


信息检索 课程实验报告

测试链接: <http://sdubaidu.applinzi.com/baidu/web/index.php>

项目主页: <http://www.mengyuanliu.cn/IR.html>

学号: 201400301007	姓名: 刘梦源	班级: 14.4
实验题目: 搜索引擎		
实验学时: 32	实验日期: 2017.5.15	
<p>实验目的:</p> <ol style="list-style-type: none"> Web网页信息抽取 以山东大学新闻网为起点进行网页信息的循环爬取, 保持spider在view.sdu.edu.cn之内。 索引构建 对上一步爬取到的网页进行结构化预处理, 包括分字段解析、分词、构建索引等。 检索排序 对上一步构建的索引库进行查询, 对于给定的查询, 给出检索结果, 明白排序的原理及方法。 检索评价 对于同样的查询, 利用百度的检索结果作为基准, 对自己的检索结果进行评价。 		
<p>硬件环境:</p> <p>PC, 新浪云应用 SAE</p>		
<p>软件环境:</p> <p>Python 3 实现搜索引擎内核部分, html 写交互前端, php 写后台</p>		
<p>实验步骤与内容:</p> <p>(I 部分由我的队友吴成敏同学完成, II 部分由我完成)</p> <p>I 搜索引擎内核部分:</p> <p>我的队友吴成敏同学负责这部分, 主要分成以下几个部分:</p> <p>编写爬取网页, 分词, 利用 bm25 算法建立倒排索引表, 以及将数据分布式处理数据准备部分。</p> <p>1. 爬取网页:</p> <p>使用方法:</p> <p>requests、BeautifulSoup, 用 BeautifulSoup 的 soup.findall(a) 方法来找包含 url 的内容, 之后再根据 link.get('href') 得到每条 url, 对得到的 url 进行进一步的处理。</p> <p>具体过程:</p> <p>分析山东大学新闻网的特点, 会发现在其标题栏上有以下几个 label:</p>  <p>除了第一个山东首页以外, 以及最后两个视点图志, 视频新闻。其他的点击进去都是一系列文章条目:</p>		



所以一开始我在新闻网上进行了如下处理，将这 14 个有用标签单独加入到一个队列中去：

```
def searchurl(start):
    cnt = 0
    queue=deque()
    result=deque()
    visited=set()
    queue.append(start)
    while queue:
        url=queue.popleft()
        visited |= {url}
        try:
            res = requests.get(url)
            res.encoding = 'UTF-8'
            soup = BeautifulSoup(res.text, 'html.parser')
            for link in soup.find_all('a'):
                each_url=link.get('href')
                if 'http://sdu.edu.cn/' == each_url or
'http://202.194.15.43:8080/system/contribute/login.jsp'==each_url:
                    continue
                if 'gjjs.htm' == each_url or 'http://www.view2011.sdu.edu.cn/'==each_url:
                    continue
                if 'http://' not in str(each_url):
                    if each_url!=None:
                        each_url='http://www.view.sdu.edu.cn/'+each_url
                now=each_url
                if now==None:
                    continue
                if "http://www.view.sdu.edu.cn/" not in now:
                    continue
                if 'jpg' in now:#去除一些不需要的链接
                    continue
                if 'jpeg' in now:
                    continue
                if 'img' in now:
                    continue
                if 'swf' in now:
```

```

        continue
    if 'png' in now:
        continue
    if 'doc' in now:
        continue
    if 'docx' in now:
        continue
    if 'pdf' in now:
        continue
    if 'asp' in now:
        continue
    if 'wlp' in now:
        continue
    if now not in visited:
        if 'http://www.view.sdu.edu.cn/info/' in now or 'http://mp.weixin.qq.com/s/' in now or
'http://www.sdrj.sdu.edu.cn/index' in now or now=='http://www.view.sdu.edu.cn/index.htm':
            continue
        elif 'http://www.view.sdu.edu.cn/new/' not in now and now != 'http://www.sdu.edu.cn/' and
now!='http://www.view.sdu.edu.cn/' :
            if now not in visited:
                if '/..' not in now:
                    if 'javascript' not in now:
                        if '@sdu.edu.cn' not in now and now!= (url+'/' ):
                            queue.append(now)
                            result.append(now)
                            visited |= {now}
                            cnt=cnt+1
                            #print(now, cnt)

    if cnt>15:
        break
except (requests.ConnectionError, IndexError, UnicodeEncodeError, TimeoutError) as e:
    mm='sd'
except requests.HTTPError as f:
    print('url 不对.')
return result

