信息检索大作业

2212410 刘俊彤

本项目基于Elastic Search框架搭建了一个面对南开大学校内资源的简易Web搜索引擎，实现了站内查询、短语查询、文档查询、通配查询四种查询方式，并且对登录的用户提供查询日志、个性化查询和个性化推荐功能。

项目文件结构：

```
NKDEX/
|
├── crawl.py                        # 爬虫
├── pagerank.py                     # 链接分析
├── buildIndex.py                   # 构建索引
|
├── Search/                         # 应用程序文件夹
|   ├── app.py                      # 后端代码
|   ├── templates/                  # 前端文件夹
|   |   ├── site_search.html        # 站内查询
|   |   ├── file_search.html        # 文档查询
|   |   ├── phrase_search.html      # 短语查询
|   |   ├── wildcard_search.html    # 通配查询
|   |   ├── login.html              # 登录页面
|   |   ├── register.html           # 注册页面
|   |   └── user_page.html          # 用户主页
|   |
|   |
|   └── static/                     # 静态资源文件夹
|       └── styles.css              # 自定义CSS文件
|
├── result.json                     # 网页信息
├── user.json                       # 用户信息
└── 网页文件 ...
```

一、网页抓取

完整代码见`crawl.py`

使用`BeautifulSoup`按照域名进行爬取网页，对于每个域名的网页信息按照`[{ title, url, content, referenced_urls, html_filename }]`存储在`json`文件中，并将`html`文件存储到域名对应的文件夹中。

```python
def crawl_website(start_url):
    """爬取整个网站"""
    visited = set()
    queue = [start_url]
```

```
    data = []  # 用于保存网页数据的列表
    while queue:
        current_url = queue.pop(0)
        if current_url not in visited:
            print(f"Crawling: {current_url}")
            visited.add(current_url)
            links = get_links(current_url)
            for link in links:
                if link not in visited and 'zfxy.nankai.edu.cn' in link and not
(link.endswith('.doc') or link.endswith('.pdf') or link.endswith('.docx') or
link.endswith('.rar') or link.endswith('.zip') or link.endswith('.xlsx') or
link.endswith('.xls') or link.endswith('.ppt') or link.endswith('.pptx') or
link.endswith('.jpg') or link.endswith('.png')):  # 筛选文档网页
                    queue.append(link)
            try:
                response = requests.get(current_url, timeout=10)
                response.encoding = chardet.detect(response.content)['encoding']
 # 检测并设置编码
                if response.status_code == 200:
                    soup = BeautifulSoup(response.text, 'html.parser')
                    title = soup.title.string if soup.title else 'No Title'
                    content = soup.get_text(strip=True, separator=' ')

                    html_filename = save_html_to_file(current_url,
response.text)
                    # 提取网页中引用的URL
                    referenced_urls = [a['href'] for a in soup.find_all('a',
href=True) if is_valid_url(a['href'])]
                    # 保存数据到列表
                    data.append({
                        'title': title,
                        'url': current_url,
                        'content': content,
                        'referenced_urls': referenced_urls,
                        'html_filename': os.path.basename(html_filename)
                    })
                    with open('nankai_zfxy.json', 'w', encoding='utf-8') as f:
                        json.dump(data, f, ensure_ascii=False, indent=4)
                    # 模拟请求之间的延迟
                    time.sleep(0.5)

            except requests.exceptions.Timeout:
                print(f"Request timed out for {current_url}")
            except Exception as e:
                print(f"Error processing {current_url}: {e}")
```

最初爬取遇到了两个问题。第一个是爬下来的很多网页内容是乱码的，发现是字符集问题，故添加了编码的检查和设置；第二个问题是由于爬取了学院网，有很多文档下载url，会导致下载了大量的文档，空间占用很大，所以对于带有文档后缀的url进行了过滤。

