南开大学
Nankai University

南　　开　　大　　学

信息检索系统原理——Web 搜索引擎

孟启轩

学号：2212452

专业：计算机科学与技术

指导教师：温延龙

2024 年 12 月 16 日

摘要

本实验基于信息检索系统原理课程所学，自主设计并实现了一个南开大学 Web 搜索引擎。结合 Elastic Search 实现了网页抓取与数据采集，采用先进的文本索引技术和链接分析算法（PageRank）对网页内容和结构进行深入解析与处理。搜索引擎支持多种查询服务，包括短语查询、通配符查询、查询日志等，显著提升了搜索的灵活性和精准度。此外，为了确保良好的用户交互，还设计了直观友好的 Web 页面界面以及个性化推荐功能，也尝试设计了个性化查询的功能。

关键字：信息检索，南开大学，搜索引擎，Python

目录

一、 项目综述

设计项目最初想要通过自己一步一步将整个 Web 搜索引擎搭建起来，但是在构建索引以及向量空间模型等步骤难度比较大，正确性也难以验证。所以经过几天的尝试后，决定结合 Elastic Search 这个强大的工具。系统架构如图1所示。

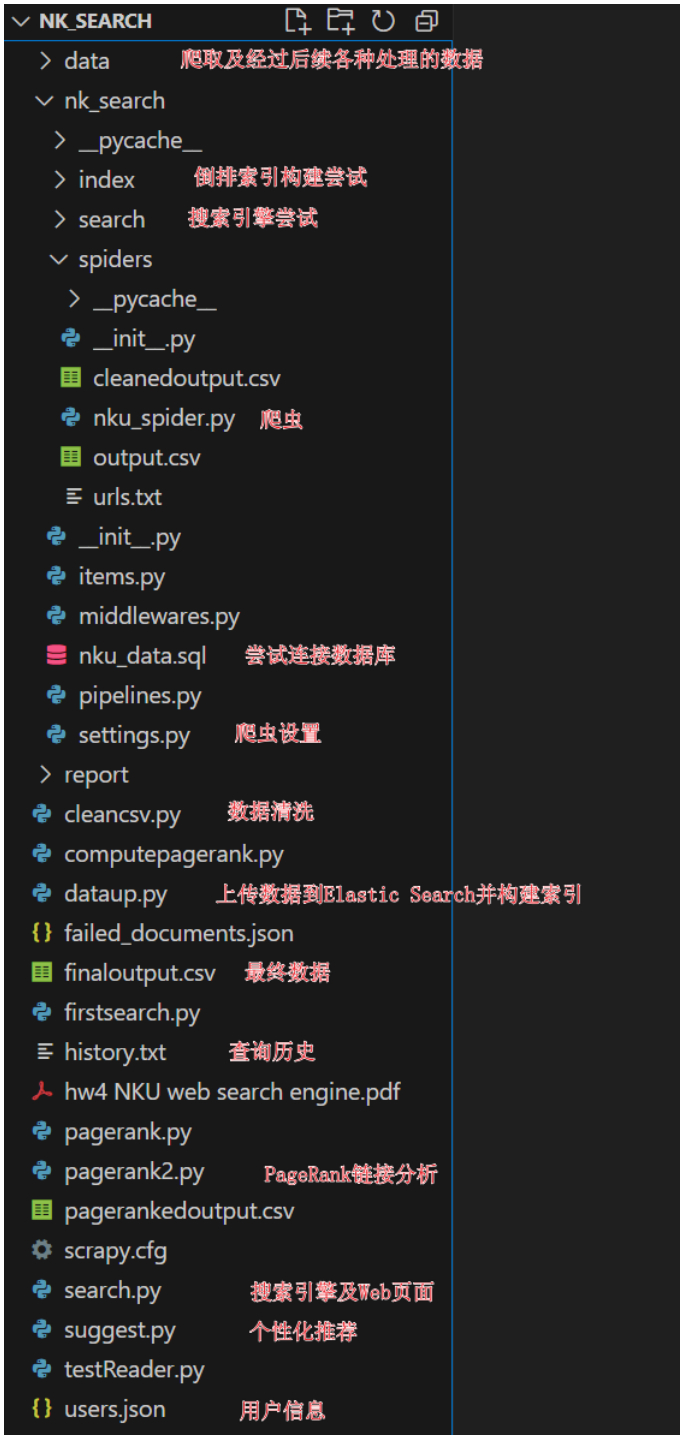

图 1: 系统架构

二、 网页抓取

(一) 设计思路

1. 爬虫目标

- **目标域：** nankai.edu.cn
- **起始网站：** 南开大学官方网站 http://www.nankai.edu.cn/
- **抓取内容：** 网页标题（title）、URL（url）、正文内容（text）、页面内链接（linksurl）。
- **数据存储：** 将抓取的数据存储至 CSV 文件，以便后续导入至 Elastic Search 进行索引构建。
- **反馈优化模块：** 通过用户反馈优化检索和生成策略。

2. 爬虫功能需求

- **全面抓取：** 遍历目标网站的可访问页面，确保覆盖主要内容。
- **数据清洗：** 提取有效文本，去除无关内容（如脚本、样式）。
- **链接管理：** 处理相对路径链接，确保抓取范围限定在指定域名内。
- **效率优化：** 通过并发请求和自动限速，提高抓取速度，同时避免对目标服务器造成过大负载。
- **数据存储：** 结构化存储抓取数据，便于后续处理和分析。

