工程实验室主任、中国科学院生物物理研究所蛋白质与多肽药物所重点实验室主任阎锡蕴教授以发现纳米酶背后的故事科学发现的偶然与必然为题，为南开师生讲授了名师引领通识课之生物科学现状与未来的第十四讲。课程通过线...

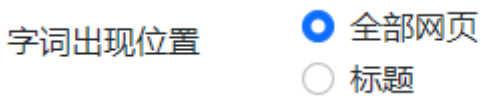
不再有“田志刚”相关内容出现。

```
    # 1. 网站或域名
    if site_or_domain:
        if site_or_domain not in result[1]:
            return result  # 如果不是指定的网站或域名，就返回这个结果方便删除，并跳出循环

    # 2. 与以下字词完全匹配
    if this_exact_word_or_phrase:
        this_exact_word_or_phrase_list = re.findall(r'\"(.+?)\"',
this_exact_word_or_phrase)  # 用正则表达式提取双引号中的内容
        for word in this_exact_word_or_phrase_list:
            if word == 'X':
                pass  # 避免用于分隔的特殊字符被误判为匹配
            if word not in text:
                return result  # 如果全部文本中没有完全匹配的词，就返回这个结果方便删除，
并跳出循环

    # 3. 以下任意字词
    if any_of_these_words:
```

```
        any_of_these_words_list = any_of_these_words.split('OR')
        if '' in any_of_these_words_list:
            any_of_these_words_list.remove('') # 删除空字符串
        for word in any_of_these_words_list:
            if word == '✗':
                pass  # 避免用于分隔的特殊字符被误判为匹配
            if word in text:
                return '' # 如果全部文本中有任意匹配的词，就跳出循环
            if word == any_of_these_words_list[-1]:
                if word not in text:
                    return result # 如果全部文本中没有任意匹配的词，就返回这个结果方便删
除，并跳出循环

    # 4. 不含以下任意字词
    if none_of_these_words:
        none_of_these_words_list = none_of_these_words.replace('\"',
'').split('-')  # 提取-分隔的内容
        if '' in none_of_these_words_list:
            none_of_these_words_list.remove('')  # 删除空字符串
        for word in none_of_these_words_list:
            if word == '✗':
                pass  # 避免用于分隔的特殊字符被误判为匹配
            if word in text:
                return result  # 如果全部文本中出现了完全匹配的词，就返回这个结果方便删
除，并跳出循环
```

4.4 标题检索

标题检索即限定是在全部网页中检索还是在标题中检索。



具体实现是在`main()`函数中设置一个`is_title_only`的判断，如果为真，则使用预处理文件中`$title_only.json`相关的文件，否则会使用所有的信息。

```
    # 判断搜索模式，全文搜索或标题搜索
    if not is_title_only:
        tf_dict = tf
        idfs = idf
        word_sets = word_set
    else:
        tf_dict = tf_title_only
        idfs = idf_title_only
        word_sets = word_set_title_only

# 读取title_only相关
with open(os.path.join(path, 'inverted_index_title_only.json'), 'r',
encoding='utf-8') as f:
    inverted_index_title_only = json.load(f)
with open(os.path.join(path, 'word_frequency_title_only.json'), 'r',
encoding='utf-8') as f:
    word_frequency_title_only = json.load(f)
    word_set_title_only = sorted(set(word_frequency_title_only.keys()))
with open(os.path.join(path, 'word_idf_title_only.json'), 'r', encoding='utf-8')
as f:
```

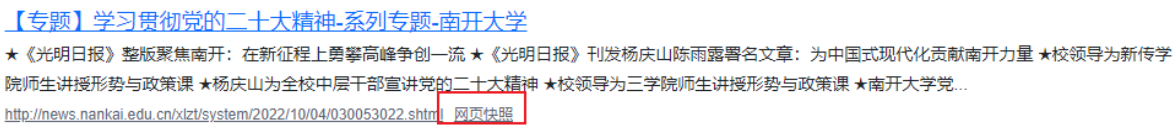
```
    idf_title_only = json.load(f)
with open(os.path.join(path, 'tf_title_only.json'), 'r', encoding='utf-8') as f:
    tf_title_only = json.load(f)
with open(os.path.join(path, 'tf-idf_title_only.json'), 'r', encoding='utf-8')
as f:
    tf_idf_title_only = json.load(f)
```