共爬取了南开大学新闻网、南开大学报以及13个学院官网，共`120729`个网页。

二、链接分析

完整代码见`pagerank.py`

具体步骤如下：

- 读取所有`json`文件获得所有网页信息`data`
- 将`data`根据`url`进行去重，保证没有重复网页
- 使用`networkx.DiGraph()`根据`referenced_url`字段构建图模型
- 使用`networkx.pagerank()`进行计算每个网页的`pr`值
- 网页数据添加`pr`字段
- 将所有网页数据写入`result.json`文件

```
# 构建图模型
def build_graph(data):
    graph = nx.DiGraph()
    for item in data:
        graph.add_node(item['url'], pr=0)
        for referenced_url in item['referenced_urls']:
            graph.add_edge(item['url'], referenced_url)
    return graph

# 计算PageRank
def calculate_pagerank(graph):
    pageranks = nx.pagerank(graph)
    for node, pr in pageranks.items():
        graph.nodes[node]['pr'] = pr
    return graph
```

完成链接分析后计算了一下网页总数以及pr值情况。输出如下：

```
PS F:\NKU\大三上\信息检索系统原理\lab4-大作业\大作业> & D:
Total number of data: 120729
Total number of unique_data: 120729
PS F:\NKU\大三上\信息检索系统原理\lab4-大作业\大作业> & D:
Total number of unique PageRank values: 546
```

共计`120729`个网页，有`546`个不同的pr值。

三、文本索引

完整代码见`buildIndex.py`

使用`elasticsearch`框架构建了名叫`webpages`的索引，使用`ik`分词器进行分词。

```
index_mapping = {
    "settings": {
        "analysis": {
            "analyzer": {
                "ik_analyzer": {
                    "type": "custom",
                    "tokenizer": "ik_smart"  # 或者使用 "ik_max_word"
                }
            }
```

```
        }
    },
    "mappings": {
        "properties": {
            "title": {"type": "text", "analyzer": "ik_analyzer"},
            "url": {"type": "keyword"},  # URL作为关键字处理
            "content": {"type": "text", "analyzer": "ik_analyzer"},
            "referenced_urls": {"type": "keyword"},  # 作为关键字处理
            "html_filename": {"type": "keyword"},
            "pr": {"type": "float"}  # PageRank值，用浮点数表示
        }
    }
}
```

`ik`分词器有两种模式，

- `ik_max_word`(最大词语粒度)：尽可能将文本切分为更多的词汇，倾向于最大化词语的数量，适用于尽可能多包含可能相关的词汇。
- `ik_smart`(智能分词)：尝试对文本进行较为合理的切分，倾向于减少无意义的短语和单字，提供更加简洁的分词结果，适用于精确的分词。

本项目中选择使用`ik_smart`模式，希望可以精确分词。

对`title`和`content`字段进行的分词处理，用于后续查询。

四、查询服务

使用`elasticsearch`框架实现查询，使用`flash`框架实现web服务

（一）站内查询

完整后端代码见`Search\app.py`，前端代码见`Search\templates\site_search.html`

```
search_query = {
    "query": {
        "function_score": {
            "query": {
                "multi_match": {
                    "query": query,
                    "fields": ["title", "content"],
                    "type": "best_fields"
                }
            },
            "functions": [
                *[
                    {"filter": {"match_phrase": {"content": term}}, "weight": 2}
                    for term in user_boost_terms
                ],
                 *[
                    {
                        "script_score": {
                            "script": {
                                "source": "if (doc['url'].value == params.url) {
return _score * 0.5 } else { return _score }",
```

```
                                "params": {"url": url}
                            }
                        }
                    }
                    for url in url_decrease_scores
                ]
            ],
            "score_mode": "sum",
            "boost_mode": "multiply"
        }
    },
    "collapse": {
        "field": "url"
    },
    "sort": [
        {"_score": {"order": "desc"}},
        {"pr": {"order": "desc"}}
    ],
    "size": 50
}
```

使用`multi_match`对`title`和`content`两个字段进行查询，使用`best_fields`匹配结果，实现站内查询，不要求用户输入必须作为一个词出现。查询结果首先按照相关度降序，其次按照pr值降序。

```
 resultItem.innerHTML = `
     <a href="${hit.url}" class="result-link" data-url="${hit.url}">${hit.title}
</a>
     <p class="url">${hit.url}</p>
     <p>${hit.content.substring(0, 300)}...</p>
     ${hit.folder && hit.html_filename ? `<a
href="/snapshot/${hit.folder}/${hit.html_filename}" target="_blank"
class="snapshot-link">网页快照</a>` : ''}
 `;
```

