南开教师搜索系统

计算机科学与技术专业 1711436 皮春莹

2019 年 11 月 26 日

目录

1 实验内容

本次实验分为以下四个子系统，四个子系统之间的关系如图 1所示。

1) 网页抓取：南开各单位教师网页

2) 建立文本索引：网页及其锚文本，按域索引

3) 进行链接分析：PageRank 算法

4) 编写前端页面，提供查询服务：参考百度，支持，但不局限于：

- site、filetype 等
- 短语、通配等
- 查询日志、网页快照等
- 搜索引擎优化

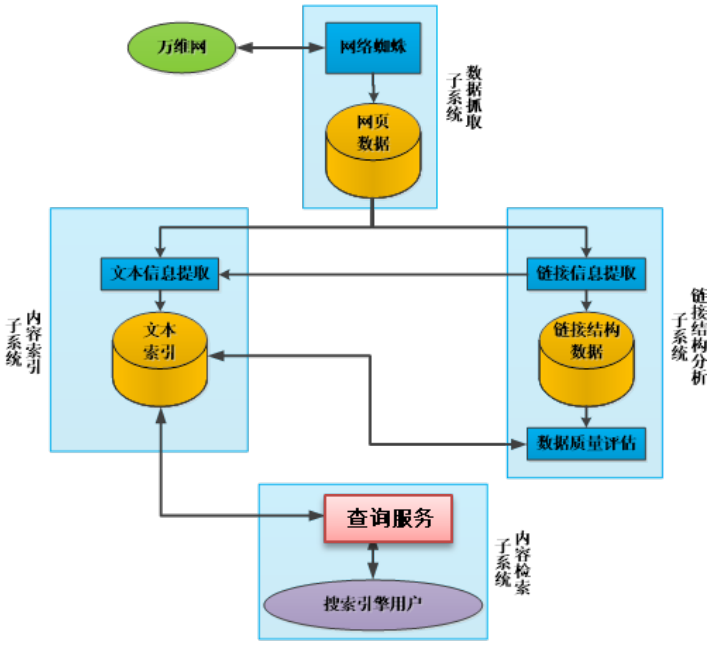

图 1: 实验内容

2 实验环境

1) 语言：Python 3.6

2) 编译器：JetBrains PyCharm Community Edition 2019.1.3

3) 爬虫系统主要用到的插件：

- beautifulSoup：用来解析 HTML 文本，可以获取爬到的 HTML 文本中的各个项的属性。
- selenium：可以模拟浏览器获取异步数据或者执行点击操作。有一些网站的部分内容是用 JQuery 或者 Ajax 异步加载的，用简单的 Get 方式并不能获取到所有想要抓到的内容。并且爬取过程中需要通过搜索递归的方式，需要实现模拟浏览器的点击操作。

4) 索引系统主要用到的插件：

- whoosh：whoosh 是用于索引文本，然后搜索索引的类和函数的库，可以使用 whoosh 为内容开发自定义搜索引擎。
- jieba：中文分词包，建立索引的过程中需要对获取的内容进行分词，但 whoosh 默认的分词对中文不太友好，所以用到 jieba 自定义了中文分词方式。

5) PageRank 主要用到的插件：

- networkx：用于构建和操作复杂的图结构，提供分析图的算法。

6) 前端页面主要用到的插件：

- logging：logging 模块定义的函数和类为应用程序和库的开发实现了一个灵活的事件日志系统，在这里用于记录用户查询日志。
- QueryParser：查询解析器，也是 whoosh 的一部分，作用是将用户提交的查询字符串转换为查询对象。
- flask 框架：flask 是一个使用 Python 编写的轻量级 Web 应用框架。基于 Werkzeug WSGI 工具箱和 Jinja2 模板引擎。

3 创新点

在前三个子系统中都是很常规的操作，在前端页面上下了一番工夫，主要增加了以下功能：

1) 按域查询：可以选择搜索的内容是链接、文件、姓名、邮件、学院、研究方向、工作职称等等。

2) 前/后缀查询：可以通过输入前几个或后几个字，来搜索完整的内容。

3) 多个短语查询：可以搜索多个用空格分开的短语。

4) 逻辑查询：支持短语间的 OR、AND、NOT 运算。

5) 对上一次检索结果进行缓存，提高响应速度。如果用户只是进行换页操作，没有重新搜索，那么可以直接从缓存中取数据。

6) 文件查看：可以查看与系统有关的文件，比如前端页面的 logo。

7) 语音输入：通常设备都有可使用的默认语音识别系统，在这里，基于 JavaScript 利用 Web Speech API 的控制接口 SpeechRecognition 实现语音输入。