4.5 搜索历史

这里使用flask库内置方法来操作cookie，实现搜索历史的记录和用户与服务端时间的交互传输。

```
        if all_these_words not in search_history:
            search_history.append(all_these_words)  # 将搜索关键词添加到历史记录中
        if len(search_history) > 12:
            search_history.pop(0)  # 如果历史记录超过12条，则删除最早的一条
        resp.set_cookie('search_history', json.dumps(search_history), max_age=60
* 60 * 24 * 30)  # 设置cookie,有效期为30天

        # 从cookie中获取搜索历史
        if request.cookies.get('search_history'):
            search_history: list =
json.loads(request.cookies.get('search_history'))  # 从cookie中获取搜索历史
        else:
            search_history = []
```

4.6 快照服务



因为爬虫记录了原始网页的html代码，这里直接利用flask内置的jinja2模板，重新将html代码渲染即可还原网页构成快照。按照搜索引擎常见的网页快照技术的逻辑，html代码中链接的资源（如js、css、jpg、png等），从原始地址获取，如果无法加载则不会在页面上体现。

```
@front.route('/snapshot')
def _snapshot():
    if url := request.args.get('url'):
        title = url_title_df.loc[url]['title']
        with open(rf'./tools/pages/{title}.html',encoding='utf-8') as f:
            snapshot = f.read()
        # 向前端以网页的形式返回快照
        return render_template(r'snapshot.html', snapshot=snapshot)
    else:
        return "不合法的参数"
```

5. 个性化查询

个性化查询为不同的用户提供不同的内容排序。可以实现一个账号登录系统，通过用户完善的年龄性别等个人信息为其呈现不同的查询结果；或者是记录用户的查询历史，通过历史查询来提供个性化的查询结果。

在我的实现中，我选择使用历史记录来优化查询结果。处理步骤为：

1. 将历史记录集合在一起分词

2. 计算历史记录与其他文档的余弦相似度
3. 加权到之前计算的检索词与各文档的余弦相似度结果中。在这里，**权重被设置为0.1**
4. 按照余弦相似度降序排序

权重设置只是更简单的一种。个性化查询是当今信息检索的一个重要研究方向，有许多更优化的算法，此处不再赘述。

```
if len(history_words) > 0:  # 没有历史记录时不计算历史记录的相似度
        history_results_dict = {}
        for i in key_results_index:  # 遍历每个文档
            num = 0
            _key_tfidf_list = key_tfidf_list[i]
            for _key_history in key_history:  # 遍历每个关键输入词
                if _key_history[1] != 0:  # 如果输入词的tf-idf不为0，才计算
                    for __key_tfidf_list in _key_tfidf_list:  # 遍历每个文档内的每个
关键词
                        if _key_history[0] == __key_tfidf_list[0]:  # 若为相同单词
                            num = num + _key_history[1] * __key_tfidf_list[1]
            history_results_dict[i] = ((key_tfidf_url_list[i], (round(num /
(len_key_history * length(_key_tfidf_list)), 4))))  # 存储第i个文档的余弦相似度

        results = []
        for i in range(len_key_tfidf_url_list):
            if j := history_results_dict.get(i):  # 海象运算符，若j不为空，则执行下面
的语句
                results.append((key_results[i][0], key_results[i][1] + j[1] /
10))  # 历史记录的权重为1/10
            else:
                results.append((key_results[i][0], key_results[i][1]))
        results = sorted(results, key=lambda d: d[1], reverse=True)
    else:
        results = []
        for i in range(len_key_tfidf_url_list):
            results.append((key_results[i][0], key_results[i][1]))
        results = sorted(results, key=lambda d: d[1], reverse=True)
```