前端对每个查询结果，显示网页标题（点击跳转网页）,网页url，网页内容的前300字，网页快照。

查询效果如下：

（二）文档查询

完整后端代码见`Search\app.py`，前端代码见`Search\templates\file_search.html`

文档查询的查询语句构建与站内查询相同。但因为这种查询会筛选`referenced_urls`中含有文档链接的网页作为最终的查询结果，所以将直接查询的返回条数增加到了1000。

```
referenced_files = [ref for ref in hit['_source'].get('referenced_urls', [])
                            if any(ref.endswith(ext) for ext in ['.dox',
'.docx', '.pdf', '.rar', '.zip', '.xlsx', '.xls', '.ppt', '.pptx', '.jpg',
'.png'])]
if referenced_files:
        result = {
        'title': hit['_source'].get('title', 'No title'),
        'url': url,
        'content': hit['_source'].get('content', '')[:300] + '...',  # 截取前300个
字符
        'html_filename':html_filename,
        'folder':folder,
        'files':referenced_files
        }
        results.append(result)
```

这样就可以保证只返回带有文档链接的网页，并在前端将文档链接显示，用户点击即可查看或者下载。

查询效果如下：

文档链接点击即可下载或者查看对应文档。

（三）短语查询

完整后端代码见`Search\app.py`，前端代码见`Search\templates\phrase_search.html`

短语查询与站内查询的差别只在查询语句的构建：

```
search_query = {
    "query": {
        "function_score": {
            "query": {
                "multi_match": {
                    "query": query,
                    "fields": ["title", "content"],
                    "type": "phrase"
                }
            },
            "functions": [
                *[
                    {"filter": {"match_phrase": {"content": term}}, "weight": 2}
                    for term in user_boost_terms
                ],
```

```
                *[
                    {
                        "script_score": {
                            "script": {
                                "source": "if (doc['url'].value == params.url) {
return _score * 0.5 } else { return _score }",
                                "params": {"url": url}
                            }
                        }
                    }
                    for url in url_decrease_scores
                ]
            ],
            "score_mode": "sum",
            "boost_mode": "multiply"
        }
    },
    "collapse": {
        "field": "url"
    },
    "sort": [
        {"_score": {"order": "desc"}},
        {"pr": {"order": "desc"}}
    ],
    "size": 50
}
```

使用`phrase`查询类型，对于用户输入的查询词作为一个完整词汇进行查询，且要保证用户输入的多个词在网页内容中出现的顺序一致。

`phrase`查询语句中有`slop`参数，表示查询词之间最多可以有几个词的距离。代码中没有使用这个参数，使用的默认值0，也就各词之间必须是相连的。

查询效果如下：

（四）通配查询

完整后端代码见`Search\app.py`，前端代码见`Search\templates\wild_card_search.html`

通配查询与站内查询的差别也在于查询语句的构建：

```
search_query = {
    "query": {
        "function_score": {
            "query": {
                "bool": {
                    "should": [
                        {
                            "wildcard": {
                                "title": {
                                    "value": query,
                                    "boost": 1.0
                                }
                            }
                        },
                        {
                            "wildcard": {
                                "content": {
                                    "value": query,
                                    "boost": 1.0
```

```
                                }
                            }
                        }
                    ],
                    "minimum_should_match": 1
                }
            },
            "functions": [
                # Boost terms from user's search history
                *[
                    {"filter": {"match_phrase": {"content": term}}, "weight": 2}
                    for term in user_boost_terms
                ],
                # Decrease score for URLs that have been clicked using
script_score
                *[
                    {
                        "script_score": {
                            "script": {
                                "source": "if (doc['url'].value == params.url) {
return _score * 0.5 } else { return _score }",
                                "params": {"url": url}
                            }
                        }
                    }
                    for url in url_decrease_scores
                ]
            ],
            "score_mode": "sum",
            "boost_mode": "multiply"
        }
    },
    "collapse": {
        "field": "url"
    },
    "sort": [
        {"_score": {"order": "desc"}},  # 首先按照相关度排序
        {"pr": {"order": "desc"}}       # 其次按照PR值降序排序
    ],
    "size": 50  # 限制返回条数为50
}
```