8) 输入内容推荐：根据之前的输入内容，在用户输入时给出建议。

9) 查询日志：记录了用户的查询日志，便于做进一步拓展，分析用户的查询行为。

10) 换页：在展示搜索结果时，我将数量上限设为了 20 个，并对内容按照 PageRank 算法得分从高到低，进行分页展示。

11) 界面美化，logo 设计：搜索系统名为“众里寻 Ta”, 取意于“众里寻他千百度”，首页图片上“众里”两个字，是由老师们的姓名为字符集绘制成的云文字。给每个页面加上了低面背景，增加浏览时的舒适度。

4 实现思路

4.1 网页抓取子系统

在网页抓取子系统中，需要爬取南开各单位教师网页，收集教师信息，主要包括：姓名、职称、邮箱、链接、研究方向、学院、个人简介等。由 scrapyData.py 实现。使用 selenium 中的 webdriver 模拟火狐浏览器执行点击操作，根据指定的 URL 获取页面内容，存放在 browser 变量中，使用 BeautifulSoup 解析 HTML 文本，剖析器为 html.parser，根据网页的结构，找到需要信息的所在位置，调用 select() 函数取出信息。也可以使用 browser.find_element_by_class_name() 方法，取出信息。在爬取信息中有几个特殊处理的地方：

1) 不同的学院老师的信息类别不同，有的信息可能并不展示，所以需要使用 try-catch 结构，将没有找到的项置为空。

2) 由于个人简介的信息太长，并且不是关键信息，没必要全部存储，所以只取了前 100 个字。

3) 为了实现 PageRank 算法，在爬取信息时，需要链接及其对应的锚文本，但是在同一个页面中，教师个人页面的链接格式不统一，如图 2所示。所以在存储之前要提前做判断，将不完整的路径补全之后再存储。

图 2: 个人网页链接

对于爬取到的数据，最初我是想存在 MySQL 中的，一番尝试之后，发现关系类数据库不适用于爬虫数据存储，需要使用非关系类数据库 MongoDB，但是配置了很久环境总是出错，最终妥协，将数据以 json 格式分块存入了磁盘。爬取到的数据主要包括：TeacherInfo 文件夹下按照学院存储的教师信息，以 TeacherInfo/computer.json 为例，它记录了计算机学院和网络空间安全学院的教师信息，如图 3所示，整个文件是一个 list 数据结构，list 中每一项是一个 dict，key 为类别，value 为具体内容。cate、collage、depart、direct、href、intro、name 分别表示教师的职称、学院、专业、研究方向、链接、个人简介、姓名，计算机学院官网上没有专门列出教师的邮箱信息，所以 computer.json 的每个字典中没有 e_mail 一项，但是在其他的学院教师信息中有记录。PRInfo 文件夹下的 pageGraph.json 记录了指出链接的网页、指向的链接及其对应的锚文本，以图 4的第一项为例，表示了在网页地址为 `http://cc.nankai.edu.cn/js/list.htm`，有一处锚文本“高裴裴”（\u9ad8\u88f4\u88f4），指向了地址为 `http://cc.nankai.edu.cn/2019/0619/c13621a179413/page.htm` 的网页。

