

Web 搜索引擎设计

原敬闰 2211771

December 18, 2024

1 简介

本项目从零实现了一个简单的 Web 搜索引擎，该搜索引擎可以进行普通查询，通配符查询，文件查询，短语查询，网页快照等功能。并提供了查询日志记录，个性化推荐的功能。

项目文件结构如下：

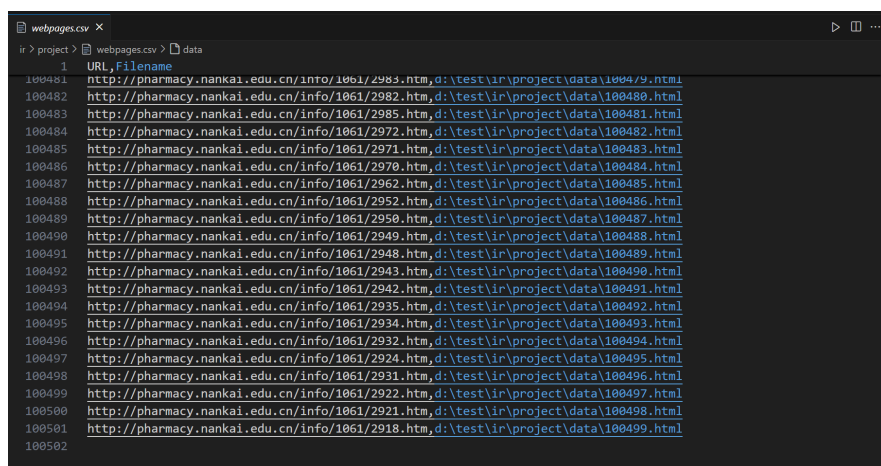
```
crawler.py          ---爬虫程序
calculate.py        ---计算pagerank
fileindex.py        ---生成文件索引
index.py            ---生成网站索引
photo.py            ---从csv文件中搜寻文件下载链接并索引
search.py           ---搜索模块
show_photos.py      ---显示网页快照
test.py             ---测试模块
flask_search/       ----web网页实现
    app.py           ---应用程序
    flask_session/   ----存储会话数据文件
    instance/        ----储存用户信息的数据库
        users.db     ---用户信息数据库
    templates/       ----储存网页html文件
        login.html   ---登录界面
        register.html ---注册界面
        search.html  ---查询界面
__pycache__/_       ----存储已编译字节码文件
```

2 搜索引擎的实现思路及细节

2.1 网页抓取

2.1.1 基本思路

该部分将 url 之间的连接关系当作一张有向图进行处理，从选取的节点进行 BFS 遍历所有节点，直到爬取的数量达到要求。每次爬取一个 url 就将其内容保存为 html 文件，并将 url 和文件保存的路径写入到 csv 文件中，csv 文件中的内容如图 1 所示。



```
webpages.csv
1  URL,Filename
100481 http://pharmacy.nankai.edu.cn/info/1061/2983.htm,d:\test\ir\project\data\100481.html
100482 http://pharmacy.nankai.edu.cn/info/1061/2982.htm,d:\test\ir\project\data\100482.html
100483 http://pharmacy.nankai.edu.cn/info/1061/2985.htm,d:\test\ir\project\data\100483.html
100484 http://pharmacy.nankai.edu.cn/info/1061/2972.htm,d:\test\ir\project\data\100484.html
100485 http://pharmacy.nankai.edu.cn/info/1061/2971.htm,d:\test\ir\project\data\100485.html
100486 http://pharmacy.nankai.edu.cn/info/1061/2970.htm,d:\test\ir\project\data\100486.html
100487 http://pharmacy.nankai.edu.cn/info/1061/2962.htm,d:\test\ir\project\data\100487.html
100488 http://pharmacy.nankai.edu.cn/info/1061/2952.htm,d:\test\ir\project\data\100488.html
100489 http://pharmacy.nankai.edu.cn/info/1061/2950.htm,d:\test\ir\project\data\100489.html
100490 http://pharmacy.nankai.edu.cn/info/1061/2949.htm,d:\test\ir\project\data\100490.html
100491 http://pharmacy.nankai.edu.cn/info/1061/2948.htm,d:\test\ir\project\data\100491.html
100492 http://pharmacy.nankai.edu.cn/info/1061/2943.htm,d:\test\ir\project\data\100492.html
100493 http://pharmacy.nankai.edu.cn/info/1061/2942.htm,d:\test\ir\project\data\100493.html
100494 http://pharmacy.nankai.edu.cn/info/1061/2935.htm,d:\test\ir\project\data\100494.html
100495 http://pharmacy.nankai.edu.cn/info/1061/2934.htm,d:\test\ir\project\data\100495.html
100496 http://pharmacy.nankai.edu.cn/info/1061/2932.htm,d:\test\ir\project\data\100496.html
100497 http://pharmacy.nankai.edu.cn/info/1061/2924.htm,d:\test\ir\project\data\100497.html
100498 http://pharmacy.nankai.edu.cn/info/1061/2931.htm,d:\test\ir\project\data\100498.html
100499 http://pharmacy.nankai.edu.cn/info/1061/2922.htm,d:\test\ir\project\data\100499.html
100500 http://pharmacy.nankai.edu.cn/info/1061/2921.htm,d:\test\ir\project\data\100500.html
100501 http://pharmacy.nankai.edu.cn/info/1061/2918.htm,d:\test\ir\project\data\100501.html
100502
```