6. Web页面（图形化页面）

在这里，选择使用flask库内置并官方推荐的Jinja2进行网页的渲染。

jinja2是Flask作者开发的一个模板系统，起初是仿django模板的一个模板引擎，为Flask提供模板支持，由于其灵活，快速和安全等优点被广泛使用。

jinja2之所以被广泛使用是因为它具有以下优点：

1. 相对于Template，jinja2更加灵活，它提供了控制结构，表达式和继承等。
2. 相对于Mako，jinja2仅有控制结构，不允许在模板中编写太多的业务逻辑。
3. 相对于Django模板，jinja2性能更好。
4. Jinja2模板的可读性很棒。

常规的方案是，把表现逻辑-响应文本维护到单个文件，通过渲染模块 (render_template) 把html中需要的数据从视图函数中传递过去。默认情况下，Flask在项目文件夹的子文件夹templates寻找模板文件，所以分离出来的模板文件都需保存到/templates。

在jinja2中，存在三种语法：

1. 控制结构 {% %}
2. 变量取值 {{ }}
3. 注释

总的来说，模板是一个包含响应文本的文件，其中包括需要从响应获取的动态数据部分。

我的图形化界面选择仿照的是google的搜索页面(www.google.com)

例如，主页的模板代码为：

```
{% extends "base.html" %}

{% block page_content %}
    <div class="container">
        <div class="row">
            <div class="col-md-12" style="padding: 5rem;text-align:center;">
                <form action="{{ url_for('front._result_page') }}" method="get">
                    <div class="input-group" style="font-size: 14px;">
                        <label style="width:100%;vertical-align:middle;">
                            <input type="text" class="form-control"
name="keywords" placeholder="请输入关键词" style="width: 100% ;max-width:
584px;border-radius: 24px;margin: 0 auto">
                        </label>
                    </div>
                    <div class="input-group-append" style="padding:5px">
                        <button id="search-button" class="btn btn-outline-
secondary" type="submit">立即搜索</button>
                        <button class="btn btn-outline-secondary"
onClick="window.location.href='{{ url_for('front._advanced_search') }}'"
type="button">高级模式</button>
                    </div>
                </form>
                <br/>
                {% if search_history %}
                    <div class="row">
                        <div class="col-md-12" style="max-width:584px;margin: 0
auto"><h6 style="text-align:left">搜索历史</h6></div>
                        <br/>
                        <div class="row">
                            <div style="max-width:584px;margin: 0
auto;float:left;position: relative;">
                                {% for i in search_history %}
                                    <div style="padding: 5px;float:left;">
```

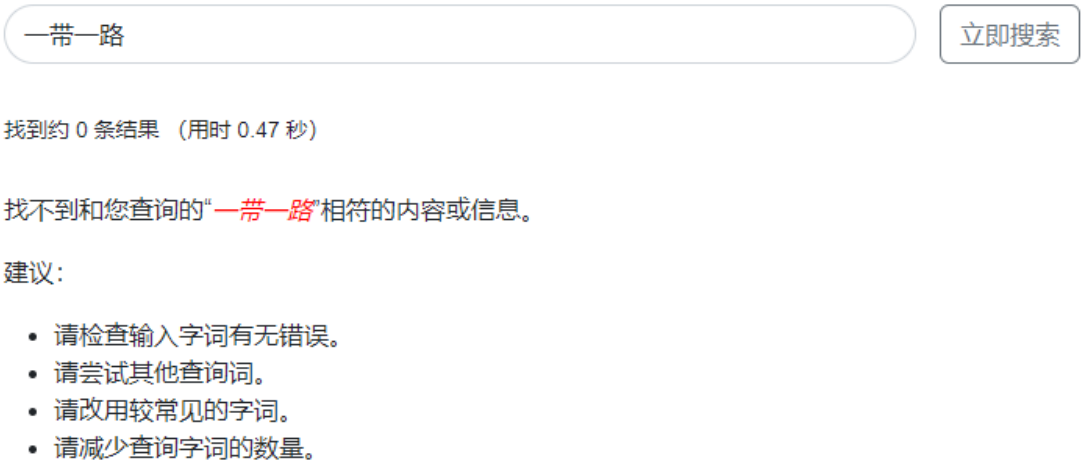
```
                                        <a href="{{
url_for('front._result_page', keywords=i) }}" class="btn btn-outline-secondary"
                                           style="width: 100% ;max-width:
584px;border-radius: 24px;margin: 0 auto">{{ i }}</a>
                                    </div>
                                {% endfor %}
                            </div>
                        </div>
                    </div>
                {% endif %}
            </div>
        </div>
    </div>

{% endblock %}
```