(二) 技术路线

1. 爬虫架构设计与实现细节

- **爬虫框架：** Scrapy，高效、灵活，支持异步抓取，适合大规模数据抓取。
- **解析方法（parse）：** 主要负责从网页中提取并处理信息。具体来说，它使用 XPath 技术来选取网页的 <title> 标签内容作为标题，并挑选 <body> 标签内的文本节点（排除 <script> 和 <style> 标签）以获取正文内容，之后将这些文本合并并调用 clean_text 方法去除多余的空格和换行符。同时，该方法会通过 XPath 提取所有 <a> 标签中的 href 属性来收集链接，处理其中的相对路径为绝对 URL，并过滤只保留属于 nankai.edu.cn 域名下的链接，最终以分号分隔的形式保存在 linksurl 字段中。所提取的数据将通过调用 save_to_csv 方法被写入 CSV 文件中进行存储。此外，对于每一个有效的链接，parse 方法还会递归地发起新请求继续抓取更多页面内容，确保了数据采集的深度与广度。
- **爬虫配置：** 请求间隔（DOWNLOAD_DELAY = 0.15），爬取深度（DEPTH_LIMIT = 100），并发配置（最大并发请求数量 32，单域并发请求数限制 16），自动限速（AUTOTHROTTLE），请求头设置（User-Agent 伪装成浏览器）
- **数据存储：** 将抓取的数据存储至 CSV 文件，以便后续导入至 Elasticsearch 进行索引构建。
- **允许爬取的域：** allowed_domains = [“nankai.edu.cn”]

```python
def parse(self, response):
        # 提取网页标题
        title = response.xpath("//title/text()").get(default="Untitled")
        # 提取正文内容，仅保留有效文本
        raw_text = response.xpath("//body//*[not(self::script or self::style)]/text()"
    ).getall()
        clean_content = self.clean_text(" ".join(raw_text))
        # 提取链接
        links = response.xpath("//a[@href]/@href").extract()
        full_links = []
        for link in links:
            if link.startswith("/"):  # 处理相对路径
                link = response.urljoin(link)
            if self.allowed_domains[0] in link:  # 确保只抓取允许的域
                full_links.append(link)
        linksurl = "; ".join(full_links)  # 用分号分隔所有链接
        # 保存数据到 CSV
        self.save_to_csv(title, response.url, clean_content, linksurl)
        # 继续爬取链接
        for link in full_links:
            yield scrapy.Request(link, callback=self.parse)
```

Listing 1: 核心爬虫代码

2. 数据清洗

在爬取到数据后，发现有许多无法使用的数据，比如很多需要登陆的网页，会显示“跳转提示：访问的链接无效”等，所以对原始数据进行简单的清洗。

```python
# 删除标题为 "提示信息"或"跳转提示"或"Untitled" 的行
df = df[df["title"] != "提示信息"]
df = df[df["title"] != "跳转提示"]
df = df[df["title"] != "Untitled"]
# 删除正文为空的行
df = df[df["text"].notnull()]
# 删除包含特定无效关键词的行
invalid_keywords = ["您使用的浏览器", "系统提示","附件下载","错误的模板参数", "暂不支
    持移动端", "  ", "跳转提示","无效的文章地址","访问地址无效"]
df = df[~df["text"].apply(lambda x:any(keyword in x for keyword in invalid_keywords))]
# 去重操作：删除完全相同的数据行
df = df.drop_duplicates()
```

Listing 2: 核心数据清洗代码

(三) 完成结果

网页抓取得到的数据如图2所示，经过清洗后，共有 134879 条数据。

图 2: 数据

三、 文本索引

最初尝试使用 Whoosh 和 jieba 等工具对爬取到的数据构建倒排索引，不仅构建时间长，而且结果异常庞大，压缩后的结果也不易查看，正确性也难以验证。最终使用 Elastic Search 构建索引，不仅快速，正确性也得到保证，并且还有不错的可视化。

(一) 设计思路

- **连接：** 与 Elastic Search 集群交互。
- **高效上传数据：** 由于数据量非常大，而 Elastic Search 直接导入数据有大小限制，所以要确保大量数据能够快速、稳定地上传。
- **准确构建索引：** 设计合理的索引映射，确保数据的准确性和查询效率。
- **错误处理与日志记录：** 有效处理上传过程中可能出现的错误，并记录失败的文档以便后续分析。

(二) 技术路线

1. Elastic Search 连接

使用 Elastic Search 类连接到指定的主机（ES_HOST = 'http://localhost:9200'）。

2. 创建索引并定义映射

- **title:** text 类型，用于全文搜索。
- **url:** keyword 类型，用于精确匹配和过滤。

- **text:** text 类型，用于全文搜索。
- **linksurl:** keyword 类型，用于存储页面内的链接。
- **pagerank:** float 类型，用于存储页面的权重评分。（在链接分析部分详细解释）
- **suggest:** completion 类型，用于实现联想搜索功能。（在个性化推荐部分详细解释）

3. 批量上传数据

使用 Pandas 的 read_csv 方法以分块方式读取 CSV 文件，设置 chunksize 为 1000，以分批处理数据，提升内存利用率和上传效率。并且使用 tqdm 库显示数据上传的进度条，提供实时反馈。如图4.