Figure 1: csv 文件展示

2.1.2 关键代码讲解

爬取函数: 初始时待访问队列中只有一个起始 url，起始 url 是南开大学官网，每次循环程序都会从队列中取出第一个 url，下载这个 url 响应的 html 文件，然后对这个 url 的中包含的跳转链接进行解析加入到待解析队列中。

```
def crawl(self):
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
    }
    while self.to_visit_urls and len(self.visited_urls) < self.max_pages:
        current_url = self.to_visit_urls.pop(0)
        if current_url in self.visited_urls:
            continue
        try:
            response = requests.get(current_url, headers=headers, timeout=(2, 5))
            if response.status_code == 200:
                response.encoding = response.apparent_encoding
                if response.encoding == None:
                    continue
                try:
                    html_content = response.content.decode(
                        response.encoding, errors="replace"
                    )
                except (UnicodeDecodeError, AttributeError):
                    pass
                self.visited_urls.add(current_url)
                print(f"Crawled: {current_url} (Total:{len(self.visited_urls)})")
                self.save_page(html_content, current_url)
                self.extract_links(html_content, current_url)
            else:
                print(f"Failed to retrieve: {current_url}")
        except requests.exceptions.Timeout:
            pass
        except requests.RequestException:
            pass
        except Exception:
            pass
    print(f"Total pages crawled: {len(self.visited_urls)}")
```

Listing 1: 爬取函数

链接提取：到达一个 url 之后，使用解析器提取页面中的所有链接，然后尝试将每个链接和当前 url 进行拼接，得到一个完整的 url，确定这个 url 中有 `nankai.edu.cn`，并确定 url 并不在已访问队列和待访问队列中，随后就将新 url 假如到待访问列表中。

```

def extract_links(self, html, base_url):
    try:
        soup = BeautifulSoup(html, "lxml")
        for link in soup.find_all("a", href=True):
            href = link["href"]
            if any(char in href for char in [" ", ":", "#", "javascript:"]):
                continue
            try:
                url = urljoin(base_url, href)
            except ValueError:
                pass
            if (
                "nankai.edu.cn" in url
                and url not in self.visited_urls
                and url not in self.to_visit_urls
            ):
                self.to_visit_urls.append(url)
    except Exception:
        pass

```

Listing 2: 链接提取

保存页面：获取到一个网页内容和一个 url 的时候，就根据当前的爬取数量作为文件名，与设置好的存储路径进行拼接，将 html 文件保存到这个文件路径下，然后将文件路径和 url 一起写入到 csv 文件中。

```

def save_page(self, html, url):
    try:
        filename = os.path.join(
            self.save_dir,
            str(self.filenum) + ".html",
        )
        self.filenum += 1
        with open(filename, "w", encoding="utf-8") as file:
            file.write(html)
        print(f"Saved: {url}")
        self.write_to_csv(url, filename)
    except OSError:
        pass
    except Exception:
        pass