```

最后得到一个队列包含上述所说的 14 个有效网址，之后分别处理这些网址，因为是根据源 url 得到所有的队列，一开始的时候进行相应的处理，比如山东要闻的 label，其网址是 <http://www.view.sdu.edu.cn/sdyw.htm>，访问后会发现，该页面是一个一个的新闻稿链接，而共有 414 个网页是属于山大要闻的，而且其网址是具有规律的：

www.view.sdu.edu.cn/sdyw/413.htm

www.view.sdu.edu.cn/sdyw/412.htm

基于上述的规律，所以自动将剩下的山大要闻链接加入队列中：

```
if url=='http://www.view.sdu.edu.cn/sdyw.htm':  
  
    n=0;  
  
    while n<413:  
  
        n=n+1  
  
        nurl='http://www.view.sdu.edu.cn/sdyw/'+str(n)+'.htm'  
  
        queue.append(nurl)
```

其他 label 一样处理。

```
def search(q):  
  
    queue=q  
  
    tem=deque()  
  
    visited=set()  
  
    number=0  
  
    while queue:  
  
        if number>12000:  
  
            break  
  
        url=queue.popleft()  
  
        if url=='http://www.view.sdu.edu.cn/sjld.htm':  
  
            temp='http://www.view.sdu.edu.cn/sjld/1.htm'  
  
            queue.append(temp)  
  
        if url=='http://www.view.sdu.edu.cn/sjdx.htm':  
  
            temp='http://www.view.sdu.edu.cn/sjdx/1.htm'  
  
            queue.append(temp)  
  
        if url=='http://www.view.sdu.edu.cn/xszh.htm':  
  
            n=0  
  
            while n<124:  
  
                n=n+1  
  
                nurl='http://www.view.sdu.edu.cn/xszh/'+str(n)+'.htm'  
  
                queue.append(nurl)  
  
        if url=='http://www.view.sdu.edu.cn/xsyg.htm':  
  
            n=0  
  
            while n<17:  
  
                n=n+1  
  
                nurl='http://www.view.sdu.edu.cn/xsyg/'+str(n)+'.htm'  
  
                queue.append(nurl)  
  
        if url=='http://www.view.sdu.edu.cn/zhxw.htm':  
  
            n=0  
  
            while n<407:  
  
                n=n+1  
  
                nurl='http://www.view.sdu.edu.cn/zhxw/'+str(n)+'.htm'  
  
                queue.append(nurl)  
  
        if url=='http://www.view.sdu.edu.cn/xyxw.htm':  
  
            n=0  
  
            while n<411:  
  
                n=n+1
```

```

nurl='http://www.view.sdu.edu.cn/xyxw/' +str(n)+' .htm'

queue.append(nurl)

if url=='http://www.view.sdu.edu.cn/sdrw.htm':

    n=0

    while n<23:

        n=n+1

        nurl='http://www.view.sdu.edu.cn/sdrw/' +str(n)+' .htm'

        queue.append(nurl)

if url=='http://www.view.sdu.edu.cn/sdrj.htm':

    n=0

    while n<42:

        n=n+1

        nurl='http://www.view.sdu.edu.cn/sdrj/' +str(n)+' .htm'

        queue.append(nurl)

if url=='http://www.view.sdu.edu.cn/gjsy.htm':

    n=0

    while n<17:

        n=n+1

        nurl='http://www.view.sdu.edu.cn/gjsy/' +str(n)+' .htm'

        queue.append(nurl)

if url=='http://www.view.sdu.edu.cn/xlyzl.htm':

    n=0

    while n<74:

        n=n+1

        nurl='http://www.view.sdu.edu.cn/xlyzl/' +str(n)+' .htm'

        queue.append(nurl)

try:

    res = requests.get(url)

    res.encoding = 'UTF-8'

    soup = BeautifulSoup(res.text, 'html.parser')

    for link in soup.find_all('a'):

        each_url = link.get('href')

        if each_url != None :

            if 'http://www.view.sdu.edu.cn' not in each_url:

                each_url='http://www.view.sdu.edu.cn/' +each_url

            if 'http://www.viewnew.sdu.edu.cn' in each_url:

                continue

            if 'info' in each_url:

                if each_url not in visited:

                    visited |={each_url}

                    number = number+1

                    newsurl.append(each_url)

                    tem.append(each_url)

                    file = 'F:\IR/' + str(number) + '.txt'

                    getnews(each_url, file, number)

                    print(each_url,number)

```

```

        if number>800:

            break

    except (requests.ConnectionError, IndexError, UnicodeEncodeError, TimeoutError) as e:

        mm='sd'

    except requests.HTTPError as f:

        print('url 不对。')

with open('F://URL_list.txt','a') as fl:

    while tem:

        fl.write(tem.popleft()+'\n')