因为通配查询`wile_card`不支持多字段匹配，不能像前面的三种查询直接使用`multi_match`，所以这里使用`should`语句，对`title`和`content`字段分别进行通配查询，要求两个查询至少满足一个，即`"minimum_should_match": 1`。

这实现了支持通配符`*`和`?`的查询，但这种查询需要对索引中的词条进行模式匹配，查询性能远不如其他查询。

查询效果如下：

（五）网页快照

爬取网页信息时，我记录的网页对应的`html`文件名`html_filename`，存储在域名对应的文件夹中。比如url为`https://ibs.nankai.edu.cn/n/1486.html`对应的`html`文件就存在`nankai_ibs`文件夹下。这样，根据查询到的网页的`url`和`html_filename`就可以找到对应的`html`文件，实现网页快照。

```
results = []
for hit in response['hits']['hits']:
    html_filename = hit['_source'].get('html_filename','')
    url = hit['_source'].get('url','')
    folder=''
    if 'ai.nankai.edu.cn' in url:
        folder='nankai_ai'
    elif 'bs.nankai.edu.cn' in url:
        folder='nankai_bs'
    elif 'cc.nankai.edu.cn' in url:
        folder='nankai_cc'
    elif 'ceo.nankai.edu.cn' in url:
        folder='nankai_ceo'
    elif 'cs.nankai.edu.cn' in url:
        folder='nankai_cs'
    elif 'cyber.nankai.edu.cn' in url:
        folder='nankai_cyber'
    elif 'finance.nankai.edu.cn' in url:
```

```
            folder='nankai_finance'
        ...
        else:
            folder=None
        result = {
            'title': hit['_source'].get('title', 'No title'),
            'url': url,
            'content': hit['_source'].get('content', '')[:300] + '...',
            'html_filename':html_filename,
            'folder':folder
        }
        results.append(result)
    return jsonify({'hits':results})
```

后端将网页快照链接到对应的文件路径就可以点击查看对应的快照文件。

```
${hit.folder && hit.html_filename ? `<a
href="/snapshot/${hit.folder}/${hit.html_filename}" target="_blank"
class="snapshot-link">网页快照</a>` : ''}
```

查询效果如图：

点击即可显示文件：

五、用户功能

（一）注册与登录

实现了简单的用户注册和登录功能。因为本项目的重点不是这一部分，所以没有连接数据库，直接将用户信息按照`[{username,password,search_history,clicked_links}]`记录在`users.json`中。来实现用户的查询日志，个性化查询和个性化推荐功能。

登录页面如图：

注册页面如图：

（二）查询日志

新添加了一个用户主页页面，在主页页面会显示用户的查询历史和访问过的网页。

```
# 捕捉用户点击的url
@app.route('/update_click', methods=['POST'])
def update_click():
    print("点击触发")
    if 'user' in session:
        url = request.json.get('url')
        if url:
            update_clicked_links(session['user'], url)
            return jsonify({"status": "success"}), 200
        else:
            return jsonify({"status": "failed", "message": "URL not provided"}),
400
    else:
        print("未登录")
    # return jsonify({"status": "failed", "message": "Not logged in"}), 403
    return jsonify({"status": "success"}), 200
```

添加新路由，在用户点击时调用保存用户访问的url。

```
if 'user' in session:
    update_search_history(session['user'],query)
```

在原本的查询最后，如果现在的回话中有登录的用户，则将查询词添加到用户的查询历史中。

```
 // 监听点击行为
document.addEventListener('DOMContentLoaded', function() {
    document.body.addEventListener('click', function(event) {
            if (event.target.classList.contains('result-link')) {
                console.log('Result link clicked:',
 event.target.getAttribute('data-url'));  // 调试信息
                handleResultClick(event, event.target.getAttribute('data-url'));
```

```
            }
        });
});

function handleResultClick(event, url) {
        event.preventDefault();
        console.log('Handling result click for URL:', url);  // 调试信息
        fetch('/update_click', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({url: url})
        })
        .then(response => response.json())
        .then(data => {
            if (data.status === 'success') {
                console.log('Click recorded successfully');  // 调试信息
                window.location.href = url;
            } else {
                console.error('Failed to update click:', data.message);
            }
        })
        .catch(error => {
            console.error('Error updating click:', error);
        });
    }
```