4. 错误处理

- **捕获上传错误:** 在批量上传过程中，可能会遇到部分文档上传失败的情况，导致 CSV 中数据数量和索引中的文档数量不匹配。对于每个失败的文档，提取错误原因和文档 url，并将其记录到 failed_documents.json 文件中。
- **异常处理:** 通过捕捉到的错误的 url（如图3），再次清洗数据，然后通过 Elastic 控制台 DELETE /xxjs 删除之前的索引，并重新构建索引上传数据。

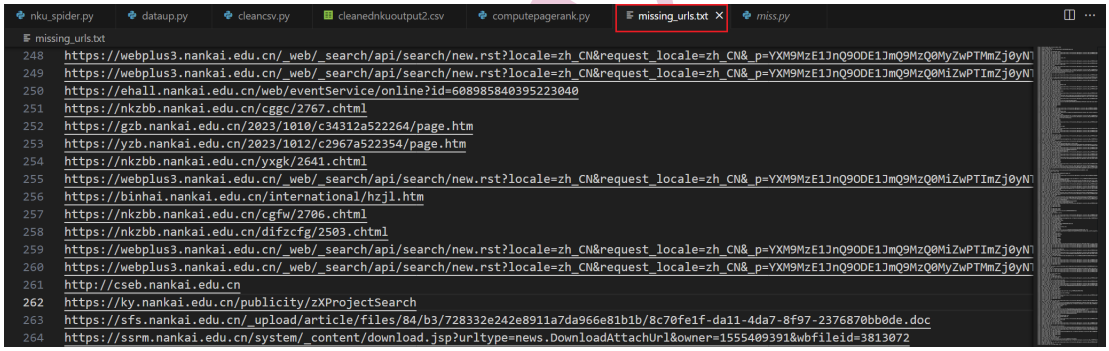

图 3: 丢失的数据

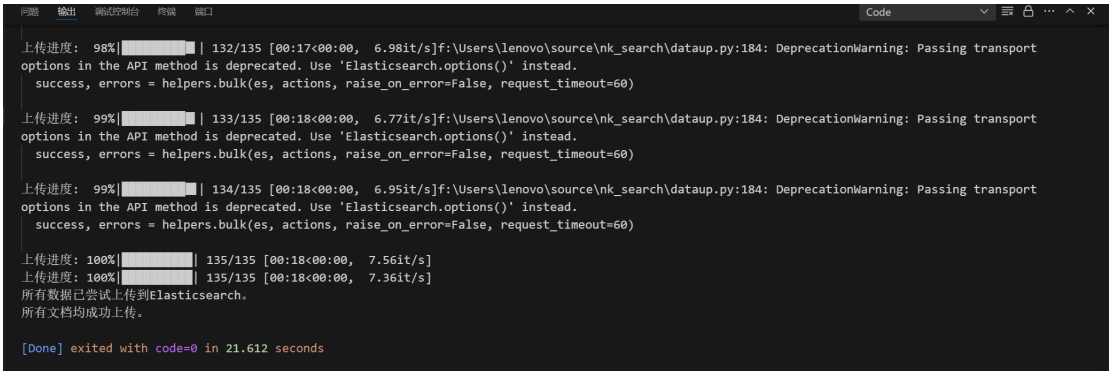

图 4: 上传数据

```python
INDEX_NAME = 'xxjs'                          # Elasticsearch索引名称
CHUNK_SIZE = 1000                            # 每批处理的行数

# 读取CSV并分批上传
chunks = pd.read_csv(CSV_FILE_PATH, chunksize=CHUNK_SIZE, encoding='utf-8')
for chunk in tqdm(chunks, total=total_lines//CHUNK_SIZE + 1, desc="上传进度"):
    # 构建批量操作
    actions = []
    for _, row in chunk.iterrows():
        action = {
            "_index": INDEX_NAME,
            "_source": {
                "title": row.get('title', ''),
                "url": row.get('url', ''),
                "text": row.get('text', ''),
                "linksurl": row.get('linksurl', ''),
                "pagerank": row.get('pagerank', ''),
                "suggest": row.get('suggest', '')
            }
        }
        actions.append(action)
```