```

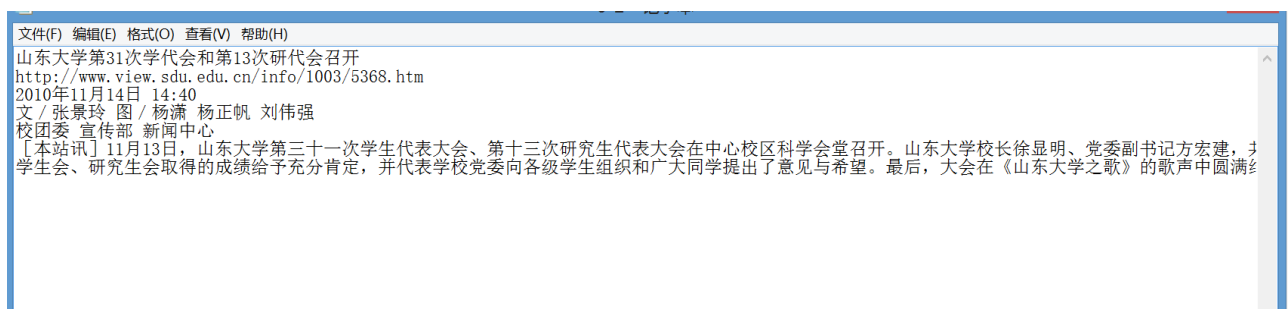
根据上述得到的所有有效 url（山东新闻稿的 url，格式为：http://www.view.sdu.edu.cn/info/数字/数字.htm）。之后对每一个 url 进行解析得到文本内容，得到标题，正文，日期，编辑，按照 url 顺序依次写入到一个 txt 中。

```

with open(filename,'w') as f:
    try:
        res = requests.get(newurl)
        res.encoding = 'utf-8'
        soup = BeautifulSoup(res.text, 'html.parser')
        result['title'] = soup.select('h3')[1].text
        result['zhengwen'] = ''.join([p.text.strip() for p in soup.select('#vsb_content')])
        temp= soup.select('p')[2].text.strip(' ')
        tem=temp.split(' ')
        temp=tem[0]+tem[1]
        result['time']=temp
        if len(str(result['zhengwen']))>=1:
            str1 = str(result['title']+'\n'+result['time']+'\n'+result['zhengwen'])
            f.write(str1)

```

以上处理后得到 12000 个 txt 文档，每个文档内容格式如下：标题，url，日期，作者，宣传中心，正文内容。



2、之后根据上述得到的文本集合，统计词频 tf，tdf

在 tf-idf 的基础上，我使用了 BM25 算法：

BM25 算法，通常用来作搜索相关性平分。一句话概况其主要思想：对 Query 进行语素解析，生成语素 q_i ；然后，对于每个文档 D，计算每个语素 q_i 与 D 的相关性得分，最后，将 q_i 相对于 D 的相关性得分进行加权求和，从而得到 Query 与 D 的相关性得分。

公式为：

$$Score(Q, d) = \sum_i^n IDF(q_i) \cdot \frac{f_i \cdot (k_1 + 1)}{f_i + k_1 \cdot (1 - b + b \cdot \frac{dl}{avgdl})}$$

其中， k_1 ， b 为调节因子，通常根据经验设置，一般 $k_1=2$ ， $b=0.75$ ； f_i 为 q_i 在 d 中的出现频率。 dl 为文档

d 的长度，avgdl 为所有文档的平均长度。

- 首先用 jieba 分词，对所有文档解析得到所有的 term，（去掉了 stop word）共 30347 个词，然后计数每个文档的长度：

```
for i in range(len(date)):
    if i>=6:
        nowdate=nowdate+date[i]
result=jieba.lcut(nowdate,HMM=True)
for i in result:
    if i !=',' and i !='\ ' and i !='“' and i !='”' and i !=']' and i !='[' and i !=';' and i !='们' and i !='+':
        if i !='。' and i !=':' and i.strip() and i !='的' and i !='和' and i !='是' and i !='在' and i !='地' and i !='得':
            if i !='了' and i !='《' and i !='》' and i !='—' and i !='—' and i !='\u3000' and i !='（' and i !='）':
                all = all + 1
                doclen = doclen + 1
                if i not in myset:
                    myset|= {i}
                    words.append(i)
                    num=num+1
```