前段添加`<script>`标签监听用户的点击行为，与后端对接后，就可以在用户查询和点击链接时保存历史到用户json文件中。最终在用户主页显示用户的查询日志`search_history`和访问历史`clicked_links`即可。

```
<h3>搜索历史</h3>
 <ul id="search-history" class="history-list">
        {% if user.search_history %}
            {% for query in user.search_history %}
                <li>{{ query }}</li>
            {% endfor %}
        {% else %}
            <li>暂无搜索历史</li>
        {% endif %}
    </ul>

    <h3>访问记录</h3>
    <ul id="clicked-links" class="history-list">
        {% if user.clicked_links %}
            {% for link in user.clicked_links %}
                <li><a href="{{ link }}" target="_blank">{{ link }}</a></li>
            {% endfor %}
        {% else %}
            <li>暂无点击记录</li>
        {% endif %}
    </ul>
```

主页显示效果如下：

（三）个性化查询

个性化查询是根据用户的查询历史和点击过的网页来提供不同的查询结果排序，基本思路是将用户查询过的term的权重提高，保证能够优先提供用户感兴趣的内容；同时将用户访问过的网页的权重降低，保证用户不会反复查询到相同网页。

在构建查询语句时：

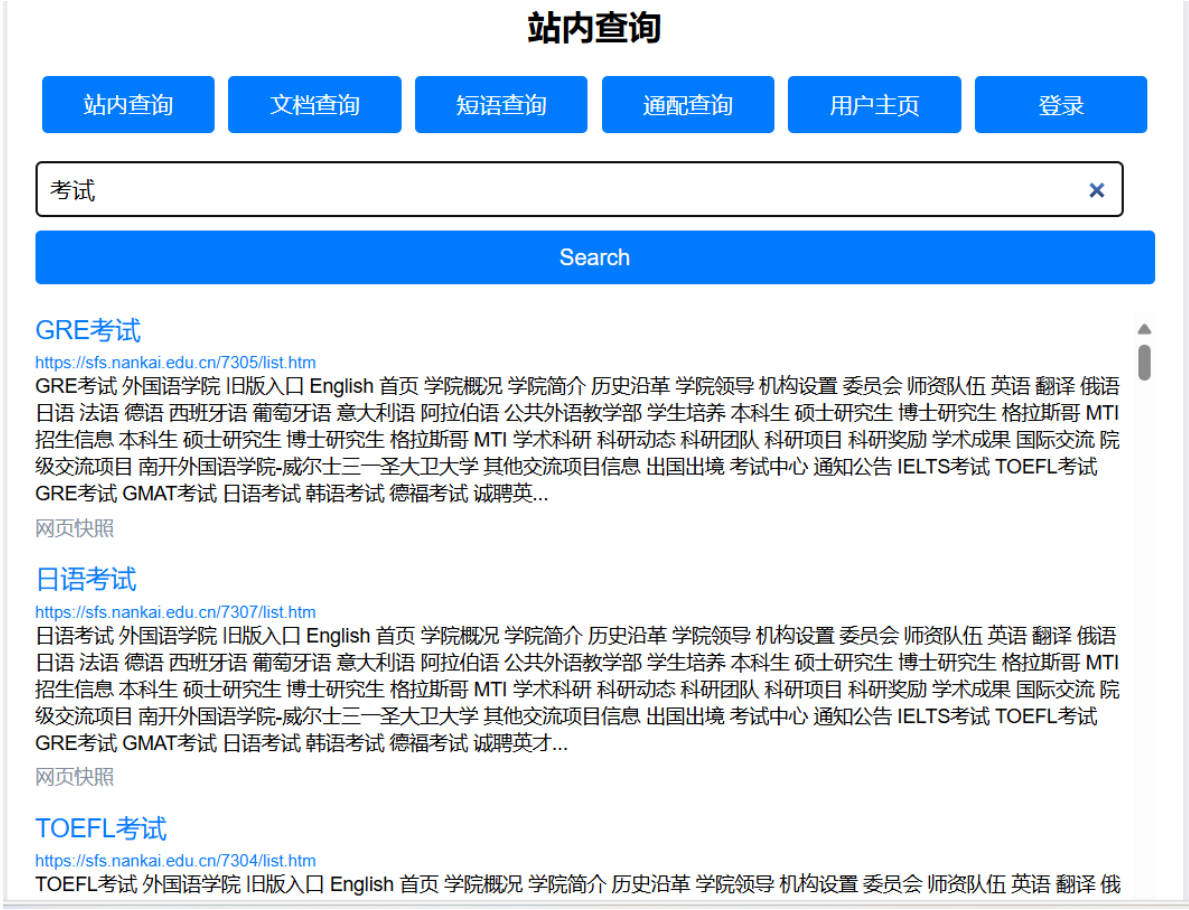
```
"functions": [
                    # Boost terms from user's search history
                    *[
                        {"filter": {"match_phrase": {"content": term}},
"weight": 2}
                        for term in user_boost_terms
                    ],
                    # Decrease score for URLs that have been clicked
                     *[
                        {
                            "script_score": {
                                "script": {
                                    "source": "if (doc['url'].value ==
params.url) { return _score * 0.5 } else { return _score }",
                                    "params": {"url": url}
                                }
                            }
                        }
                        for url in url_decrease_scores
                    ]
```

```
                ],
```

