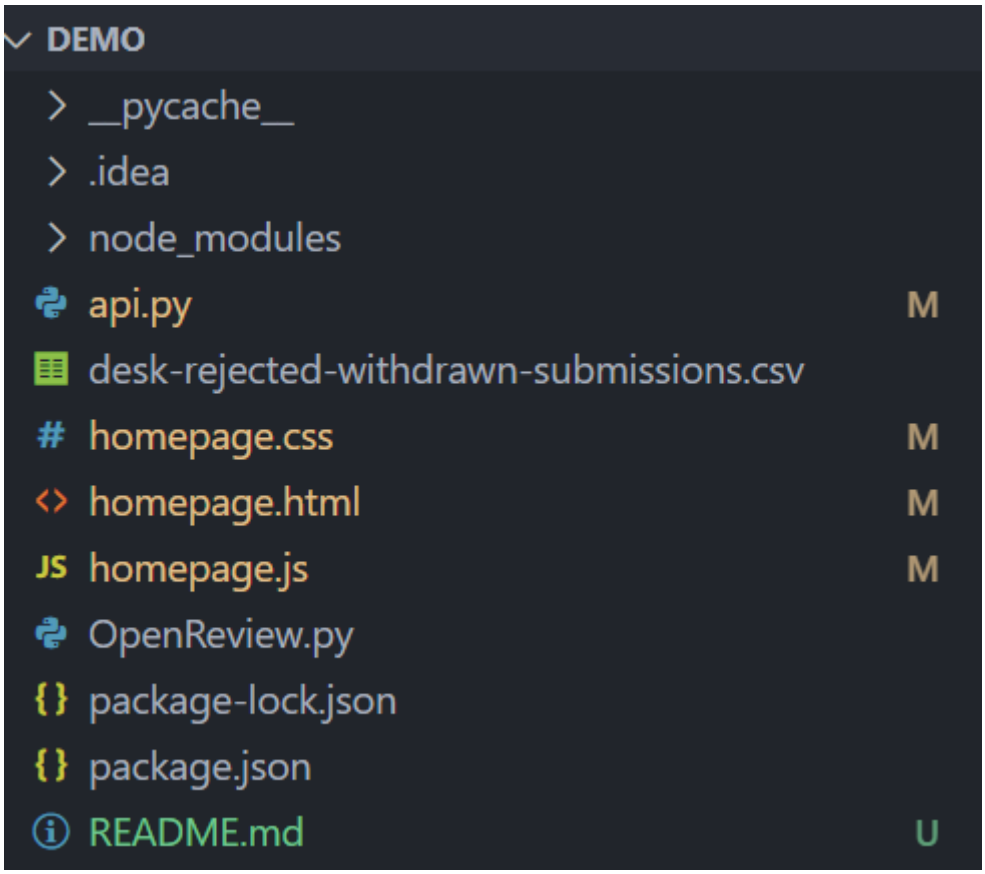
期末大作业实验报告

`Python第四小组`

作业要求

```
# 题目三：爬虫

## 题A：

具体描述：

* 从以下网站爬取`ICLR2023`的论文数据，输出所有论文的pdf链接；
* 爬取每篇论文的标题、作者和论文摘要，用这些信息制作一个新的网页（离线网页即可）；
* 针对论文的关键词绘制词云。

[](https://openreview.net/group?id=ICLR.cc/2023/Conference)
```