- 之后根据上述结果进一步计算，读取每一个文本 D，然后使用相同的分词模式，记录对于一个词所出现的文档集：

```
for i in wordlist:#wordlist 存储的是 D 中所有的词。
    if i in word:#word 存储上述已得到的三万个词项。
        visited[i]=0
for w in wordlist:
    if w in word:
        loc=term.index(w)
        td[loc][dindex-2]=td[loc][dindex-2]+1
        if visited[w] == 0:
            idf[w]=idf[w]+1
            visited[w]=1
        if tf[w]==None:
            tf[w]=str(dindex-1)+'\t'
        else:
            tf[w]=tf[w]+str(dindex-1)+'\t'
```

idf 记录了词项在文本集合中出现过的文本数，tf 记录出现在那些文本中，重复也计数。

- 根据上述结果计算真正的 idf:

```
for i in word:
    if idf[i]!=0:
        idf[i] = math.log(docs/idf[i])
```

由 BM25 公式用算法实现得到 score:

```
for i in range(wordnum):
    for j in range(docs):
        temp0=2*(0.25+0.75*(doc[j]/avgdl))
        temp=(td[i][j]*3)/(td[i][j]+temp0)
```

```
score[i][j]=idf[term[i]]*temp
```

以上是整个计算分数的过程。

3、切分索引表

在后来的分布式存储数据中，我们使用了新浪云的 storage 文本存储方式，所以将存储 score 的 txt 分成词数 1500 个的单个 txt:

```
with open ('F://score.txt','r') as f:
    data=f.readlines()
    temp=''
    num=1
    print(len(data))
    for i in range(len(data)):
        if i==(num*1500):
            print(i)
            filename='F://word'+str(num)+' .txt'
            with open(filename,'w') as fl:
                fl.writelines(temp)
            temp=''
            num = num + 1
        temp = temp + data[i]
```

结果如下:

word1	2017/5/23 22:38	文本文档	6,285 KB
word2	2017/5/23 22:38	文本文档	5,981 KB
word3	2017/5/23 22:38	文本文档	4,833 KB
word4	2017/5/23 22:23	文本文档	5,910 KB
word5	2017/5/23 22:23	文本文档	5,896 KB
word6	2017/5/23 22:23	文本文档	5,887 KB
word7	2017/5/23 22:24	文本文档	5,883 KB
word8	2017/5/23 22:24	文本文档	5,880 KB
word9	2017/5/23 22:24	文本文档	5,877 KB
word10	2017/5/23 22:24	文本文档	5,877 KB
word11	2017/5/23 22:24	文本文档	5,875 KB
word12	2017/5/23 22:25	文本文档	5,874 KB
word13	2017/5/23 22:25	文本文档	5,872 KB
word14	2017/5/23 22:25	文本文档	5,872 KB
word15	2017/5/23 22:25	文本文档	5,871 KB
word16	2017/5/23 22:25	文本文档	5,870 KB
word17	2017/5/23 22:25	文本文档	5,870 KB
word18	2017/5/23 22:26	文本文档	5,870 KB
word19	2017/5/23 22:26	文本文档	5,869 KB
word20	2017/5/23 22:26	文本文档	5,869 KB
word21	2017/5/23 22:36	文本文档	1,358 KB