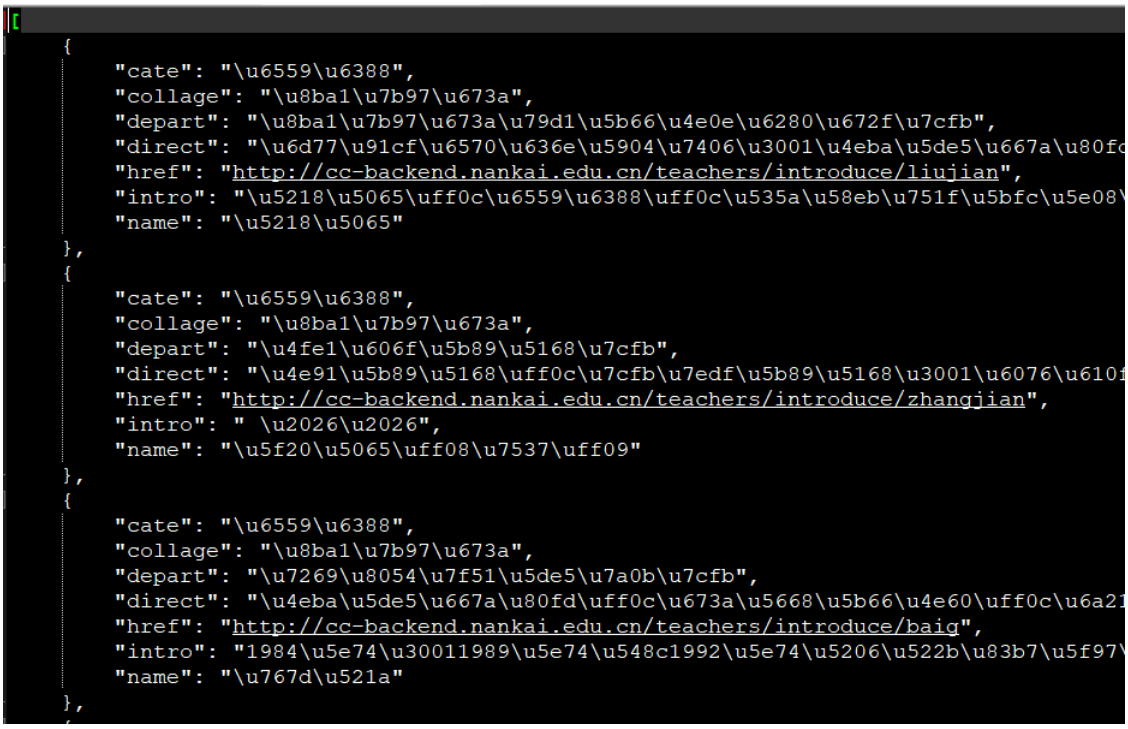

图 3: TeacherInfo/computer.json

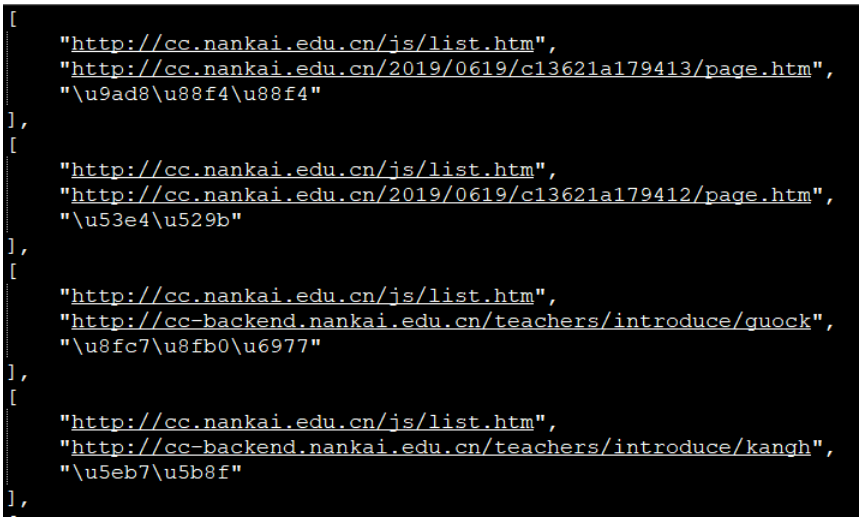

图 4: PRInfo/pageGraph.json

4.2 内容索引子系统

在内容索引子系统中，需要对文本进行分词，然后按域存储信息。借助 whoosh 和 jieba 建立索引很方便，整个过程在 buildIndex.py 中实现。首先需要用 Schema 创建存储的模式，其中参数 stored 为 True 表示能够被检索，参数 analyzer 需要设置成 jieba 中的 ChineseAnalyzer()，因为 whoosh 自带的分词对中文不太友好。之后调用 create_in(indexdir, schema) 函数，创建索引。再按照 schema 定义信息，调用 writer() 对象的 add_document() 函数，增加需要建立索引的文档。最后将写入的内容进行 commit。至此，索引创建成功，存储在 indexdir 文件夹下 (图 5)。