项目结构



项目主要包括三个部分：

- 前端页面部分：
  - homepage.html
  - homepage.js
  - homepage.css
- api接口部分：

    - api.py
- 爬虫部分：
    - OpenReview.py

其中，针对关键词绘制词云，以及扩展的搜索功能都在api接口部分完成，

爬取论文数据，并清洗数据，获得每篇论文的标题、作者和论文摘要以及下载链接，这一部分在爬虫部分完成，

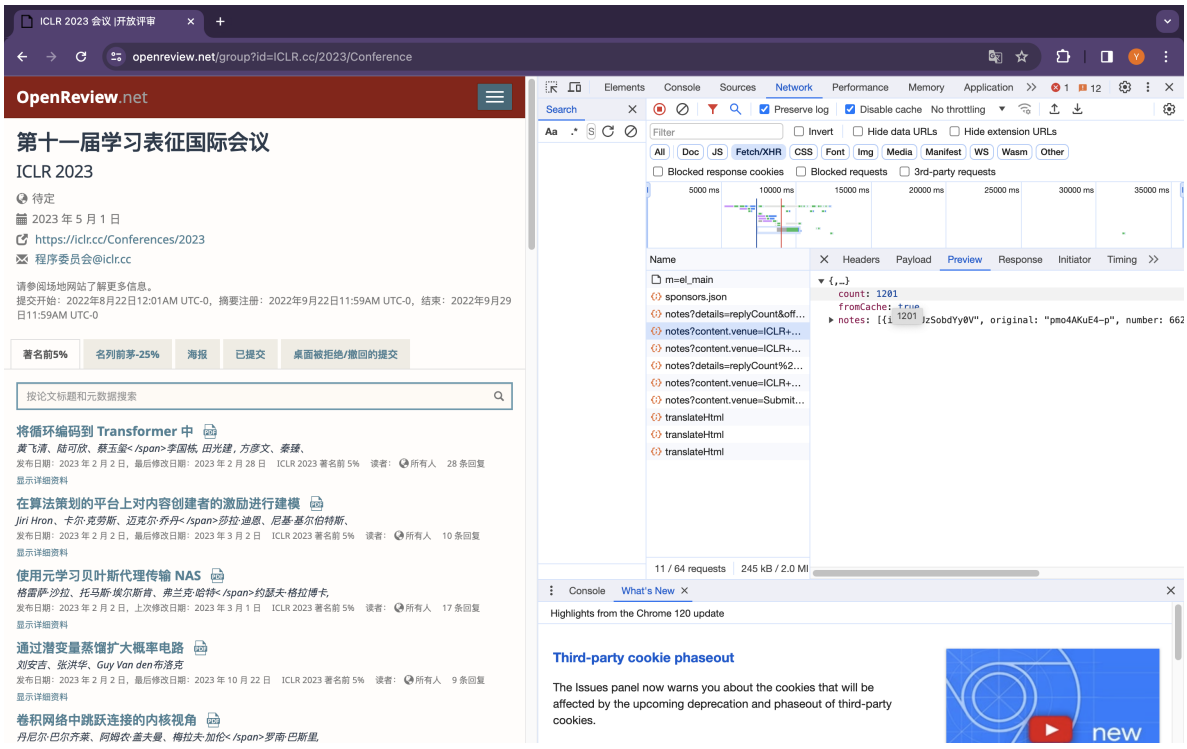
而调用api接口，进行页面渲染，这部分在前端页面部分完成。

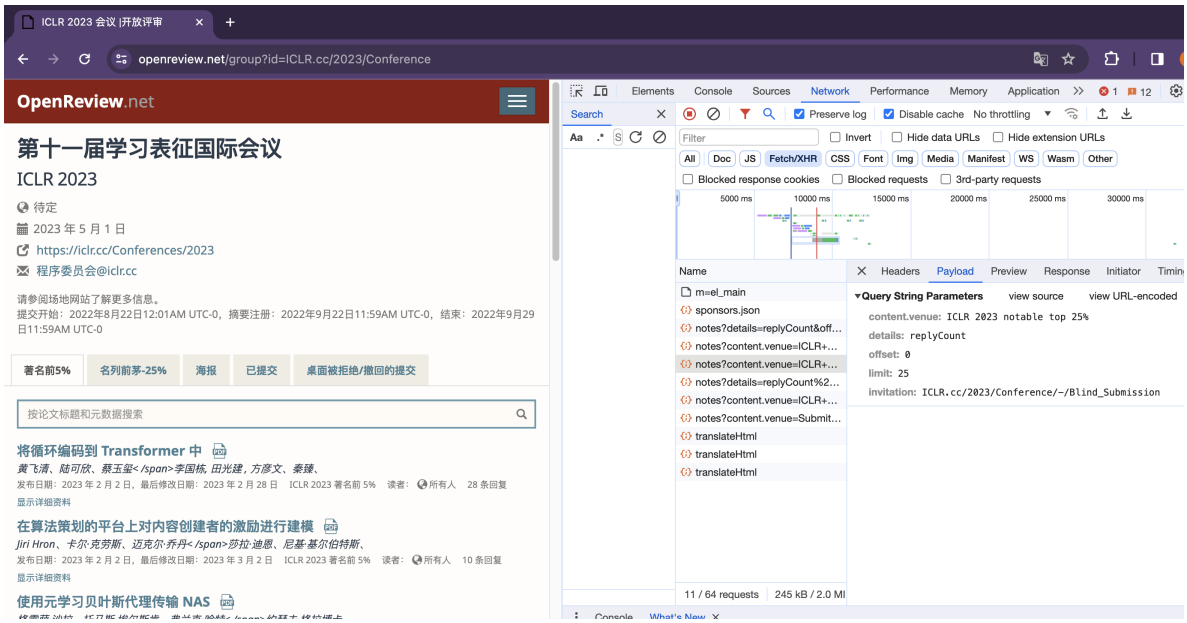