对于用户查询历史中的term，每个的权重值+2，对用户访问过的url，相关度得分降为原本的一半。

查询效果如下：

用户初次站内查询“考试”得到的结果如下，`https://sfs.nankai.edu.cn/7305/list.htm`这个网页为相关度第一的结果。



点击第一个网页后，用户主页的访问历史中添加该url，再次查询"考试"，得到结果：

可以看到`https://sfs.nankai.edu.cn/7305/list.htm`这个网页的相关度降低，再次查询时这个网页不再是第一个结果。

（四）个性化推荐

个性化推荐的主要思路是对于用户查询历史最近的5条进行查询，并对用户的所有查询历史term权重提高2，访问过的url的相关度降低到原来的一半。

```
if has_search_history:
    user_boost_terms = [term for term in user['search_history']]
    url_decrease_scores = [url for url in user['clicked_links']]
    query = " ".join(user_boost_terms[-5:])
    search_query = {
        "query": {
            "function_score": {
                "query": {
                    "multi_match": {
                        "query": query,
                        "fields": ["title", "content"],
                        "type": "best_fields"
                    }
                },
                "functions": [
                    *[
                        {"filter": {"match_phrase": {"content": term}},
"weight": 2}
                        for term in user_boost_terms
                    ],
                    *[
                        {
```

```
                            "script_score": {
                                "script": {
                                    "source": "if (doc['url'].value ==
params.url) { return _score * 0.5 } else { return _score }",
                                    "params": {"url": url}
                                }
                            }
                        }
                        for url in url_decrease_scores
                    ]
                ],
                "score_mode": "sum",
                "boost_mode": "multiply"
            }
        },
        "collapse": {
            "field": "url"
        },
        "sort": [
            {"_score": {"order": "desc"}},
            {"pr": {"order": "desc"}}
        ],
        "size": 100  # 推荐的数量
    }
else: # 没有查询历史则直接返回权重最高的20个网页
    search_query = {
        "query": {
            "match_all": {}
        },
        "sort": [
            {"_score": {"order": "desc"}},
            {"pr": {"order": "desc"}}
        ],
        "size": 20  # 返回20个最高权重的网页
    }
```