图 5: indexdir 文件夹

4.3 PageRank 算法

在 PRInfo/pageGraph.json 文件中已经收集了关于网页地址及锚文本的数据，这一部分的主要任务是用 PageRank 算法给这些 URL 赋权，由 pageRank.py 实现。PageRank 是 Google 公司所使用的对其搜索引擎搜索结果中的网页进行排名的一种算法，其本质上是一种以网页之间的超链接个数和质量作为主要因素粗略地分析网页的重要性的算法。网页间的指向关系可以用数据结构中的“图”来表示。对于一个页面 A，它的 PR 值为：

$$PR(A) = (1-d) + d(PR(T_i)/C(T_i) + ... + PR(T_n)/C(T_n)). \quad (1)$$

$PR(A)$ 是页面 A 的 PR 值。$PR(T_i)$ 是页面 Ti 的 PR 值，在这里，页面 T_i 是指向 A 的所有页面中的某个页面。$C(T_i)$ 是页面 T_i 的出度，也就是 T_i 指向其他页面的边的个数。d 为阻尼系数，其意义是，在任意时刻，用户到达某页面后并继续向后浏览的概率，该数值是根据上网者使用浏览器书签的平均频率估算而得，通常 $d = 0.85$。迭代计算各个页面的 PR 值，当上次迭代结果与本次迭代结果小于某个误差，结束程序运行。具体在 pageRank.py 中，先调用 networkx 的 DiGraph() 方法初始化一个空的图 graph，读取 PRInfo/pageGraph.json 中的数据，调用 graph.add_edge() 函数给图中加入顶点和边。顺便可以将锚文本记录在 PRInfo/anchor.json 中，便于之后做高亮处理。之后调用 pagerank(graph, alpha=0.85) 函数，得到各个页面的 PR 值，将计算结果存储在 PRInfo/pageRank.json 中，如图 6所示。本次实验中用到的网页间的指向关系都比较简单，通常都是由官网首页的“师资队伍”页

面，指向每位教师的个人页面，所以算出来的各个网页的 PR 值基本相同。图 7是顺便收集的锚文本的信息（PRInfo/anchor.json）。


```
"http://cc-backend.nankai.edu.cn/teachers/introduce/baig": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/caigq": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/caixiangrui": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/chengmm": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/guock": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/jiacf": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/kangh": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/lim": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/liqc": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/lit": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/liujian": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/liuxg": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/lixj": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/renmm": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/shaoxl": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/shenbq": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/songcy": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/wangg": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/wangz": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/weijm": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/xinyw": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/xujd": 0.004178027980632942,
"http://cc-backend.nankai.edu.cn/teachers/introduce/xujun123": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/xuxia": 0.004159187487677625,
"http://cc-backend.nankai.edu.cn/teachers/introduce/yangzl": 0.004178027980632942,
```


图 6: PRInfo/pageRank.json


```
"\u53e4\u529b": [
    "http://cc.nankai.edu.cn/2019/0619/c13621a179412/page.htm"
],
"\u53f6\u6c11": [
    "https://history.nankai.edu.cn//2019/0910/c16091a201740/page.htm"
],
"\u5434\u82f1": [
    "http://cc.nankai.edu.cn/2019/0619/c13620a179380/page.htm"
],
"\u5468\u73af": [
    "http://cc.nankai.edu.cn/2019/0619/c13620a179370/page.htm"
],
"\u590f\u708e": [
    "https://history.nankai.edu.cn//2019/0910/c16087a201690/page.htm"
],
"\u59dc\u80dc\u5229": [
    "https://history.nankai.edu.cn//2019/0910/c16088a201712/page.htm"
],
"\u5b59\u536b\u56fd": [
    "https://history.nankai.edu.cn//2019/0910/c16088a201709/page.htm"
],
```