在开发期间，我们使用git进行版本管理，所有源码可以在这里找到。

项目细节

爬虫分析

首先对原来的页面进行分析，使用F12，观察它所采用的传输传输数据方式，通过***Network***页面中的***Fetch/XHR***选项对这个网页的传送文件进行观察。



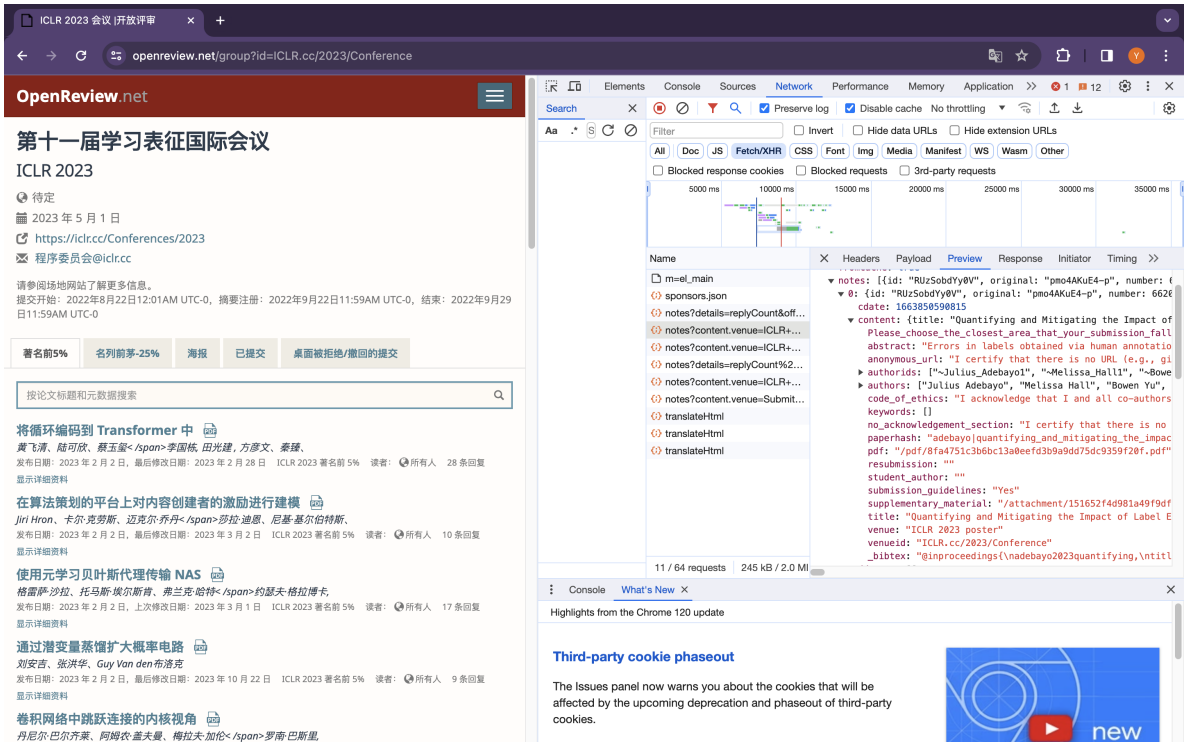


我们可以看到，该网页进行数据传输的方式十分简单，就是通过向服务器发送request请求，然后由服务器返回一份json文件并最终在前端页面执行渲染，所以我们所需要做的，就是理解网站作者在这些json文件中进行传输的内容以及方式，并最终将所有的数据获取到本地。