```

Listing 3: 保存页面

csv 文件写入：获取到一个 url 和文件路径后就将其写入到准备好的 csv 文件中

```
def write_to_csv(self, url, filename):
    try:
        with open(self.csv_file, mode="a", newline="", encoding="utf-8") as file:
            writer = csv.writer(file)
            print(f"Writing to CSV: {url}, {filename}")
            writer.writerow([url, filename])
    except OSError:
        pass
    except Exception:
        pass
```

Listing 4: csv 文件写入

2.2 索引生成

2.2.1 基本思路

使用 elasticsearch 工具建立索引，从爬取阶段写入的 csv 文件中读取一行，调用工具解析文件路径下的 html 文件，从中提取出 title, content, anchors, 然后将这三项和 url 写入到索引中。

2.2.2 关键代码实现

对于每个 url，索引中都储存四个字段，url 字段作为 keyword 索引时不进行分词操作，title 和 content 作为字符串，使用 ik_smart 分析器解析后建立倒排索引，anchors 是一个复杂字段，包含一个锚文本到 url 映射的字典。

```
index_settings = {
    "settings": {
        "number_of_shards": 1,
        "number_of_replicas": 0,
        "analysis": {"analyzer": {"default": {"type": "ik_smart"}}}},
    },
    "mappings": {
        "properties": {
            "url": {"type": "keyword"},
            "title": {"type": "text", "analyzer": "ik_smart"},
            "content": {"type": "text", "analyzer": "ik_smart"},
            "anchors": {
                "type": "nested",
                "properties": {
                    "anchor_text": {"type": "text", "analyzer": "ik_smart"},
                    "target_url": {"type": "keyword"},
                },
            },
        },
    },
}
```

Listing 5: 索引结构

html 内容解析：首先根据文件内容使用 `chardet` 库来尝试判断其中的编码方式，然后使用检测到的编码方式打开文件，使用 `lxml` 分析 `html` 文件，提取 `title`，`content`，`anchor` 字段，对字符串中的空白、换行符等进行处理，对于提取到的链接，分析这个链接是否是相对链接，如果是相对的就需要和当前的 `url` 进行拼接得到这个锚文本的 `targeturl`。

```

def extract_data_from_html(url, html_path):
    with open(html_path, "rb") as file:
        raw_data = file.read()
        result = chardet.detect(raw_data)
        encoding = result["encoding"]

    with open(
        html_path.replace("\\", "/"), "r", encoding=encoding, errors="ignore"
    ) as file:
        soup = BeautifulSoup(file, "lxml")
        title = (
            soup.title.string.strip().replace("\n", "").replace(" ", "")
            if soup.title and soup.title.string
            else ""
        )
        content = ", ".join(
            [
                line.strip().replace("\n", "").replace(" ", "")
                for line in soup.get_text().splitlines()
                if line.strip()
            ]
        )
        anchors = []
        for a in soup.find_all("a"):
            anchor_text = a.get_text().strip().replace("\n", "").replace(" ", "")
            try:
                href = a.get("href", "")
                if not urlparse(href).netloc:
                    target_url = urljoin(url, quote(href))
                else:
                    target_url = href
                anchors.append({"anchor_text": anchor_text, "target_url": target_url})
            except ValueError as e:
                print(f"Skipping invalid URL {href}: {e}")
        return title, content, anchors

```

Listing 6: html 内容解析

索引写入：遍历 csv 文件的每一行，获取一组文件路径和对应的 url，检查文件的大小，如果文件过大就跳过，调用 `extract_data_from_html` 函数来解析该文件。然后创建一个 action 来将这些内容储存到索引缓冲区中，处理完每一行后，调用 `es` 来进行索引内容的更新。

```
with open(csv_file_path, "r", encoding="utf-8") as csvfile:
    reader = csv.DictReader(csvfile)
    i = 0
    for row in reader:
        url = row["URL"]
        html_path = row["Filename"]

        # Check file size
        if os.path.getsize(html_path) > max_file_size:
            print(f"Skipping {html_path} due to large file size.")
            continue

        title, content, anchors = extract_data_from_html(url, html_path)

        action = {
            "_index": index_name,
            "_source": {
                "url": url,
                "title": title,
                "content": content,
                "anchors": anchors,
            },
        }
        actions.append(action)
    print("Indexing...")
    helpers.bulk(es, actions, chunk_size=100, request_timeout=120)
    print("Done!")
```