图 7: PRInfo/anchor.json

4.4 内容检索子系统

在内容检索子系统中，设计实现按域搜索、短语查询、逻辑查询的查询格式，支持语音输入，分页展示搜索到的教师信息。由 webQuery.py 文件实现。

4.4.1 前端页面设计

使用 flask 框架，根据上面提到的前端页面的功能，主要设计了两个页面：首页 hello.html，如图 8所示，以及查询结果页面 search_result.html，如图 9所示。首页 hello.html 的正上方是搜索系统的标志“众里寻 Ta”，“众里”两个字，是由老师们的姓名为字符集绘制成的云文字。标志下方是搜索框，下面的下拉按钮中的内容是搜索域，有 website、filetype、name、direction、collage 等选项可供选择，左边是输入框，输入的查询内容可以是单个短语，也可以是前缀、后缀或多个空格隔开的短语。输入框右端有一个话筒的图标，点击图标时，可以进行语音输入，最右边是提交表单的按钮。下边还有一个“Logic Quiry”的 checkbox，若勾选，表示查询内容中含有“OR”等关键字，进行逻辑查询。

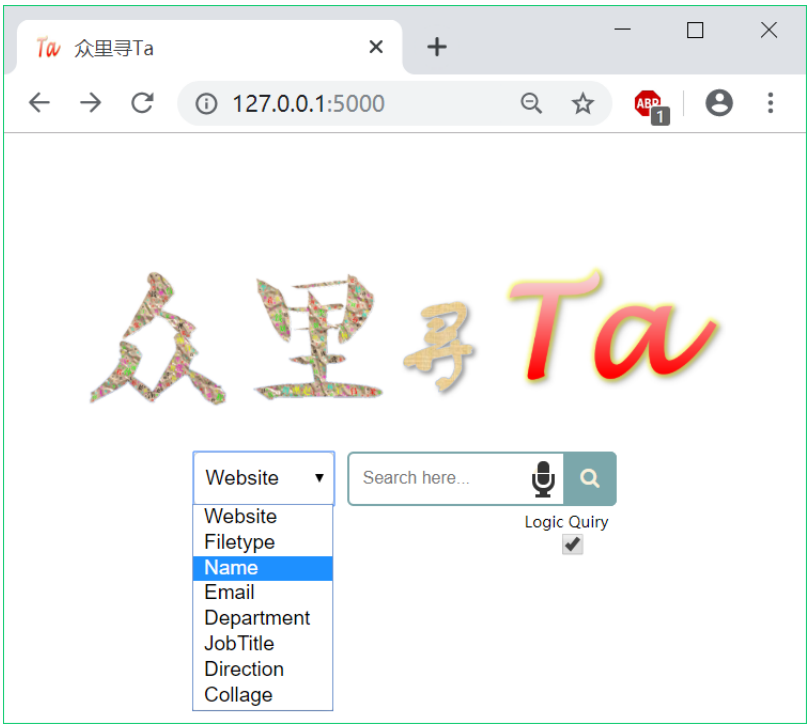