Listing 3: 索引构建与数据上传

(三) 完成结果

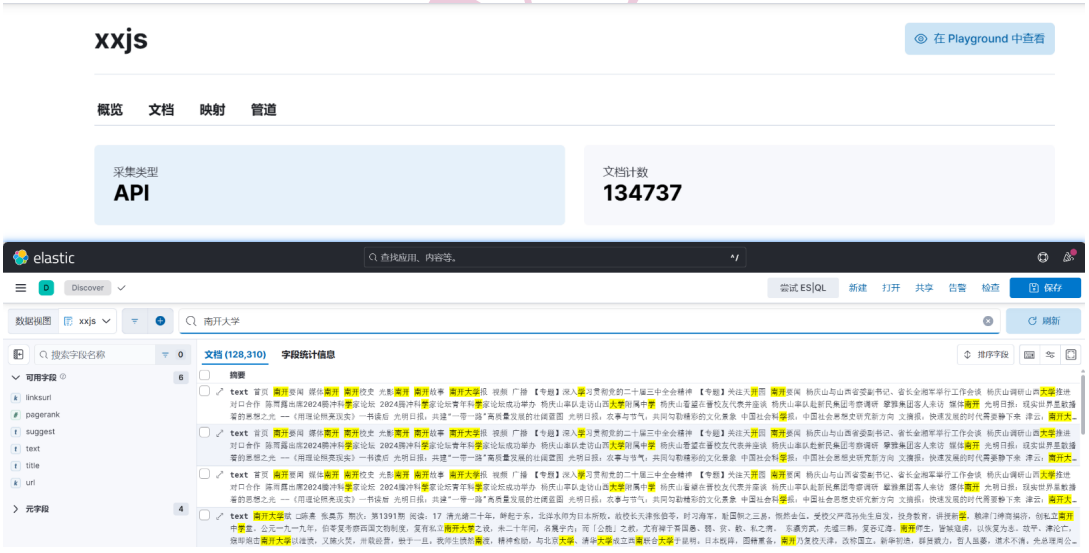

图 5: 索引

四、 链接分析

由于我的 CSV 文件数量和索引中的文档数量不一致（上传 Elastic 时有些文档出错），并且通过 CSV 计算 pagerank 的文档 id 是行号，而 Elastic 上会为每个文档生成唯一的随机 id，使得文档到 id 的映射不对应，导致所有文档的 pagerank 都是 0。

那么先计算 pagerank，然后将 pagerank 属性添加到每个文档后面，再构建索引上传数据，问题不就解决了吗。

(一) 设计思路

- **构建图结构：** 基于 CSV 中的链接数据，构建一个有向图（Directed Graph），每个网页视为一个节点，链接关系视为边。
- **计算 PageRank：** 使用 NetworkX 的 pagerank 算法计算每个节点（网页）的 pagerank 值。
- **添加 pagerank 属性到索引：** 为每个文档添加 pagerank 属性，然后重新构建索引上传数据。

(二) 技术路线

- **处理链接数据：** 在每一行数据中，提取网页的 url 和它链接的 linksurl 字段。对每个链接进行检查，若非空，则将该链接作为目标网页的指向网页，建立有向边。
- **图的构建：** 使用 NetworkX 库中的 DiGraph（有向图）数据结构。对于每一对（网页 A, 链接 B），我们在图中添加一条从网页 A 到网页 B 的有向边。
- **图的优化与限制：** 对于每条链接，进行有效性检查，确保其不为空并且不重复。若图中某些节点（网页）没有任何出链（即没有任何网页链接到它），也需在图中保留这些节点，防止图的计算出错。
- **PageRank 算法：** 使用 NetworkX 中的 pagerank 函数来计算每个节点的 pagerank 值。PageRank 算法基于以下公式：

$$PR(A) = \frac{1-d}{N} + d \sum_{i \in In(A)} \frac{PR(i)}{L(i)}$$

其中：

  - $PR(A)$ 是网页 A 的 PageRank 值。
  - d 是阻尼因子，通常设置为 0.85。
  - $In(A)$ 是指向网页 A 的网页集合。
  - $L(i)$ 是网页 i 的出链数。
  - N 是图中网页的总数。

该算法基于迭代计算，直到所有网页的 PageRank 值收敛。

```python
def compute_pagerank(csv_file_path):
    G = nx.DiGraph()    # 创建有向图

    # 读取 CSV 数据
    with open(csv_file_path, 'r', encoding='utf-8') as f:
        reader = csv.DictReader(f)

        for row in reader:
            url = row['url']
```

```
            links = row['linksurl'].split(';')  # 链接是以分号分隔的
            # 将每个网页与其链接添加到图中
            for link in links:
                link = link.strip()
                if link:  # 确保链接不为空
                    G.add_edge(url, link)

    pagerank = nx.pagerank(G, alpha=0.85)    # 计算 PageRank 值
    return pagerank
```

Listing 4: 链接分析

(三) 完成结果

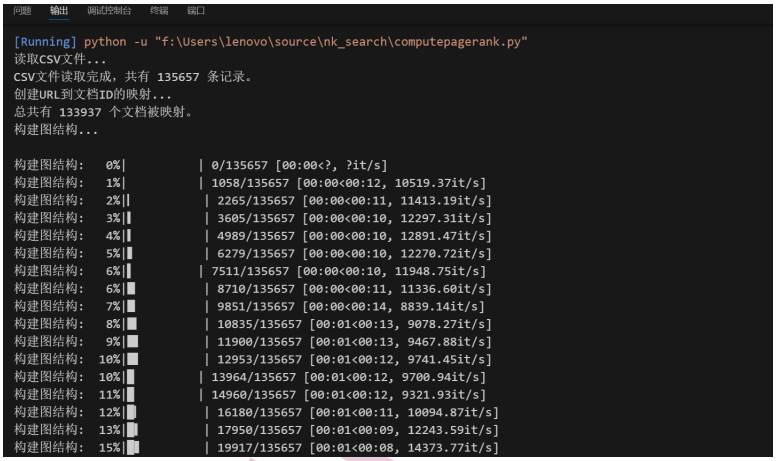

图 6: 链接分析

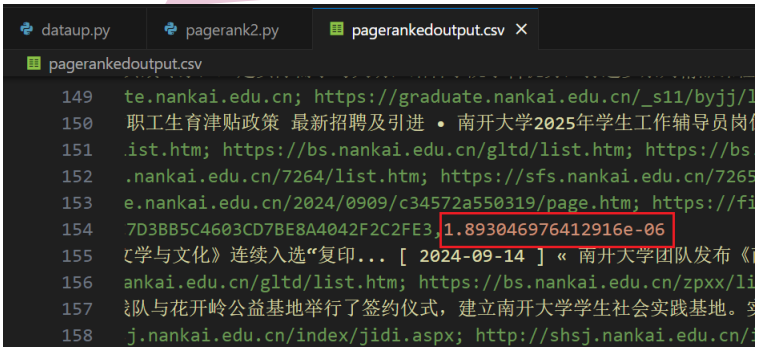

图 7: pagerank

五、 查询服务

(一) 实现功能

- **站内查询：** 基本的查询操作。
- **短语查询：** 基本的查询操作，支持对多个 Term 的查询。

- **通配查询：**用户可能并不知道自己要查什么，支持通配符（正则）查询操作。
- **查询日志：**查询日志（历史），能够知道用户之前查了什么。
- **结果评分和排序：**根据 final_score 将查询到的结果排序，返回得分高的结果。