word1 - 记事本	
文档(F) 编辑(E) 格式(O) 查看(V) 帮助(H)	
1.02985 0.039 0.06023 0.05476 0.05959 0.03627 0.0454 0.05363 0.03007 0.0 0.04931 0.05575 0.06166 0.05337 0.03647 0.054	
3.04903 0.06166 0.05388 0.05213 0.04833 0.04731 0.04583 0.06511 0.05659 0.06144 0.04404 0.07332 0.06235 0.03777 0.08783 0.054	
3.04744 0.04164 0.03997 0.04614 0.04672 0.04903 0.04021 0.03833 0.01558 0.05262 0.04468 0.0 0.03394 0.01733 0.07683 0.054	
1 0.05027 0.05034 0.04691 0.0146 0.06223 0.0496 0.05197 0.04691 0.04798 0.04528 0.04059 0.05707 0.0302 0.0385 0.02	
34 0.04896 0.04196 0.03017 0.05449 0.0 0.05158 0.05727 0.04785 0.04546 0.05042 0.02189 0.02765 0.05707 0.02249 0.054	
126 0.03922 0.04536 0.0333 0.0611 0.0499 0.0448 0.05312 0.04646 0.04064 0.05181 0.04778 0.05213 0.03756 0.04882 0.03	
374 0.04104 0.03707 0.07386 0.04189 0.04421 0.04546 0.04659 0.04094 0.04889 0.03927 0.04589 0.04314 0.02847 0.01701 0.044	
3.05423 0.04931 0.04254 0.05836 0.04474 0.02527 0.04903 0.01648 0.03343 0.02837 0.04392 0.05095 0.0 0.05707 0.06281 0.06	
3.03088 0.03995 0.0617 0.05609 0.06104 0.03716 0.0465 0.05493 0.0308 0.0 0.05051 0.05711 0.06316 0.05467 0.03736 0.06	
96 0.03027 0.06316 0.05319 0.0534 0.0495 0.04846 0.04695 0.0667 0.05797 0.06293 0.04511 0.0751 0.06386 0.03859 0.06	
3.0486 0.04266 0.04094 0.04727 0.04786 0.05022 0.04118 0.03927 0.01595 0.0539 0.04577 0.0 0.03476 0.01775 0.0787 0.05	
05149 0.05157 0.04806 0.01495 0.06375 0.05081 0.05324 0.04806 0.04915 0.04638 0.04158 0.05846 0.03094 0.03913 0.02853 0.03	
35778 0.05015 0.04598 0.03091 0.05385 0.0 0.05283 0.03866 0.04901 0.04657 0.05165 0.02342 0.02837 0.05846 0.02384 0.05	
3 0.04441 0.03411 0.06259 0.05111 0.04589 0.05441 0.04759 0.04163 0.05307 0.04894 0.0534 0.03848 0.05 0.03348 0.05	
304 0.03797 0.07578 0.0425 0.04529 0.04657 0.04158 0.04193 0.05008 0.04023 0.04701 0.04419 0.02916 0.01743 0.04806 0.05	
35 0.05051 0.04357 0.05918 0.04583 0.05288 0.05027 0.01688 0.03425 0.05068 0.04499 0.05219 0.0 0.05846 0.06434 0.06	
3.05726 0.0748 0.11552 0.10502 0.1143 0.06957 0.08707 0.10285 0.05767 0.0 0.09458 0.10693 0.11827 0.10236 0.06995 0.11	
3.09999 0.09769 0.09074 0.0879 0.12488 0.10855 0.0 0.08446 0.14063 0.11958 0.07245 0.0 0.09613 0.06684 0.0 0.10	
3.07352 0.05987 0.10093 0.08569 0.0 0.06509 0.03323 0.14735 0.10782 0.079 0.08457 0.08684 0.07823 0.03153 0.07585 0.0	
384 0.07786 0.10946 0.06792 0.07327 0.05343 0.05696 0.10368 0.11511 0.07102 0.09802 0.09309 0.13676 0.09255 0.10836 0.08	
34199 0.03303 0.10946 0.04314 0.10746 0.06796 0.07454 0.07558 0.09656 0.0 0.09613 0.06582 0.09247 0.09177 0.10351 0.10	
34 0.09999 0.07205 0.09363 0.06269 0.0 0.10693 0.08592 0.04496 0.08077 0.0 0.0879 0.11892 0.0967 0.08766 0.09	
3.09908 0.09907 0.11914 0.09295 0.06712 0.0 0.097 0.0819 0.09164 0.10928 0.0837 0.10172 0.09938 0.0967 0.0 0.06	
3.10711 0.09514 0.1216 0.09907 0.07197 0.09758 0.09817 0.09295 0.07657 0.09151 0.08899 0.0838 0.05761 0.09577 0.06382 0.0	
3.23426 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	
3.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	
3 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	
3.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	
2.89233 1.04703 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	
3.0 0.0 0.0 1.72008 2.65243 0.0 0.0 0.0 2.84922 0.0 0.0 0.0 1.48346 0.0 0.0	
3.0 0.0 0.0 0.70405 1.67082 0.0 0.0 1.2596 1.28826 0.0 0.0 0.0 0.81085 0.0 0.0	
0 1.33575 0.0 0.0 2.56636 0.0 0.0 2.15444 0.0 0.7808 0.0 0.0 0.0 0.0 0.0	
0 0.0 0.0 1.34567 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	

II. 人机交互与服务器架构部分:

(1) 综述

为得到更广泛的测试，我们将引擎最高层的交互程序封装成 web application 的形式，这就需要在云端有服务器支撑程序架构。

众所周知，搜索引擎交互简单，数据库中主要存储的是我们的文档与索引表，所以数据大而不杂，J2EE 等传统 web 架构显然繁琐冗余，也是不必要的。所以，我们选取最简单的，也是现如今最流行的 html+php 的 web 开发语言，服务器部署在新浪云平台。