Listing 7: 索引写入

2.3 链接分析

2.3.1 基本思路

该部分构造一个有向图，读取 web_pages 中的文档并根据文件的指向关系为有向图添加边，最后根据有向图来计算每个节点（代表一个 url）的 PageRank。并重新创建一个索引 pagerank，存储每个 url 和 PageRank 的对应关系。

2.3.2 关键代码分析

创建索引： pagerank 索引中存储一个 url 到 pagerank 值的映射。


```

index_name = "web_pages"
pagerank_index_name = "pagerank"
# Create the new index for PageRank values
index_settings = {
    "settings": {"number_of_shards": 1, "number_of_replicas": 0},
    "mappings": {
        "properties": {"url": {"type": "keyword"}, "pagerank": {"type": "float"}}
    },
}

```

Listing 8: 索引结构

获取文档：通过滚动查询获取在上一步建立索引时存储的所有文档，并创建一个空有向图

```

query = {"query": {"match_all": {}}}

# 从 Elasticsearch 获取所有文档
response = es.search(
    index=index_name, body=query, size=10000, scroll="5m", request_timeout=120
)
G = nx.DiGraph()
# Scroll through all documents
scroll_id = response["_scroll_id"]

```

Listing 9: 获取 es 中所有的文档

计算 PageRank：根据 response 中返回的结果，从锚文本中读取当前 url 可以链接到的 target_urls。然后在有向图中添加一条从当前 url 到 target_url 的边。在图构建完成后，根据这张图计算每个节点的 PageRank。

```

while True:
    print("Processing scroll batch...")
    for hit in response["hits"]["hits"]:
        source_url = hit["_source"]["url"]
        # Extract anchor texts and target URLs from the document
        anchors = hit["_source"].get("anchors", [])
        for anchor in anchors:
            target_url = anchor.get("target_url")
            if target_url:
                G.add_edge(source_url, target_url)
                # print(f"Added edge from {source_url} to {target_url}")

    if "hits" not in response or not response["hits"]["hits"]:
        break

    scroll_id = response["_scroll_id"]
    try:
        response = es.scroll(scroll_id=scroll_id, scroll="5m", request_timeout=120)
    except Elasticsearch.NotFoundError:
        print("Scroll ID not found or expired. Exiting scroll loop.")
        break

# 计算 PageRank
pagerank = nx.pagerank(G)

```

Listing 10: 计算 PageRank

写入索引：根据计算出的 PageRank，为每个 url 构建 action，最后更新索引

```

actions = []
for url, rank in pagerank.items():
    action = {"_index": pagerank_index_name, "_source": {"url": url, "pagerank": rank}}
    actions.append(action)
    print(f"Prepared action for URL: {url}, PageRank: {rank}")
print("action", len(actions))
success, failed = helpers.bulk(es, actions, chunk_size=100, stats_only=True)
print(f"Bulk indexing completed. Success: {success}, Failed: {failed}")