这样就可以根据用户的查询历史筛选出用户近期感兴趣的网页，并避免向用户重复推荐其访问过的网页。

在用户主页中增加推荐网页的显示：

```
<h3>为你推荐</h3>
<div id="results"></div>
<script>
     document.addEventListener('DOMContentLoaded', function() {
         fetchRecommendations();
         // 使用事件委托处理所有推荐链接的点击事件
         document.body.addEventListener('click', function(event) {
             if (event.target.classList.contains('result-link')) {
                 console.log('Recommendation link clicked:',
event.target.getAttribute('data-url'));  // 调试信息
                 handleResultClick(event, event.target.getAttribute('data-
url'));
             }
          });
```

```
    });
    function fetchRecommendations() {
        fetch('/recommendations', {
            method: 'GET'
        })
        .then(response => response.json())
        .then(data => {
            var recommendationsDiv = document.getElementById('results');
            recommendationsDiv.innerHTML = '';
            data.hits.forEach(hit => {
                var recommendationItem = document.createElement('div');
                recommendationItem.innerHTML = `
                    <a href="${hit.url}" class="result-link" data-
url="${hit.url}">${hit.title}</a>
                    <p class="url">${hit.url}</p>
                    <p>${hit.content.substring(0, 300)}...</p>
                    ${hit.folder && hit.html_filename ? `<a
href="/snapshot/${hit.folder}/${hit.html_filename}" target="_blank"
class="snapshot-link">网页快照</a>` : ''}
                `;
                recommendationsDiv.appendChild(recommendationItem);
            });
        })
        .catch(error => {
        console.error('Error fetching recommendations:', error);
        });
    }
    function handleResultClick(event, url) {
        event.preventDefault();
        console.log('Handling result click for URL:', url);  // 调试信息
        fetch('/update_click', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({url: url})
        })
        .then(response => response.json())
        .then(data => {
            if (data.status === 'success') {
                console.log('Click recorded successfully');  // 调试信息
                window.location.href = url;
            } else {
                console.error('Failed to update click:', data.message);
            }
        })
        .catch(error => {
            console.error('Error updating click:', error);
        });
    }
</script>
```

并保证用户点击推荐的网页的行为也会记录到访问历史。

用户主页的推荐如下：

为你推荐

我的“成长记” -南开大学

https://weekly.nankai.edu.cn/index/article/articleinfo.html?doc_id=1064

我的“成长记” -南开大学 我的“成长记” 期次：第933期 阅读：13 王子祯 我的“记者”生活 来到南开，最大的成就莫过我加入了《南开大学报》记者团。一进校园，铺天盖地的社团招聘传单、海报令人眼花缭乱。我像所有大一新生一样没有目标，只是觉得哪里都好，什么都有兴趣，直到一个小册子《吾爱吾团》被送...

网页快照

“以夏之名，与你相约”——文学院第六届运动会成功举行

http://wxy.nankai.edu.cn/2018/0515/c15915a197444/page.htm

“以夏之名，与你相约”——文学院第六届运动会成功举行 教室预定 | 校友合作 | 交流合作 | English | 登录 | 首页 学院概况 学院简介 各委员会 组织机构 教学机构 研究机构 学科专业 学院新闻 师资团队 中国语言文学系 东方艺术系 艺术设计系 文化素质教学部 党政学工 学院办公室 教学办公室 研究生办公室 实验教学示范中心 图书资料中心 《文学与文化》编辑部 荣休教师名录 党务行政 党团机构 行政机构 师资人事 申请表格 规章制度 图书资料 学术研究 新闻通告 学术会议 学术讲座 研究项目 研究成果 文件下载 教育教学 本科教育 研究生教育 留学生教育 实验教学 访问进修 招...

网页快照

商学院举办第四届研究生趣味运动会

https://ibs.nankai.edu.cn/n/429.html

商学院举办第四届研究生趣味运动会 新网站入口 教工登录 南开大学 English 首 页 | 学院简介 | 科学研究 | 师资队伍 | 学科专业 | 教学教务 | 实验教学 | 对外交流 | 学生工作 | 教育培训 | 校友工作 | 人才招聘 首 页 学院动态 学术活动 通知公告 教学招生 教授声音 新闻专题 精品课程 本科精品课程 研究生精品课程 商学院举办第四届研究生趣味运动会 浏览次数: 9117 更新时间: 2011年11月15日 南开新闻网讯(通讯员 徐琳 刘翰 王玫) 11月11日，南开大学商学院学工办主办、研究生会承办的第四届研究生趣味运动会在MBA大楼多功能厅举行，商学院研...

网页快照

退出