云应用取名为 Sdubaidu，截图如下。

语言版本	安全	应用信息	状态	近30天云豆消耗(不包括今天)	操作
 5.3		 http://sduabaidu.applinzi.com	正常	1165.88	  

部署在云端的 web application 与本地程序在本项目中有明显的不同，主要有以下几个原因：

- ◆ 搜索文档是庞大的，倒排索引表也是庞大的，本地的内存空间往往支持程序较为容易，然而云端申请一个与 PC 配置相同的服务器需要大量的资金支持，如何在小内存空间内完成高性能计算，需要空间的优化。
- ◆ 时间上来讲，虽然交互层需要完成的事件仅仅是计算总得分与返回文档结果排序的 html 任务，但云端的计算资源同意难以保证，为保证较快的完成查询任务，使用户获得较好的交互体验与回馈感，需要在时间上优化。
- ◆ 数据存储上来讲，数据库并不能对我们的数据提供百分之百的支持，例如倒排索引表，按行（词条数）与按列（文档）都不具有数据库的性质，而且条目太大，写几万条 sql 语句写进数据库显然不现实，这需要开拓数据库以外的新的存储方式。
- ◆ 云端因网络原因不稳定线性较多，需要更繁琐的异常处理。

（2）交互层

1. 对 html 文本框输入的 query，进行分词，php 缺少较好的分词工具，采用在线 API 语言云，发送 get 请求后返回分词好的结果。代码如下：

```
$word = $_POST["searchInput"];
// echo $word;
$url = http://api.ltp-cloud.com/analysis/?api_key=S12168E1h2IMEFX7mHUZ8jxHFmJREafEVH7B0erc@text='. $word.' @pattern=ws@format=plain';
$html = file_get_contents($url);
// echo $html;
```

2. 对分好词的每一个词条，在文档词条的数据库中进行检索，如果 query 的词条在数据库中出现，返回词条的 ID。代码如下：

```
for($i = 0; $i < $count; $i++) {
    $w = $strs[$i];
    //echo $w;
    $query = "select id from words where words_content = '$w' ";
    $result = mysql_query($query);

    if(mysql_num_rows($result) == 0) {
    }
    else {
        while($row_result = mysql_fetch_array($result)) {
            // echo $row_result['id'];
            array_push($ids, $row_result['id']);
        }
    }
}
```

3. 拿到 ID 之后，在索引表中取出 ID 对应的行，然后把各行求和，计算 query 的总得分，代码如下。

```

for($j = 0; $j < count($ids); $j++) {
    $whichfile = floor(($ids[$j] - 1) / 1500) + 1;
    $filerow = ($ids[$j] - 1) % 1500 + 1;

    $wordfilename = $wordfilename.$whichfile.".txt";
    $handle = @fopen($wordfilename, "r");
    $line_number = 1;

    if ($handle) {
        while (!feof($handle)) {
            $buffer = fgets($handle, 8000);
            if($line_number == $filerow) {
                //echo "find it";
                if($ids_index > count($ids)) break;

                //echo $buffer;
                $nums=str_replace("\t", "_", $buffer);
                $nums=explode("_", $nums);
                //print_r($nums);

                for($j = 0; $j < 1000; $j++) {
                    $sum[$j] = $sum[$j] + $nums[$j];
                }
                break;
            }
            $line_number++;
            //echo $line_number." ";
        }
    }
}
}

```

4.对总得分排序，按照得分降序返回文档，反传给 HTML 前端。代码如下：

```

for($i = 0; $i < 1000; $i++) {
    if($sum[$i] > 0) {
        array_push($searchContent_ids, $i + 1);
        array_push($searchContent_scores, $sum[$i]);
    }
}
//echo "searchContent_ids:".count($searchContent_ids);

for($i = 0; $i < count($searchContent_ids); $i++) {
    for($j = 0; $j < $i; $j++) {
        if($searchContent_scores[$i] > $searchContent_scores[$j]) {
            $t = $searchContent_scores[$i];
            $searchContent_scores[$i] = $searchContent_scores[$j];
            $searchContent_scores[$j] = $t;

            $t = $searchContent_ids[$i];
            $searchContent_ids[$i] = $searchContent_ids[$j];
            $searchContent_ids[$j] = $t;
        }
    }
}

```

(3) 优化：

- 内存方面的优化：

倒排索引表是巨大的，如果直接加载到内存会使服务器崩溃，而索引表又难以存储在数据库中，我们采取分布式的方法，如下图所示，我们讲巨大的索引表采用分布式存储在若干 **storage** 单元中，这样每次只需要加载进一个索引表的子文件至内存就可以了。