```

Listing 11: 写入索引

2.4 查询服务

查询这部分使用的是 `es.search` 的方法，以下会分各个功能来进行关键代码的讲解

2.4.1 站内查询

站内查询：首先需要定义一个接口，这个函数接收一个查询字符串和用户学院信息字符串，接收到之后，首先查看 query 是否开头是 http，来判断是否查询一个 url，随后按照空格将字符串分开进行处理，如果其中有通配符就调用通配查询函数，否则就使用短语查询，短语查询结果如果不足 1000 个，就补充进行多词查询。

```
def search_and_rank(query, college=None):
    """ 处理查询并对结果进行排序的主搜索函数。 """
    print(f"Original query: {query}")
    if is_url(query):
        url_response = search_url(query)
        if url_response["hits"]["hits"]:
            title = url_response["hits"]["hits"][0]["_source"]["title"]
            return [(query, title)]
        else:
            return None
    query_parts = query.split(" ")
    results_list = []
    for part in query_parts:
        print(f" 处理查询部分: {part}")
        if "*" in part or "?" in part:
            print("  进入到了通配符查询部分")
            wildcard_response = search_wildcard(part, college)
            results_list.append(wildcard_response)
        else:
            # 标准分词查询
            phrase_response = search_phrase(part, college)
            if phrase_response["hits"]["hits"]:
                print("  进入到了短语查询部分")
                results_list.append(phrase_response)
                if len(phrase_response["hits"]["hits"]) > 100:
                    break
            print("  进入到了多词查询部分")
            match_response = search_match(part, college)
            results_list.append(match_response)
```

Listing 12: 将查询字符串进行分部分处理

返回查询结果：将之前查询的结果进行合并，然后将 es 查询的分数和 PageRank 的分数进行权重分配，由于二者大小差距太大，所以权重需要调整为合适的值来平衡二者的影响，最后根据综合得分排序并返回结果。

```
merged_results = merge_results(results_list)
urls = [hit["_source"]["url"] for hit in merged_results]
titles = [hit["_source"]["title"] for hit in merged_results]
scores = [hit["_score"] for hit in merged_results]
pagerank_values = get_pagerank(urls)
combined_scores = []
for i, url in enumerate(urls):
    combined_score = (
        scores[i] * 0.001 + pagerank_values[url] * 1000
    ) # Pagerank 权重为 1.0
    combined_scores.append((url, titles[i], combined_score))
sorted_scores = sorted(combined_scores, key=lambda x: x[2], reverse=True)
return [(url, title) for url, title, score in sorted_scores]
```

Listing 13: 返回查询结果

2.4.2 短语查询

使用 bool 查询，将 query 放在 must 中，将 college 放在 should 中，将二者的权重设置为 3: 2，控制 es 返回得分存在偏好，并将 title, content, anchors 的权重设置为 7: 3: 2。

```
def search_phrase(query, college):  
    """ 使用 'match_phrase' 查询, 并将 college 添加为加权因子 """  
    response = es.search(  
        index=index_name,  
        body={  
            "query": {  
                "bool": {  
                    "must": [  
                        {  
                            "multi_match": {  
                                "query": query,  
                                "fields": [  
                                    "title^7",  
                                    "content^3",  
                                    "anchors.anchor_text^2",  
                                ],  
                                "boost": 3.0,  
                            }  
                        }  
                    ],  
                    "should": [  
                        {  
                            "multi_match": {  
                                "query": college,  
                                "fields": [  
                                    "title^7",  
                                    "content^3",  
                                    "anchors.anchor_text^2",  
                                ],  
                                "boost": 2,  
                            }  
                        }  
                    ],  
                    "minimum_should_match": 0,  
                }  
            },  
            "size": 1000,  
        },  
    )  
    return response
```

Listing 14: 计算 PageRank

2.4.3 通配查询

通配查询：使用 es 中的 wildcard 查询实现。

```
def search_wildcard(query_text, college):  
    """ 使用 'wildcard' 查询进行通配符匹配, 并将 college 添加为加权因子 """  
    response = es.search(  
        index=index_name,  
        body={  
            "query": {  
                "bool": {  
                    "must": [  
                        {  
                            "wildcard": {  
                                "title": {  
                                    "value": query_text,  
                                    "boost": 3.0,  
                                }  
                            }  
                        ],  
                    },  
                    "should": [  
                        {  
                            "multi_match": {  
                                "query": college,  
                                "fields": [  
                                    "title^7",  
                                    "content^3",  
                                    "anchors.anchor_text^2",  
                                ],  
                                "boost": 2,  
                            }  
                        }  
                    ],  
                    "minimum_should_match": 0,  
                },  
            },  
            "size": 1000,  
        },  
    )  
    return response
```