图 8: hello.html

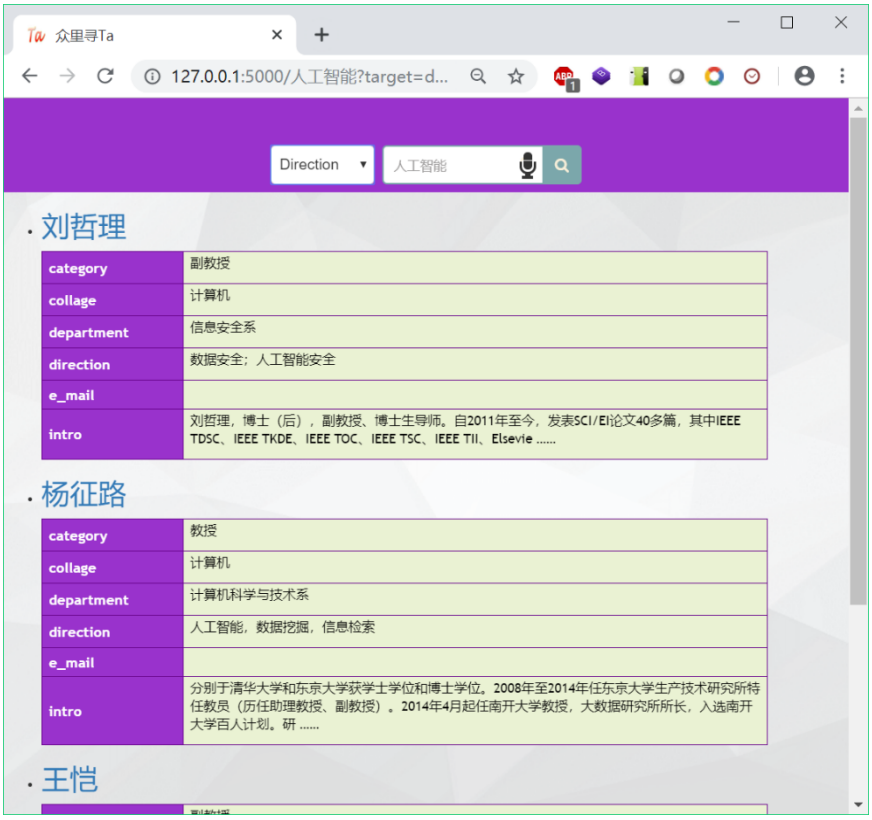

图 9: search_result.html

查询结果页面 search.html 加上了低面背景，增加浏览时的舒适度。它的首部仍然是搜索框，以便用户再次搜索。中间是搜索结果的展示，按照 PageRank 算法得分从高到低排列，每一项中包括教师的姓名、职称、学院、专业、研究方向、邮件、个人简介，其中教师姓名链接到教师的个人网页，点击后会进行跳转。若搜索到的结果较多时，会进行分页，每页展示三位教师，如图 10所示。

4.4.2 检索功能逻辑

搜索功能主要由 webQuery.py 中的 search() 和 search_result(sent) 函数完成。导入 flask 类之后，用 app = Flask(__name__) 创建 Flask 类的实例。使用 route() 修饰器注明通过什么样的 url 可以访问定义的函数，同时在函数中返回要显示在浏览器中的信息，这里就相当于是 MVC 中的 Contoller。当请求的地址符合路由规则时，就会进入该函数。这里相当于 MVC 的 Model 层。可以在里面获取请求的 request 对象，返回的内容为 response，