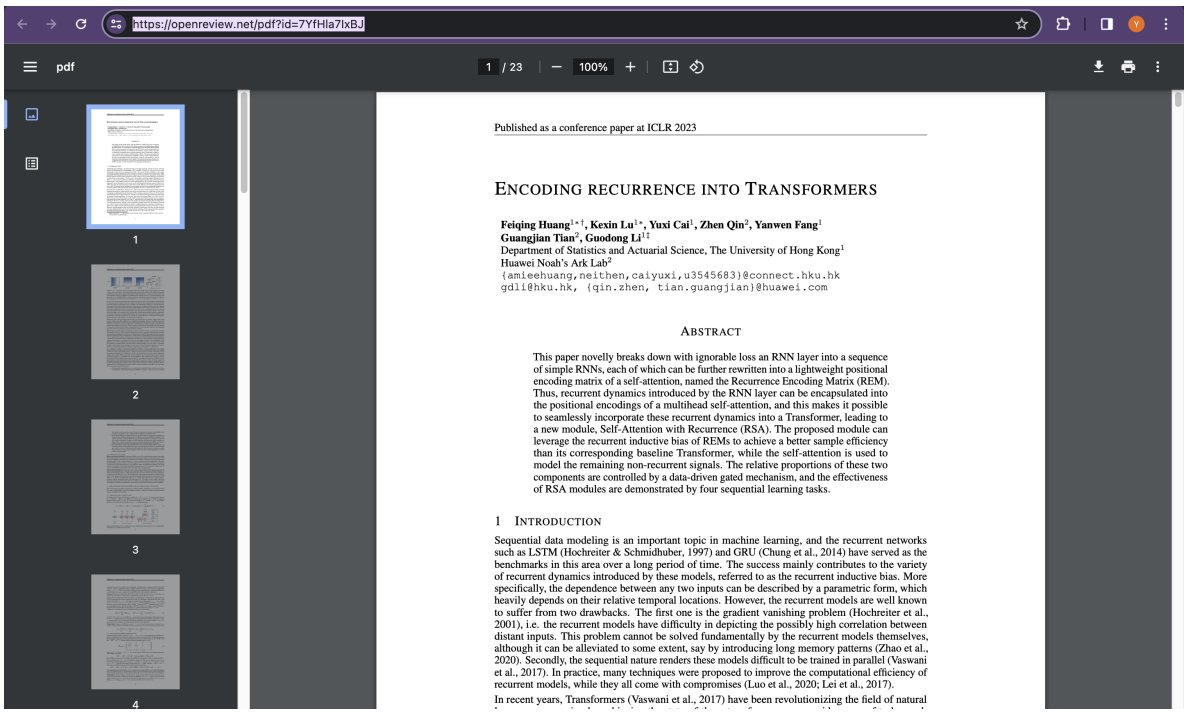
接下来对Fetch到的json文件进行分析。从第一张图的count，我们可以知道在该文件中一共包含1201份文章，再从第二张图的Payload中得到content venue、limit以及offset信息，limit即使每一个页面中所包含的文章数量，而offset则是这个页面从数据库中的第几篇文章开始。在知道这些细节之后，考虑完这些我们便可以开始考虑如何抓取json文件以及如何从json文件中提取出我们所需要的信息。

Request URL: https://api.openreview.net/notes?content.venue=ICLR+2023+poster&details=replyCount&offset=0&limit=25&invitation=ICLR.cc%2F2023%2FConference%2F-%2FBlind_Submission

这是浏览器向服务器发出的Request请求，在？之后的内容可以看出便是写在payload中的内容，页面通过改变content.venue从而改变要获取的文章类型，因此只需要在爬虫之中修改请求url中的content.venue就可以实现不同分类的抓取。



接下来就是对我们所需要内容如何进行获取进行解析，点开Preview，点开notes，里面存放好了25份数据而note中的content下则依次存放着文章的title、keyword、author、pdf链接等内容，其中要使pdf变为可打开形式，我们则需要对该内容进行处理。