这里一共有6个模板页面，分别为：

- base.html，为基础模板
- index.html

- advanced_search.html

- no_result_page.html



- result_page.html

- snapshot.html 即为快照页

7. 个性化推荐

个性化推荐系统通过用户的个人信息和查询历史获取用户可能的兴趣点，在用户查询时给用户推荐相关领域的其他内容。比如在百度上搜索 iphone，其会在查询结果的右侧为你推荐ipad、iMac 等相关产品。

在我最初的想法中，我认为可以使用一个联想词库来实现，其大致思路为：

1. 使用联想词库获取检索词的相关词
2. 对相关词再进行信息检索，获得余弦相似度
3. 再使用搜索历史对相关词的余弦相似度加权，并重新排序

但是后来发现，我无法在互联网内获取这个“相关词库”。因此我转变了一下思路，并使用了一个trick：直接调用百度的搜索推荐API，并显示在右侧页面。

这边使用了flask2.0版本之后新增加的异步特性，保证flask进程在请求百度API时不会被IO阻塞，同时请求也需要通过异步发送，使用await关键词返回结果。

相关代码为：

```
// suggest.js
function query_suggest(keywords){
        suggestion_dom = document.getElementById('suggestion');
        $.ajax({
            type : "get",
            async: true,
            url : "/suggest?keywords="+keywords,
            success : function(data){
                console.log(data)
                var tag = '';
                for(var i=0;i<data.length;i++){
                    tag += '<li>'+data[i]+'</li>';
                }
                suggestion_dom.innerHTML = tag;
            },
            error:function(){
                console.log('fail');
            }
        });
    }
```

`suggest.js` 调用了jQuery的网络交互库ajax，为了保证页面的加载速度和使用体验，ajax使用了 `async: true` 参数，以异步的方式渲染在页面上，保证页面加载不被请求百度API的网络IO阻塞。

之后请求flask后端的API接口：

```
@front.route('/suggest')
async def _suggest():
    keywords = request.args.get('keywords')
    if keywords:
        try:
            async with httpx.AsyncClient(timeout=5) as client:
                r = await client.get(f'https://www.baidu.com/sugrec?prod=pc&wd=
{keywords}', headers=headers) # 利用了百度的搜索建议接口
                return_list = [i['q'] for i in r.json()['g']] # 返回的是json格式的
数据，需要用json模块解析

        except Exception as e:
            return_list = []
    else:
        return_list = []
    return jsonify(return_list)
```

总结与感想

最后一次大作业对我来说十分具有挑战性，它意味着不仅要集合所有所学知识，还需要编写出一个完整的可实现的应用。对于倒排索引和查询这两个部分，由于之前作业已经涉及过，因此我比较熟悉；但是我觉得最难的还是web的搭建与前后端交互，因为这是我从未涉及过的场景。或许这让我的重心有所偏移，但实际上我从这次大作业也学到了许多实用的知识。

在个性化搜索与个性化推荐这两个项目上我花费了较多时间，特别是个性化推荐，无奈之下直接调了百度的推荐API。和助教略微交流后感觉可以使用数据挖掘技术，后续可以再进行优化。

感谢我的朋友在flask库与爬虫的应用上给予我的技术支持，让我得以顺利地完成了这次作业；也很感谢老师的悉心教导与助教与我的细心讨论。总体来说，我感悟颇多，收获丰厚！