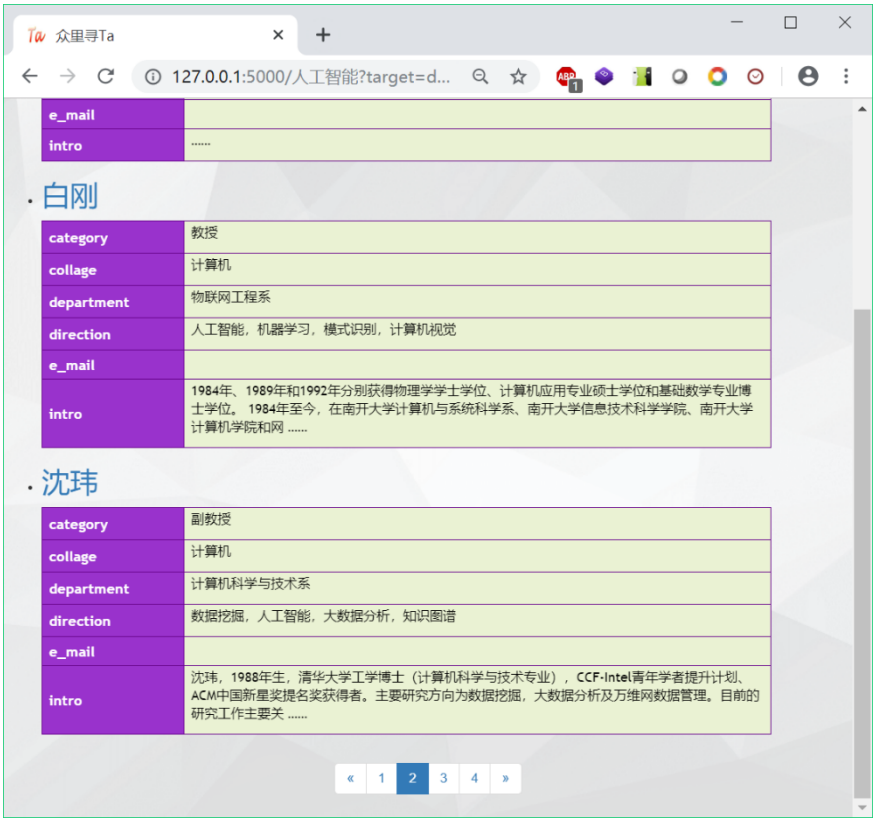

图 10: 分页展示

或者调用 render_template() 传递参数并跳转到另一个页面。最后调用 run() 方法，运行 flask web 应用程序。

search() 的 route 为‘\’，在浏览器输入“127.0.0.1:5000”，回车，会跳转到图 8所示的系统首页 hello.html。用户点击搜索按钮之后，网页会发送 POST 请求，将表单提交到服务器，search() 函数获取请求的 request 对象，从表达中取出查询的类别（target）、查询的内容（sent）以及是否进行逻辑查询（isLogic），然后调用 redirect(url_for('search_result', sent=sent, target=target, isLogic=isLogic, page=1)) 跳转到 search_result(sent) 函数对应的 URL，其中，url_for() 函数接受视图函数的名字（字符串形式）作为参数，返回视图函数对应的 url，对于有传参的视图函数，将函数名字符串作为第一个参数，将参数以 key=value 的形式写在后面。redirect() 函数的功能就是跳转到指定的 url。

search_result(sent) 的 route 是‘/<sent>’，它的含义是“<”和“>”中间的内容会被传来的参数替换，组成 search_result(sent) 函数的路由。在 search_result(sent) 函数中，

获取请求的 request 对象，获得查询的类别（target）、查询的内容（sent）以及是否进行逻辑查询（isLogic）。接下来就进入最主要的检索阶段：

首先使用 whoosh 工具包中的 index.open_dir("indexdir")，加载索引，记为 myindex。其次需要构造查询器，记为 qp，这里调用 whoosh 工具包中的 QueryParser(target, schema=myindex.schema)，参数 target 是前面获取到的查询的域，schema 是在 buildIndex.py 中定义过的索引的模式。然后构造查询向量，记为 q，如果不是逻辑查询，这里就这需要简单地使用 qp.parse(sent)，sent 是前面获取到的查询的内容。如果是逻辑查询，这里需要先用关键词“OR”进行拆分，然后将拆分出来的短语调用 Or([Term(target, terms[0]), Term(target, terms[1]),⋯]) 函数，组合成查询向量。这里需要说明的是，逻辑查询一般有“OR”、“AND”、“NOT”三种，但是“AND”的情况可以用空格隔开的短语来表示，并且考虑到教师搜索系统的实际使用，“NOT”运算并没有什么用处，所以这里的逻辑查询只实现了“OR”运算。最后使用 myindex.searcher() 的 search (q, limit=20) 函数，返回查询结果，将结果存储在 buffer 变量中，buffer 中是查询结果在索引中的位置。limit 是返回的结果数上限，默认为 10，这里我设置成了 20 个。

到此为止，基本的检索过程已经完成，search_result(sent) 函数的关键部分流程如图11所示。但还需要做一些完善和优化：

buffer 中存储的仅仅是在索引中的位置信息，并不是真正要展示的结果，所以需要调用 buffer[i].fields() 取出每一个教师信息，这里得到的是一个字典结构，再将这些信息存入一个数组中，记为 details，details 中的内容才是最重要展示的搜索结果。

whoosh 中的 search() 函数默认使用的排序不是 PageRank 算法，所以要对 details 重新排序，这里定义了 resort() 函数来完成这一任务。在 resort() 函数中，加载 PRInfo/pageGraph.json 文件，得到各个网页的 PR 值的信息，结构为 list，list 中每一项是一个 dict。list 原本是有排序函数的，在这里需要进行重载，让每一项的大小比较转换为他们各自的按照 PR 值比较，由 func() 函数实现，调用 sorted(buf, func) 得到按照 PageRank 排序的结果。

为了便于浏览，对结果进行了分页展示，在页面下方增加了换页的按钮。按钮上增加了 URL，链接到 search_result 函数，参数 sent、target 都不做改变，只是按照用户选择，对 page 进行改变。