由图中所见，pdf即为原网站的域名拼接上pdf?id=...来实现pdf的打开，所以只需将note-contents中文章的id拼贴上去即可获得pdf链接。

至此，爬虫的解析部分就已经完成，之后只需要在第一次为获得Payload以及count等内容发起请求后，循环发起请求直到offset+limit >= count即可完成所有文章的获取。

爬虫实现

我们共导入了以下包，其中第三行导入了线程池用来实现多线程爬虫，第四行用于处理url：

```
import requests
import pandas as pd
from concurrent.futures import ThreadPoolExecutor
from urllib.parse import urlencode, urlunparse, urlparse, parse_qsl
```

首先，规定了实体格式和目标网页每页文章数量：

```
pd_dic = {"titles": [], "authors": [], "keywords": [], "pdfs": [], "abstracts":
[]}
limit = 25
```

接下来定义了须爬取的五大类型对应的基础url：

```
# 所有ICLR前%5的论文
url_0 = "https://api.openreview.net/notes?
content.venue=ICLR+2023+notable+top+5%25&details=replyCount&offset=25&limit=25&in
vitation=ICLR.cc%2F2023%2FConference%2F-%2FBlind_Submission"
temp_resp_0 = requests.get(url_0)
count_0 = temp_resp_0.json()["count"]
# 所有ICLR前%25的论文
url_1 = "https://api.openreview.net/notes?
content.venue=ICLR+2023+notable+top+25%25&details=replyCount&offset=0&limit=25&in
vitation=ICLR.cc%2F2023%2FConference%2F-%2FBlind_Submission"
temp_resp_1 = requests.get(url_1)
count_1 = temp_resp_1.json()["count"]
# 所有发表的论文
```

```
url_2 = "https://api.openreview.net/notes?
content.venue=ICLR+2023+poster&details=replyCount&offset=0&limit=25&invitation=IC
LR.cc%2F2023%2FConference%2F-%2FBlind_Submission"
temp_resp_2 = requests.get(url_2)
count_2 = temp_resp_2.json()["count"]
# 所有提交的论文
url_3 = "https://api.openreview.net/notes?
content.venue=Submitted+to+ICLR+2023&details=replyCount&offset=0&limit=25&invitat
ion=ICLR.cc%2F2023%2FConference%2F-%2FBlind_Submission"
temp_resp_3 = requests.get(url_3)
count_3 = temp_resp_3.json()["count"]
# desk-rejected-withdrawn-submissions
url_4 = "https://api.openreview.net/notes?
details=replyCount%2Cinvitation%2Coriginal&offset=0&limit=25&invitation=ICLR.cc%2
F2023%2FConference%2F-%2FWithdrawn_Submission"
temp_resp_4 = requests.get(url_4)
count_4 = temp_resp_4.json()["count"]
```

然后实现了爬取每一页，并将数据存入dict字典的函数：

```
def get_one_page(_url, _dict):
    try:
        resp = requests.get(_url)
        resp.raise_for_status()  # 如果响应状态码不是200，就主动抛出异常
    except requests.RequestException as e:
        print(f"请求{_url}时发生错误：{e}")
        return

    notes = resp.json()["notes"]
    for note in notes:
        id = note["id"]
        content = note["content"]
        pdf = f"https://openreview.net/pdf?id={id}"
        _dict["titles"].append(content["title"])
        _dict["authors"].append(content["authors"])
        _dict["keywords"].append(content["keywords"])
        _dict["pdfs"].append(pdf)
        _dict["abstracts"].append(content["abstract"])
        print("ok")
```

接下来是运用多线程，对于需要爬取的每一页，提交一个任务，用最多50个线程并行爬取。

```
def Thread_Method(__url, _count, _dict = pd_dic):
    with ThreadPoolExecutor(50) as t:
        for i in range(0, _count, limit):
            # 解析URL
            url_parts = list(urlparse(__url))
            # 解析查询参数
            query = dict(parse_qsl(url_parts[4]))
            # 更新offset参数
            query.update({"offset": str(i)})
            # 重新生成查询参数字符串
            url_parts[4] = urlencode(query)
            # 重新生成URL
            _url = urlunparse(url_parts)
            t.submit(get_one_page, _url, _dict)
```