(二) 技术路线

1. 结果评分和排序

- **查询执行与数据提取：**执行查询，获取匹配的文档，并提取所需的数据（score 和 pagerank 值）。每个文档的评分（score）是 Elastic Search 根据查询条件自动计算的，表示该文档与查询的相关度。
- **标准化处理：**因为 score 和 pagerank 并不在同一量级，所以无法直接加权组合，需要对 score 和 pagerank 值进行标准化处理，以便将它们放在相同的量纲下进行加权计算。
- **加权组合得到 final_score：**设置了 TFIDF_WEIGHT = 0.7 和 PAGERANK_WEIGHT = 0.3，使用加权平均的方式，将标准化后的 score 和 pagerank 结合起来得到 final_score，根据最终评分返回得分靠前的结果。

```
scores = [hit['_score'] for hit in hits]
pageranks = [hit['_source'].get('pagerank', 0) for hit in hits]
normalized_scores = self.normalize_values(scores)
normalized_pageranks = self.normalize_values(pageranks)
TFIDF_WEIGHT = 0.7
PAGERANK_WEIGHT = 0.3

results = []
for idx, hit in enumerate(hits):
    source = hit['_source']
    final_score = TFIDF_WEIGHT * normalized_scores[idx] + PAGERANK_WEIGHT *
    normalized_pageranks[idx]
    results.append({
        'title': source.get('title', ''),
        'url': source.get('url', ''),
        'text': source.get('text', '')[:200],  # snippet
        'pagerank': source.get('pagerank', 0),
        'final_score': final_score
    })
results.sort(key=lambda x: x['final_score'], reverse=True)
```

Listing 5: 结果评分和排序

2. 短语查询

- **查询类型：**使用 Elastic Search 的 match_phrase 查询来查找包含精确短语的文档。
- **查询过程：**用户输入查询短语，函数将其传递给 match_phrase 查询模块。然后通过 Elasticsearch 查询文档中是否包含该短语。最后返回匹配文档的结果。
- **注意：**“南开大学”和“南开 大学”的区别。

```
def search_phrase(self, phrase, user=None):
        query = {
            "query": {
                "match_phrase": {
                    "text": {
                        "query": phrase
                    }
                }
            }
        }
        return self.execute_query(query, user)
```

Listing 6: 短语查询

3. 通配查询

- **查询类型：** 使用 Elastic Search 的 wildcard 查询来实现通配符查询。
- **查询过程：** 用户输入查询字符串，允许模糊匹配。然后将查询传递给 wildcard 查询模块，搜索文本字段中符合条件的文档。最后返回匹配结果。

```
def search_wildcard(self, wildcard_query, user=None):
        query = {
            "query": {
                "wildcard": {
                    "text": {
                        "value": wildcard_query.lower(),
                        "case_insensitive": True
                    }
                }
            }
        }
        return self.execute_query(query, user)
```

Listing 7: 通配查询

4. 查询日志

- **查询日志功能：** 在每次用户执行查询时，记录查询时间、用户 ID、查询的短语或通配符、相关文档（如标题、URL、摘要）等信息，保存为日志文件（history.txt）。
- **日志查看：** 返回用户的搜索历史。

```
def log_query(self, user, query, results, history_file='history.txt'):
        timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        with open(history_file, 'a', encoding='utf-8') as f:
            f.write(f"{timestamp} | {user} | {query}\n")
            for res in results:
                snippet = res['text'].replace('\n', ' ').replace('\r', ' ')[:100]
                f.write(f"{timestamp} | {user} | {res['title']} | {res['url']} | {
    snippet}\n")
```