Listing 15: 计算 PageRank

2.4.4 文档查询

文档查询的基本思路是，通过对 webpages.csv 中的文档下载链接进行过滤，使用代码提取出文件中的文本，然后再构建一个索引，在进行文档查询的时候，从这个文档中进行搜索并返回文档下载链接。

建立文档索引：文档索引的结构就是文件的下载链接和文件的内容。

```
INDEX_MAPPING = {
    "mappings": {
        "properties": {
            "url": {"type": "keyword"},
            "content": {"type": "text", "analyzer": "ik_smart"},
        }
    }
}

def check_create_index(index_name):
    if es.indices.exists(index=index_name):
        es.indices.delete(index=index_name)
    es.indices.create(index=index_name, body=INDEX_MAPPING)
check_create_index(INDEX_NAME)
```

Listing 16: 建立文档索引

筛选并处理文件下载链接：从 csv 文件中读取每一行，随后判断 url 的末尾是否是文件后缀，如果是就下载文件并解析其中的内容，将其添加到索引中

```
def process_csv(csv_path):
    with open(csv_path, newline="", encoding="utf-8") as csvfile:
        csvreader = csv.DictReader(csvfile)
        i = 0
        for row in csvreader:
            url = row["URL"]
            if is_supported_file(url):
                print(f"Processing: {url}")
                file_path, filename = download_file(url, DOWNLOAD_FOLDER)
                if file_path:
                    file_extension = filename.split(".")[1].lower()
                    content = extract_text(file_path, file_extension).replace(" ", "")
                    index_document(url, content)
```

Listing 17: 筛选并处理文件下载链接

提取文档中的文本内容：此处通过调包实现，具体细节都在源码中，此处不再赘述。

```
def extract_text(file_path, file_extension):  
    """  
    根据文件扩展名提取文件内容  
    """  
    text = ""  
    try:  
        if file_extension == "pdf":  
            text = extract_text_from_pdf(file_path)  
        elif file_extension == "doc":  
            text = extract_text_from_doc(file_path)  
        elif file_extension in ["ppt", "pptx"]:  
            text = extract_text_from_ppt(file_path)  
        elif file_extension in ["xls", "xlsx"]:  
            text = extract_text_from_excel(file_path)  
        print(f"Extracted text {text} from {file_path}")  
    except Exception as e:  
        print(f"Error extracting text from {file_path}: {e}")  
    return text
```

Listing 18: 提取文档中的文本

将 url 和文本内容写入到文件索引中。

```
def index_document(url, content):  
    """  
    将文件的 URL 和内容索引到 Elasticsearch 中  
    """  
    doc = {"url": url, "content": content}  
    try:  
        es.index(index=INDEX_NAME, document=doc)  
        print(f"Document indexed: {url}")  
    except Exception as e:  
        print(f"Error indexing document {url}: {e}")
```

Listing 19: 写入索引

文件查询：从文件索引中查询文本，采用 match 查询，返回文件的下载链接。


```
def search_file(query):
    response = es.search(
        index="fileindex", body={"query": {"match": {"content": query}}, "size": 100}
    )
    results = []
    for hit in response["hits"]["hits"]:
        file_url = hit["_source"]["url"]
        results.append((file_url, " 下载链接"))
    return results if results else None
```

Listing 20: 文件查询

2.4.5 web 界面实现 & 查询日志

web 界面使用的是 flask 前后端分离框架，使用数据库储存用户信息，实现简单的界面。

在查询之前，会检查该查询记录是否已经存在，如果不存在，就将其添加到查询记录中。在处理查询时，前端页面会获取到当前查询的类型 (file 或 None) 以及用户的 college 字段，然后调用不同的函数进行处理。