最后，封装为核心功能，并测试文件功能是否正常。

```
def func(mode):
    dic = {"titles": [], "authors": [], "keywords": [], "pdfs": [], "abstracts":
[]}
    url = globals()['url_' + str(mode)]
    count = globals()['count_' + str(mode)]
    Thread_Method(url, count, dic)
    return dic

if __name__ == "__main__":
    pd_dic = func(4)
    db = pd.DataFrame(pd_dic)
    db.to_csv("desk-rejected-withdrawn-submissions.csv")
```

测试后，成功获取了csv文件`desk-rejected-withdrawn-submissions.csv`，文件内容正确。

api接口实现

我们共导入了以下包，包括刚刚爬虫部分封装的func()函数。

```
from flask import Flask, jsonify, request, send_file
from flask_cors import cross_origin
from OpenReview import func
from wordcloud import WordCloud
from io import BytesIO
```

我们采用Flask框架作为服务端架构，首先创建实例app，并初始化data为None。

```
app = Flask(__name__)
data = None
```

在路由`/api/data`处封装api，并允许跨源访问。

```
@app.route('/api/data', methods=['GET'])
@cross_origin()
def get_data():
    mode = int(request.args.get('mode'))

    global data
    data = func(mode)

    return jsonify(data)  # 返回指定页的数据
```

封装词云生成api，先把所有文章的关键词展平，再调用WordCloud库绘制词云，并使用flask库的send_file()方法将图片发送到前端。

```
@app.route('/api/keywords-wordcloud', methods=['GET'])
@cross_origin()
def generate_keywords_wordcloud():
    global data
    flat_list = []
    for item in data['keywords']:
        for words in item:
```

```
            flat_list.append(words)
    keywords_text = ",".join(flat_list)

    # 生成词云
    wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(keywords_text)

    # 保存词云为图片
    img_buffer = BytesIO()
    wordcloud.to_image().save(img_buffer, format='PNG')
    img_buffer.seek(0)

    return send_file(img_buffer, mimetype='image/png')
```

拓展了搜索功能，时间有限没有做基于正则表达式的部分匹配，仅支持完全匹配，且仅可搜索当前页面上显示的文章。

```
@app.route('/api/search', methods=['GET'])
@cross_origin()
def search_data():
    keyword = request.args.get('keyword')
    # 在全局变量 data 中搜索关键词并返回结果
    result = search_in_data(keyword)
    return jsonify(result)

def search_in_data(keyword):
    global data

    # 在 data 中搜索关键词，找到匹配的索引
    result = []
    for i, sublist in enumerate(data['keywords']):
        if keyword in sublist:
            result.append(i)
    return result
```

启动Flask实例，监听5000端口。

```
if __name__ == '__main__':
    app.run(port=5000)
```

前端代码实现

前端代码共包含三个部分，其中页面的标签栏和搜索栏是静态网页，而词云与论文部分是动态加载的。

静态页面结构

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>论文聚合平台</title>
    <link rel="stylesheet" type="text/css" href="homepage.css">
</head>
<body onload="initPage()">
    <header class="header">
```