Listing 8: 查询日志

(三) 完成结果

短语查询，通配查询，查询日志结果如下：

```
===== 搜索引擎 =====
选择查询类型:
1. 短语查询
2. 通配查询
3. 联想关联（搜索建议）
4. 查看查询历史
5. 退出登录
输入数字选择操作: 1
请输入短语查询: 南开大学
F:\Users\lenovo\source\nk_search\search.py:147: DeprecationWarning: Received 'size' via a specific parameter in the presence of a 'body' parameter, which is deprecated and will be removed in a future version. Instead, use only 'body' or only specific parameters.
  response = self.es.search(index=self.index, body=function_score_query, size=4)

===== 搜索结果 =====

Rank 1:
Title: 南开大学
URL: https://www.nankai.edu.cn/
Final Score: 1.000000
Snippet: 信息公开 图书馆 服务指南 登录邮箱 办公网 校友入校 | English 学校概况 学校简介 现任领导 历届领导 历史回眸 南开新闻 南开新闻网 南开大学报 院系机构 专业学院 职能部门 直属和后勤单位 研究机构 人才师资 人事人才 人才招聘 博士后 教育教学 本科教育 研究生教育 本科教学质量报告 远程教育 继续教育 科学研究 自然科学 社会科学 自然科学研究机构 社会科学研究机构 仪器设备 ...

Rank 2:
Title: 规章制度
URL: https://xcb.nankai.edu.cn/gzzd/list.htm
Final Score: 0.189863
Snippet: 导航 首 页 机构设置 部门简介 现任领导 规章制度 常用下载 联系我们 规章制度 当前位置： 首页 规章制度 规章制度 南开大学 2023 年宣传思想工作要点 2023-04-13 中共南开大学委员会关于学习宣传贯彻党的二十大精神的安排意见 2023-04-13 南开大学 2022 年宣传思想工作要点 2023-04-13 关于深入学习宣传贯彻第十次党代会精神的安排意见 2023-04-13 南...

Rank 3:
Title: 常用下载
URL: https://xcb.nankai.edu.cn/11347/list.htm
Final Score: 0.122589
Snippet: 导航 首 页 机构设置 部门简介 现任领导 规章制度 常用下载 联系我们 常用下载 当前位置： 首页 常用下载 常用下载 南开大学教师到校外举办讲座、报告会申请 2020-05-19 南开大学报告会、研讨会、讲座、论坛和读书会、学术沙龙申请 2020-05-19 南开大学网上直播活动备案表 2020-05-19 每页 14 记录 总共 3 记录 第一页 <<上一页 下一页>> 尾页 页码 1 / ...

Rank 4:
Title: 宣传部
URL: https://xcb.nankai.edu.cn/main.htm
Final Score: 0.100000
Snippet: 导航 首 页 机构设置 部门简介 现任领导 规章制度 常用下载 联系我们 重要论述 更多>> 习近平寄语南开师生：只有把小我融入... 17日上午，习近平在天津南开大学参观了百年校史主题展览，察看了化学学院和元素有机化... 习近平总书记视察南开大学 2019-01-17 习近平：开辟马克思主义中国化时代化新境界 2023-10-17 习近平对宣传思想文化工作作出重要指示 2023-10-10 习...
======================
```

图 8: 短语查询 1

```
===== 搜索引擎 =====
选择查询类型:
1. 短语查询
2. 通配查询
3. 联想关联（搜索建议）
4. 查看查询历史
5. 退出登录
输入数字选择操作: 1
请输入短语查询: 大学 南开
F:\Users\lenovo\source\nk_search\search.py:147: DeprecationWarning: Received 'size' via a specific parameter in the presence of a 'body' parameter, which is deprecated and will be removed in a future version. Instead, use only 'body' or only specific parameters.
  response = self.es.search(index=self.index, body=function_score_query, size=4)

===== 搜索结果 =====

Rank 1:
Title: 南开大学校史网
URL: https://xs.nankai.edu.cn/
Final Score: 1.000000
Snippet: 首页 走进南开 活动要闻 纪念专题 校史研究 南开人物 校史文萃 更多 » 活动要闻 / News 教室搬上257米高空！天塔迎来首堂南开校史课 南开校史文化沙龙分享老校友口述史 “百年南开校史文化图书专架”揭幕 张元龙作专场讲座 叶嘉莹先生追思会举行 “叶先生，一路走好！” 南开大学师生追念叶嘉莹先生 国际著名教育家、诗人、中国古典文学研究泰斗、南开大学讲席教授叶嘉莹先生逝世 “西南联大时期民族...

Rank 2:
Title: 欢迎光临陈省身数学研究所数学图书馆！
URL: http://www.mathlib.nankai.edu.cn/
Final Score: 0.220993
Snippet: English Version 本馆概况 入馆指南 读者守则 赔偿规则 证籍管理 读者服务 ProQuest公司赠送American Mathematical Society出版社电子书53册 德古意特出版社和普林斯顿出版社数学专著电子图书开通通知 全球科学出版社(Global Science Press)数学电子期刊库开通通知 关于规范电子资源数据库使用的通知 馆藏目录检索 到馆新书目录 中文订...

Rank 3:
Title: 欢迎关注公众号
URL: https://law.nankai.edu.cn/2023/0910/c33629a519482/page.htm
Final Score: 0.100000
Snippet: 返回首页 会议室预定 服务指南 下载专区 English 旧版入口 学院概况 学院简介 学院党委 行政领导 历任领导 专业委员会 教代会、工会 管理团队 党团建设 风采展示 工作流程 常用下载 师资队伍 在职教师 荣休教师 永远的怀念 人才培养 教学培养 毕业学位 奖励资助 常用下载 科研动态 科研机构 科研成果 科研项目 学术活动 合作交流 国内合作 国际交流 招生工作 本科生 学术硕士 专业硕...
```