那么如何找到子文件呢，很简单，我们的存储是按照顺序和固定大小，只要查到了 **word** 的 ID 索引号，我们便可计算出它在的 **storage** 中的位置。从而在节省内存空间的同时又不损失时间。

<input type="checkbox"/>	名称	大小	类型	修改时间	操作
<input type="checkbox"/>	score.txt	116.7M	文件	2017-05-21 18:24:06	删除
<input type="checkbox"/>	word1.txt	6.1M	文件	2017-05-23 22:28:33	删除
<input type="checkbox"/>	word10.txt	5.7M	文件	2017-05-23 22:31:59	删除
<input type="checkbox"/>	word11.txt	5.7M	文件	2017-05-23 22:32:03	删除
<input type="checkbox"/>	word12.txt	5.7M	文件	2017-05-23 22:32:26	删除
<input type="checkbox"/>	word13.txt	5.7M	文件	2017-05-23 22:32:39	删除
<input type="checkbox"/>	word14.txt	5.7M	文件	2017-05-23 22:32:40	删除
<input type="checkbox"/>	word15.txt	5.7M	文件	2017-05-23 22:33:04	删除
<input type="checkbox"/>	word16.txt	5.7M	文件	2017-05-23 22:33:15	删除
<input type="checkbox"/>	word17.txt	5.7M	文件	2017-05-23 22:33:18	删除
<input type="checkbox"/>	word18.txt	5.7M	文件	2017-05-23 22:33:44	删除
<input type="checkbox"/>	word19.txt	5.7M	文件	2017-05-23 22:33:49	删除
<input type="checkbox"/>	word2.txt	5.8M	文件	2017-05-23 22:28:29	删除
<input type="checkbox"/>	word20.txt	5.7M	文件	2017-05-23 22:33:51	删除
<input type="checkbox"/>	word21.txt	1.3M	文件	2017-05-23 22:35:23	删除
<input type="checkbox"/>	word3.txt	5.8M	文件	2017-05-23 22:30:41	删除
<input type="checkbox"/>	word4.txt	5.8M	文件	2017-05-23 22:30:42	删除
<input type="checkbox"/>	word5.txt	5.8M	文件	2017-05-23 22:30:45	删除
<input type="checkbox"/>	word6.txt	5.7M	文件	2017-05-23 22:31:15	删除

● 时间方面的优化:

第一, 倒排索引表一定要 `getline()` 方法, 也就是按照行来读, 虽然 PHP 中没有按行读的方便, 但就算程序员手动实现本方法, 也很有必要要这样做, 事实证明, 这会使程序快很多。

第二, 使用了小技巧, 拿到 `query` 中的 `word` 的 ID 号之后先排序。如果不排序, 搜索 `query` 中的词条的时间复杂度是 $m \cdot O(n)$, 其中, m 是 `query` 中的 `word` 数目, n 是索引表中的词条数目; 而如果先排序的话, 时间复杂度缩减为 $O(n)$ 。

● 存储结构的优化:

采用数据库和 `storage` 两种存储结构。

(a)其中数据库存储具有数据库结构的数据, 文档与词表, 如下图所示

<input type="checkbox"/>	表	操作	记录数	类型	整理	大小	字节
<input type="checkbox"/>	documents	   	1,000	InnoDB	utf8_general_ci	2.5 MB	-
<input type="checkbox"/>	words	   	~30,851	InnoDB	utf8_general_ci	1.5 MB	-
	2 个表	总计	~31,851	InnoDB	utf8_general_ci	4.0 MB	0 字节

(b)`Storage` 用于存储云端海量大数据文件, 直接保留 `txt` 数据, 按照 `url` 访问。在本项目中, `storage` 负责存储不具有数据库属性的倒排索引表。

(4) 项目截图展示:



山东大学站内搜索

sdu.baidu.applinzi.com/baidu/web/index.php

该页面的提供者尚未完成 实名认证 您的访问可能存在风险



山东大学
SHANDONG UNIVERSITY

丁肇中报告

丁肇中

丁肇中同志

丁肇中报告

丁肇中参观我校

丁肇中是个好人

搜索

高级搜索

Copyright © 2017 by 刘梦源 吴成敬 林俊宇 All rights reserved.