```
        <img class="logo"
src="https://th.bing.com/th/id/OIG.3w2auwiZ7abMx4qVqOrA?
w=1024&h=1024&rs=1&pid=ImgDetMain">
        <a href="homepage.html" class="home">论文聚合平台</a>
        <div class="search">
            <input id="searchInput" type="text" placeholder="搜索">
            <button onclick="search()">
                <svg width="18" height="18" viewBox="0 0 24 24"
fill="currentColor">
                    <g fill-rule="evenodd" clip-rule="evenodd">
                        <path d="M11.5 18.389c3.875 0 7-3.118 7-6.945 0-3.826-
3.125-6.944-7-6.944s-7 3.118-7 6.944 3.125 6.945 7 6.945Zm0 1.5c4.694 0 8.5-3.78
8.5-8.445C20 6.781 16.194 3 11.5 3S3 6.78 3 11.444c0 4.664 3.806 8.445 8.5
8.445Z"></path>
                        <path d="M16.47 16.97a.75.75 0 0 1 1.06 0l3.5 3.5a.75.75
0 1 1-1.06 1.06l-3.5-3.5a.75.75 0 0 1 0-1.06Z"></path>
                    </g>
                </svg>
            </button>
        </div>
    </header>
    <h1>The Eleventh International Conference on Learning Representations</h1>
    <div style="box-shadow: 0 1px 3px hsla(0,0%,7%,.1);display: flex;flex-
direction: column;align-items: center">
        <div class="tabCardContainer">
            <button id="tab_0" class="tabCard" onclick="clickTab(0)"
style="color: #121212">Notable-top-5%</button>
            <button id="tab_1" class="tabCard" onclick="clickTab(1)"
style="color: #121212">Notable-top-25%</button>
            <button id="tab_2" class="tabCard" onclick="clickTab(2)"
style="color: #121212">Poster</button>
            <button id="tab_3" class="tabCard" onclick="clickTab(3)"
style="color: #121212">Submitted</button>
            <button id="tab_4" class="tabCard" onclick="clickTab(4)"
style="color: #121212">Desk Rejected/Withdrawn Submissions</button>
        </div>
        <div id="loading"><p>加载中...</p></div>
        <div class="img-container"></div>
        <div class="article-container"></div>
    </div>
    <script src="homepage.js"></script>
</body>
</html>
```

在搜索框和文章容器部分，参考了知乎的样式表，包括svg搜索图标等。

前端代码

鉴于本次课程为Python，故不详细展开介绍js部分。

```
const loading = document.getElementById('loading');
let data;

function clickTab(x) {
    let elem = document.getElementById("tab_" + x)
    if (elem.style.color === "rgb(18, 18, 18)")
        elem.style.color = "#056de8";
```

```
    for (let i = 0; i < 5; i++) {
        let temp = document.getElementById("tab_" + i)
        if (i !== x && temp.style.color === "rgb(5, 109, 232)")
            temp.style.color = "#121212";
    }
    loading.style.display = 'flex';
    getData(x).then((d) => {
        data = d;
        console.log(data);
        displayPage();
    });
}

async function getData(mode) {
    try {
        const response = await fetch('http://localhost:5000/api/data?mode=' +
mode);
        const data = await response.json();
        // 使用返回的数据渲染页面
        loading.style.display = 'none';
        return data;
    } catch (error) {
        console.error('Error:', error);
    }
}

function displayPage() {
    if (data.length === 0) console.log("获取文章失败");
    const container = document.querySelector('.article-container');
    if (container) {
        container.innerHTML = '';
        getKeywordsWordcloud()
        // 遍历数据数组，渲染每个文章块
        for (let i = 0; i < data.authors.length; i++) {
            // 创建一个新的文章块元素
            let article = document.createElement('div');
            article.className = 'article';

            // 填充文章块的内容
            article.innerHTML = `
                <h2>${data.titles[i]}</h2>
                <p>Author: ${data.authors[i]}</p>
                <p>Keywords: ${data.keywords[i]}</p>
 <p>Abstracts: ${data.abstracts[i]}</p>
                <a href="${data.pdfs[i]}" target="_blank" class="download-
button">Download PDF</a>
            `;

            // 将文章块添加到 article-container 中
            container.appendChild(article);
        }
    } else {
        console.error('Article container not found!');
    }
}

function initPage() {
    clickTab(0)
```

```
}

function getKeywordsWordcloud() {
    const container = document.querySelector('.img-container');
    container.innerHTML = '';
    fetch(`http://localhost:5000/api/keywords-wordcloud`)
        .then(response => response.blob())
        .then(blob => {
            // 创建一个表示图片的URL
            let imageUrl = URL.createObjectURL(blob);