图 9: 短语查询 2

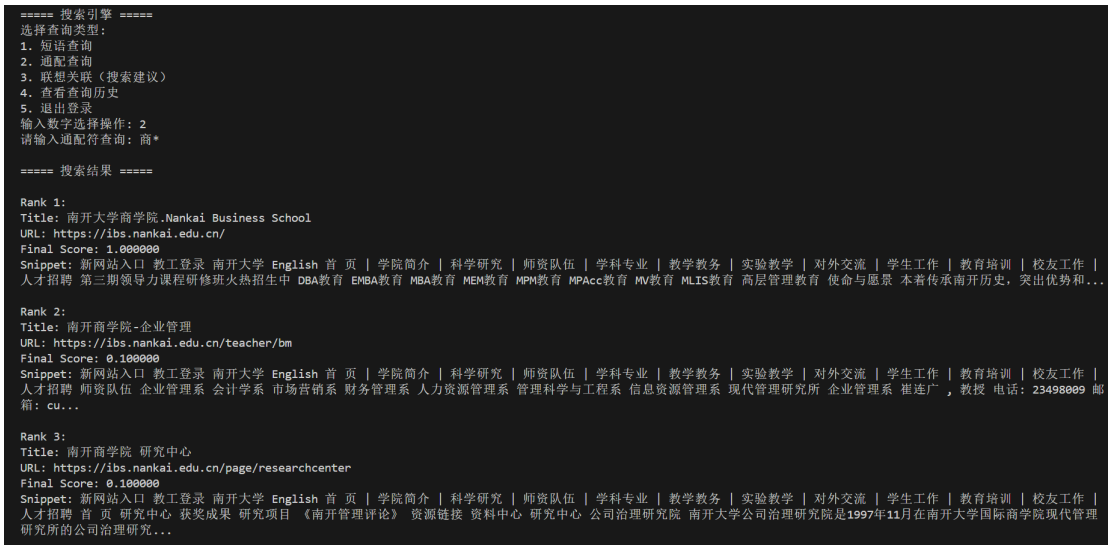

图 10: 通配查询

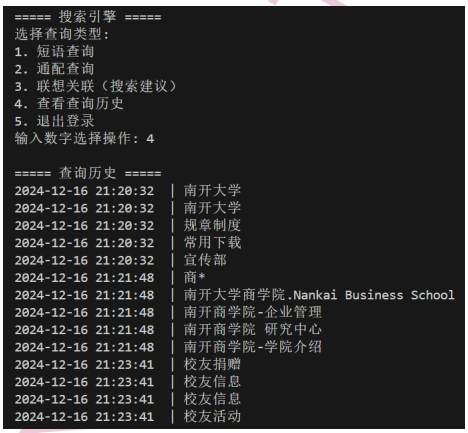

图 11: 查询日志

六、 个性化查询

(一) 设计思路

- **用户系统：** 设计用户注册登录功能来存储不同用户的信息以及查询记录。
- **用户历史记录的利用：** 查询的历史记录包含了用户的兴趣偏好。通过从历史记录中提取关键词（history_terms），在查询过程中，给予包含这些关键词的文档更高的权重。
- **动态加载用户历史：** 为了确保个性化查询的实时性，系统需要能够根据当前用户的身份，动态加载其历史查询记录。
- **搜索结果的重新排序：** 使用查询偏好得分和归一化加权操作，结合常规的搜索得分（如 TF-IDF、Pagerank 等）对结果进行重新排序。对于包含用户历史关键词的文档，可以通过加大其得分来使其优先显示，从而增强个性化推荐的效果。

(二) 技术路线

设计好了用户的注册登录系统来存储不同的用户信息和查询日志。

最初尝试从查询历史中的文档提取关键词和查询词，计算查询偏好得分，但是失败了。然后想要为每个文档定一个属性，比如科技、美食、文化等，根据用户查询的历史中哪一类别的文档最多，来推测用户的查询偏好，但是效果并不理想。所以个性化查询并没有得到完全的实现。

```
history_terms = self.load_user_history(user)     # 加载用户历史记录
if history_terms:
    # 对包含历史关键词的文档加权
    function_score_query["query"]["function_score"]["functions"].append({
        "filter": {
            "terms": {
                "text": history_terms
            }
        },
        "weight": 1.5  # 提高包含用户历史关键词的文档的权重
    })
```

Listing 9: 个性化查询

七、 Web 界面

为了展示功能和方便查询以及个性化操作，设计了初始页面、登录页面、查询页面、搜索结果页面、退出页面等简单可视化页面。未展示过的页面如下所示：

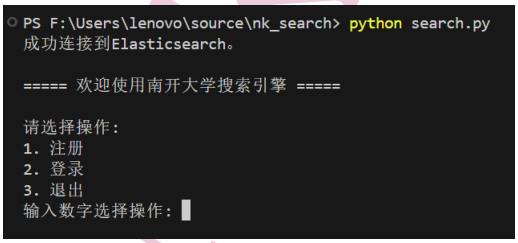

图 12: 初始页面

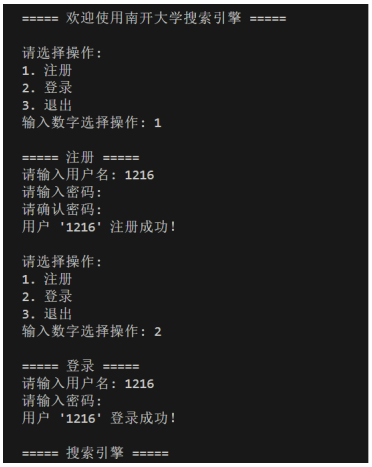

图 13: 注册登录页面

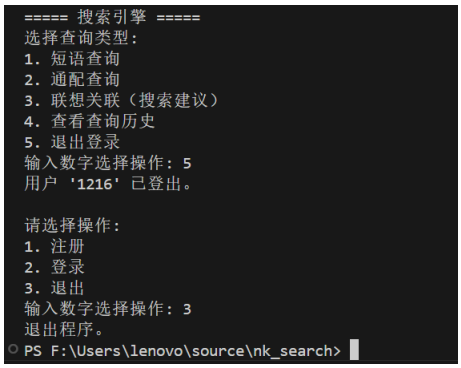

图 14: 退出页面

八、 个性化推荐

(一) 设计思路

- **联想建议的生成：**通过标题中的前几个词来构造一个简单的联想短语。通过分割标题字符串并取前两个词作为建议。如果标题仅包含一个词，则该词会作为建议的一部分。同时为了避免建议过长，使用省略号（...）来截断建议短语，确保联想建议简洁明了。
- **suggest 属性：**添加一个新的属性 suggest，将得到的联想建议存入映射中。
- **Elastic Search 联想功能：**使用 Elastic Search 的 completion suggester 提供基于前缀的模糊匹配，返回与用户输入前缀相关的关键词或短语建议。