这一段实现搜索记录的时候，调用的控件会自行根据历史记录中存储的查询日志以及当前输入的查询来为用户推荐补全查询。

```

@app.route("/search", methods=["GET", "POST"])
def search_page():
    if "username" not in session:
        return redirect(url_for("login")) # 如果未登录, 重定向到登录页面

    user_id = session["user_id"]
    college = session["college"]
    results = None
    if request.method == "POST":
        query = request.form["query"]
        search_type = request.form["search_type"] # 获取查询类型

        # 检查是否已经存在相同的搜索记录
        existing_record = SearchHistory.query.filter_by(
            user_id=user_id, search_query=query
        ).first()
        if not existing_record:
            # 保存搜索记录
            new_record = SearchHistory(user_id=user_id, search_query=query)
            db.session.add(new_record)
            db.session.commit()

        if search_type == "file":
            results = search.all_search(query, college, "file") # 调用文件搜索
        else:
            results = search.all_search(query, college, None) # 调用普通搜索

        # 获取用户的搜索记录
        search_history = SearchHistory.query.filter_by(user_id=user_id).all()
        search_history = [record.search_query for record in search_history]

    return render_template(
        "search.html", results=results, search_history=search_history
    )

```

Listing 21: search 逻辑

2.4.6 网页快照

网页快照的实现思路是通过将网页, 包括所有资源文件都存储在本地来实现的, 在储存网页内容的时候, 将 url 和网页的储存路径存储在索引中, 在用户点击网页快照的时候, 查询 url 来获取到快照的存储路径, 调用浏览器根据存储路径打开本地储存的网页内容。

快照索引的结构：主要存储的就是 url 链接和对应的快照 html 的存储路径。

```
index_settings = {  
    "settings": {"number_of_shards": 1, "number_of_replicas": 0},  
    "mappings": {  
        "properties": {"url": {"type": "keyword"}, "filepath": {"type": "keyword"}}  
    },  
}
```

Listing 22: 建立快照索引

下载网页资源：从网站的响应中提取出 html, css, jpg 等资源，并修改 html 文件中的资源路径。

```
def download_webpage(url):
    # 请求网页 HTML
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
    }
    response = requests.get(url, headers=headers, timeout=(4, 8))
    response.encoding = response.apparent_encoding # 使用响应的推测编码
    soup = BeautifulSoup(response.text, "html.parser")
    # 创建一个目录来存储网页快照
    domain = urlparse(url).netloc
    snapshot_dir = os.path.join("d:\\test\\ir\\project\\photos", f"{domain}_snapshot")
    if not os.path.exists(snapshot_dir):
        os.makedirs(snapshot_dir)
    # 保存 HTML 内容
    html_file = os.path.join(snapshot_dir, "index.html")
    with open(html_file, "w", encoding="utf-8") as f:
        f.write(soup.prettify())
    # 下载页面中的所有资源 (图片、CSS、JS 等)
    resources = soup.find_all(["img", "link", "script"])
    for resource in resources:
        if resource.name == "img" and resource.get("src"):
            resource_url = resource["src"]
            if not resource_url.startswith(("http", "https")):
                resource_url = urljoin(url, resource_url)
            resource_path = download_resource(resource_url, snapshot_dir)
            resource["src"] = os.path.relpath(resource_path, start=snapshot_dir)
        elif resource.name == "link" and resource.get("href"):
            resource_url = resource["href"]
            if not resource_url.startswith(("http", "https")):
                resource_url = urljoin(url, resource_url)
            resource_path = download_resource(resource_url, snapshot_dir)
            resource["href"] = os.path.relpath(resource_path, start=snapshot_dir)
        elif resource.name == "script" and resource.get("src"):
            resource_url = resource["src"]
            if not resource_url.startswith(("http", "https")):
                resource_url = urljoin(url, resource_url)
            resource_path = download_resource(resource_url, snapshot_dir)
            resource["src"] = os.path.relpath(resource_path, start=snapshot_dir)
    # 保存更新后的 HTML 内容
    with open(html_file, "w", encoding="utf-8") as f:
        f.write(soup.prettify())
    return html_file
```

Listing 23: 下载网页资源

由于电脑内存有限，所以只存储了一部分网页快照。

```
with open(csv_file_path, "r", encoding="utf-8") as csvfile:
    reader = csv.DictReader(csvfile)
    i = 0
    for row in reader:
        url = row["URL"]
        try:
            filepath = download_webpage(url)
        except:
            print("An error!")
            continue
        action = {
            "_index": index_name,
            "_source": {
                "url": url,
                "filepath": filepath,
            },
        }
        actions.append(action)
        i += 1
        if i > 1000:
            break
helpers.bulk(es, actions, chunk_size=100, request_timeout=120)
```