            // 在页面上显示词云图片
            let imgElement = document.createElement('img');
            imgElement.src = imageUrl;
            container.appendChild(imgElement);
        })
        .catch(error => console.error('Error fetching wordcloud:', error));
}

function search(){
    // 获取输入的关键词
    let keyword = document.getElementById('searchInput').value;
    // 发送搜索请求到后端API
    fetch(`http://localhost:5000/api/search?
keyword=${encodeURIComponent(keyword)}`)
        .then(response => response.json())
        .then(data => {
            // 处理搜索结果
            console.log(data);
            showSearchResult(data)
        })
        .catch(error => {
            console.error('Error during search:', error);
        });
}

function showSearchResult(indexs) {
    const img_container = document.querySelector('.img-container');
    img_container.innerHTML = '';
    const art_container = document.querySelector('.article-container');
    art_container.innerHTML = '';
    for (let i = 0; i < indexs.length; i++) {
            // 创建一个新的文章块元素
            let article = document.createElement('div');
            article.className = 'article';

            // 填充文章块的内容
            article.innerHTML = `
                <h2>${data.titles[indexs[i]]}</h2>
                <p>Author: ${data.authors[indexs[i]]}</p>
                <p>Keywords: ${data.keywords[indexs[i]]}</p>
                 <p>Abstracts: ${data.abstracts[i]}</p>
                <a href="${data.pdfs[indexs[i]]}" target="_blank"
class="download">Download PDF</a>
            `;

            // 将文章块添加到 article-container 中
            art_container.appendChild(article);
```

```
        }
}
```

样式表

鉴于本次课程为Python，故不详细展开介绍css部分。

```
body {
    font-family: -apple-system,BlinkMacSystemFont,Helvetica Neue,PingFang SC,Microsoft YaHei,Source Han Sans SC,Noto Sans CJK SC,WenQuanYi Micro Hei,sans-serif;
    color: #121212;
    font-size: 15px;
    display: flex;
    flex-direction: column;
    padding: 80px;
}
h1 {
    text-align: center;
    padding: 20px;
}

.header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    background-color: #fff;
    height: 100px;
    padding: 0 10px;
    box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
}

.logo {
    width: 80px;
    height: 80px;
}

.home {
    margin-right: auto;
    padding: 0 40px;
    color: #121212;
    text-align: center;
    font-size: 25px;
    font-weight: 600;
    text-decoration:none;
}

.search {
    display: flex;
    align-items: center;
    background-color: #f5f5f5;
    border-radius: 20px;
    padding: 0 20px;
    height: 40px;
    width: 300px;
}
```

```
.search input {
    border: none;
    outline: none;
    background-color: transparent;
    margin-left: 10px;
    font-size: 16px;
    width: 100%;
}

.search input::placeholder {
    color: #999;
}

.search button {
    border: none;
    outline: none;
    background-color: transparent;
    cursor: pointer;
}

.tabCardContainer {
    display: flex;
    align-items: center;
    justify-content: space-between;
    height: 58px;
    width: 100%;
}

.tabCard {
    cursor: pointer;
    margin: 0 22px;
    border: none;
    outline: none;
    background-color: transparent;
    font-size: 16px;
}

#loading {
    display: flex;
    align-items: center;
    justify-content: center;
    width:100%;
    height:80px;
    text-align: center;
}
#loading p {
    font-size: 20px;
}

.article {
    color: #121212;
    font-family: -apple-system,BlinkMacSystemFont,Helvetica Neue,PingFang SC,Microsoft YaHei,Source Han Sans SC,Noto Sans CJK SC,WenQuanYi Micro Hei,sans-serif;
    font-size: 15px;
    -webkit-tap-highlight-color: rgba(18,18,18,0);
    background: #fff;
    box-sizing: border-box;
```

```
    border-radius: 0;
    outline: none;
    overflow: initial;
    position: relative;
    padding: 20px;
    border-bottom: 1px solid #f5f5f5;
    box-shadow: none;
    margin-bottom: 0;
}

.img-container {
    display: flex;
    justify-items: center;
    align-items: center;
}

.download-button {
    background-color: #999999;
    color: white;
    font-size: 20px;
}
```

成员分工及贡献占比

本小组共5名成员，分工及贡献占比如下：

学号+姓名	分工	贡献占比
61522312万奕含	爬虫部分及文档的完成	100%
61522314吴清晏	api接口实现及文档撰写	100%
71122207董子翔	前端js代码完善	99%
58122327李家豪	前端页面搭建，文档的完成	99%
61522122李宗辉	前端css样式表调整及制作视频汇报	99%