(二) 技术路线

使用 Elastic Search 提供的 completion suggester API，根据前缀（如标题的部分文字）获取联想建议。然后根据已索引的文档和建议的字段进行模糊匹配，返回最相关的建议项。最后通过 wildcard_suggest 方法查询并返回相关的建议。

```
def wildcard_suggest(self, prefix):
        # 使用 Elasticsearch 的 completion suggester
        suggest = {
            "suggest": {
                "simple_phrase": {
                    "prefix": prefix,
                    "completion": {
                        "field": "suggest",
                        "fuzzy": {
                            "fuzziness": "AUTO"
                        },
                        "size": 5
                    }
                }
            }
        }
        try:
            response = self.es.search(index=self.index, body=suggest)
            suggestions = response.get('suggest', {}).get('simple_phrase', [])
            if suggestions:
```

```
                options = suggestions[0].get('options', [])
                return [sugg['text'] for sugg in options]
            return []
        except Exception as e:
            print(f"获取建议时发生错误: {e}")
            return []
```

Listing 10: 联想关联（搜索建议）

(三) 完成结果

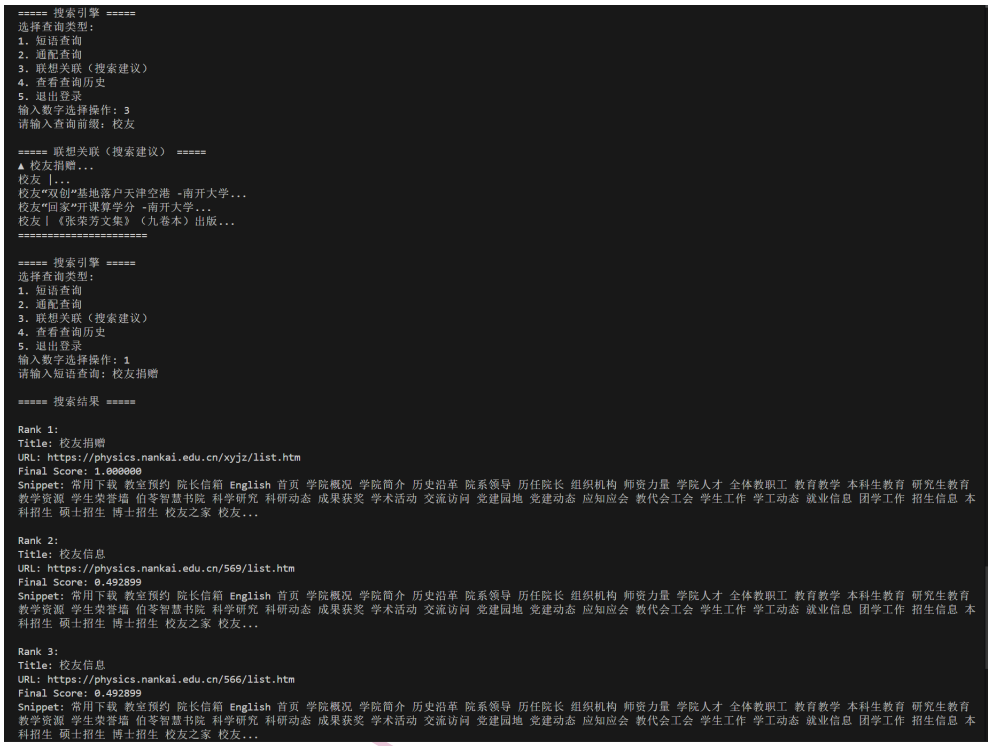

图 15: 联想关联（搜索建议）

九、 总结与感悟

历时 8 天，终于完成了本次系统性的实验。每一步都充满挑战，尤其是爬虫爬取数据阶段，在刚开始时存在很多疑虑，10 万条数据怎么爬？要爬多久？爬虫怎么这么慢？爬下来的数据后面能用吗？前前后后爬了好几次，每次一爬就是 8、9 个小时，甚至第一版爬虫爬了 24 小时，还不敢干别的，生怕爬虫会断，真的是“万事开头难”。这次试验遇到了很多问题，也做过很多尝试，不断地试错、改错以及调整方向。

当然，通过应用信息检索系统原理课程所学知识，设计和实现这个 Web 搜索引擎，收获颇丰。我深刻理解了搜索引擎的工作原理及其背后的技术细节，尤其是索引构建和链接分析。这次实验还增强了我对海量数据的处理能力，也掌握了强大工具 Elastic Search 的使用方法，锻炼了我实际项目开发与问题解决的能力，可能也会让我以后在查询我想要的信息时能够更加精准。