Listing 24: 写入索引

搜索快照存储路径：使用精确查询来寻找快照路径，如果不存在就返回 None。

```
def searchphoto(url):
    response = es.search(
        index=index_name,
        body={
            "query": {"term": {"url": url}},
            "size": 100,
        },
    )
    hits = response["hits"]["hits"]
    if hits:
        return hits[0]["_source"]["filepath"].replace("\\", "/")
    return None
```

Listing 25: 搜索快照路径

显示网页快照：调用 showphoto 模块来搜索 url 对应的快照存储路径，然后调用 webbrowser 来打开这个位置的 html 文件。如果不存在快照就显示“没有对应的快照”。

```
@app.route("/home")
def home():
    url = request.args.get("url")
    # 调用 showphoto 模块生成网页快照
    snapshot_path = show_photos.searchphoto(url)
    # 返回快照文件地址
    if snapshot_path is None:
        flash(" 没有对应的快照")
        return redirect(url_for("search_page"))
    print(snapshot_path)
    file_url = "file:/// " + snapshot_path
    webbrowser.open(file_url)
    return jsonify({"success": True, "file_url": file_url})
```

Listing 26: 显示网页快照

2.4.7 个性化查询 & 个性化推荐

个性化通过数据库来储存用户信息和用户查询日志，并在用户输入数据的时候，寻找是否存在过去相似的查询记录，为用户推荐补全查询。

数据库：数据库中保存用户昵称，密码，学院。

```
# 数据库模型
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    nickname = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=False)
    college = db.Column(db.String(200), nullable=False)
# 根路由，默认重定向到登录页面
@app.route("/")
def index():
    return redirect(url_for("login")) # 如果未登录，跳转到登录页面
```

Listing 27: 数据库设置：

总搜索函数：根据用户查询类型调用不同的函数，搜索类型如果为 file 就搜索文件，否则就正常搜索。

```
def all_search(query, college, qtype):  
    if qtype == "file":  
        return search_file(query)  
    return search_and_rank(query, college)
```

Listing 28: 总搜索函数

个性化搜索：个性化搜索是根据用户的学院信息来调整检索结果的权重，从而增加那些用户可能更感兴趣的文档的权重。

个性化推荐：个性化推荐的思路是基于当前用户过去的查询以及所有用户过去的查询来对当前输入的查询进行推荐补全。

3 遇到的问题

本次大作业，我从零开始自行搭建一个简单的搜索引擎，对我而言是一个很大的挑战，是我第一次尝试一个简单的项目，遇到了很多问题。对我而言比较困难的地方有以下几个：

3.1 爬虫部分

由于对各种编码方式以及爬取细节不熟悉，我的爬虫总是会产生各种各样的错误，导致我无法爬取到足够数量的网页，在这个过程中我尝试了许多方案，例如使用函数根据二进制数据判断网页的编码方式，使用请求头来伪装等等。

3.2 search 的实现

在实现 search 的时候，由于对 es 自带的查询函数不熟悉，一开始我是将 search 函数的结果再进行余弦相似度的计算从而得到分数，这样的函数编写复杂且搜索时间较长，需要十多秒才能返回结果，在优化时我仔细学习了 es.seach 的查询格式，极大的简化了代码，加速了查询效率。

3.3 网页的搭建

由于我从未涉足过网站的搭建，所以一开始我极为忐忑，不知道该怎样将我已经编写好的后端查询和前端的界面进行拼接，也不知道该如何编写跳转逻辑，这部分也花了很多时间来调整各种报错，最终我也只是简单实现了一个界面，让其能够正常完成各种跳转功能。

4 总结与感想

本次作业的实现对我而言较为坎坷，在实现各个功能的过程中遇到了许多问题，解决问题的过程中，我也受益匪浅，对信息检索系统，以及一些简单的编程思想都有了更深入的体会。