山东大学站内搜索

sdu.baidu.applinzi.com/baidu/web/result.php



山东大学
SHANDONG UNIVERSITY

搜索

高级搜索

找到约 8 条结果(用时0.36秒), 共约1页

按时间排序

按相关度排序

897

丁肇中教授参观我校博物馆

- 设计分类: 山大新闻 相关度得分: 11.83434

[本站讯] 6月29日下午, 在展涛校长的陪同下, 丁肇中教授参观了我校博物馆精品文物展和西校区校园。山大博物馆精品文物展展出的主要是出土文物。在博物馆馆长于海广教授的讲解中, 丁肇中教授兴趣很浓, 并提出了许多问题, 像“饮酒容器产生的年代”、“远古时代人类活动范围和文化传播半径”等。在于海广教授讲到出土文物高柄杯时, 丁肇中教授摘掉眼镜, 仔细地观看并叮嘱助手多拍几张照片。在博物馆休息室, 丁肇中教授愉快地在博

作者: 葛亮 单位: 宣传部新闻中心 发布时间: 2004年06月30日 14:42

624

丁肇中教授与山大科研人员座谈AMS02项目

- 设计分类: 山大新闻 相关度得分: 10.22975

[本站讯] 3月22日上午9点, 诺贝尔物理学奖获得者丁肇中教授在山东大学南校区空间热科学研究中心, 与山东大学AMS项目组的研究人员进行了工作会谈, 听取了该项目的工作进展汇报, 参观并察看了实验室及AMS项目所取得的成果。之前, 丁肇中教授曾分别于2004年6月和2005年6月两次访问空间热科学研究中心, 考察山东大学AMS项目进展情况。在山东大学空间热科学研究中心会议室, 校长展涛主持了座谈会。Joseph

作者: 文/苏虹燕 图/韩军 单位: 宣传部新闻中心 发布时间: 2006年03月22日 14:40

767

中央卫视、山东卫视即将播出对刘建亚教授、丁肇中教授的专访

- 设计分类: 山大新闻 相关度得分: 5.85323

[本站讯] 中央电视台10套“科教频道”《走近科学》栏目将于今晚8:30播出对我校数学与系统科学学院刘建亚教授的专访。刘建亚教授为我校“长江学者奖励计划”特聘教授、博士生导师, 教育部首届“高等学校教学名师奖”获得者。另据悉, 山东卫视将于21日(星期天)中午12点播出对诺贝尔物理学奖获得者、著名物理学家丁肇中教授的专访。丁肇中教授于2月26日~29日对我校进行了访问, 在社会上引起了很大的反响。]

作者: 单位: 发布时间: 2004年03月19日 10:40

(5) 拓展模块:

热门搜索:

本模块很简单, 只需要在数据库中新加表, 记录搜索的 query, 每一个 query 有属性 times, 记录本 query 被搜索的次数, 热门搜索, 即 times 最大的 quert。

热门搜索:

青岛校区 刘梦源是个好人 丁肇中 青岛 评优 放假通知 建设世界一流大学

推荐系统:

本项目推荐文档不局限于文档内, 在百度中根据搜索 query 的 pagerank 排序, 找到被搜索到的 url 指向在百度搜索中相关联最大的网站, 作为使用者的推荐 url, 事实证明我们的推荐系统所推荐的网站效果较好, 都是具有较大可能性。

推荐系统：山大要闻—山东大学新闻网 青春山大 山东大学 学生在线 精彩无限

● 相关搜索：

记录 query 的数据库中与本次搜索 query 相关联的记录，我们仅限于文本相关，而不考虑语义。

丁肇中

搜索

高级搜索

丁肇中

丁肇中同志

丁肇中报告

丁肇中参观我校

丁肇中是个好人

结论分析与体会：

我们的项目是两个人一起完成，两个人分工明确，人员梯度合理。我的队友吴成敏同学完成搜索引擎内核，我负责交互程序与服务器架构，最终将内核封装为完整的 web-application。程序完全自己实现，除分词使用 jieba 分词与语言云在线分词之外，没有用到任何开源工具，最终实现了 lucene 的 BM25 算法，Nutch 的分布式存储的思想。可谓收获不小。

仅从我完成的工作上来说，我的收获很大，我认为我们的项目按数据量来说已经可以算得上是中型以上的项目级别了，又部署在云端，所以为了可以让 application work 起来，完成的优化工作不少，而还可以做的优化工作仍然有很多。

程序现在还在跑，如下



实时分析



看似伟大的搜索引擎已经被我实现了较为理想的 **Demo**，希望可以把工作继续和我的队友一起做下去。今后可探索的方面有两点：一是内核的评分算法，需要考虑 **query** 中词与词的相关性，而不仅仅是把每个词的 **score** 求和的方法；二是大数据优化，还有很多工作可以做，利用最少的计算资源取得最好的计算效果，完成最多的计算任务。

希望可以和我的队友继续把这个 **demo** 做下去。