增加换页功能后，每次换页都会触发 search_result 函数，返回搜索的结果，但实际上换页的操作时不需要再次进行搜索的，只需要对上一次检索结果进行缓存，提高响应速度。如果用户只是进行换页操作，没有重新搜索，那么可以直接从缓存中取数据。在 search_result() 中判断 page 是否等于 1，就可以知道用户是不是查询了新的内容。

另外，记录查询日志，便于分析用户的查询行为，做进一步拓展。这个借助 logging 工具包可以很容易地实现，只需要调用 logging.basicConfig() 函数指出日志记录的级别、文

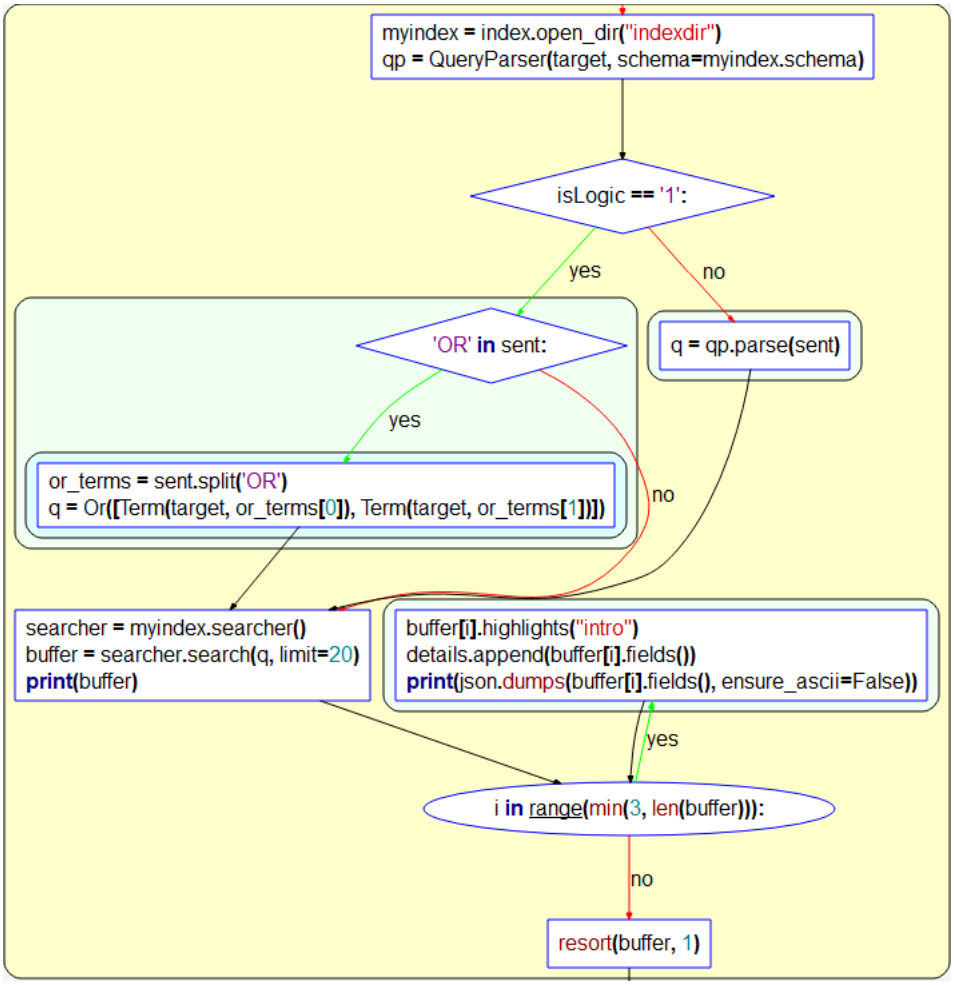

图 11: 检索过程流程图

件名、写模式和格式即可。记录的日志文件在 new.log 中，内容如图 12所示。

5 代码实现

5.1 网页抓取相关代码

在网页抓取子系统中，使用了 selenium 和 BeautifulSoup 工具包，每各学院网站的结构都不尽相同，具体的代码也会稍有不同，所以我分学院写了爬虫的函数，这些函数的原理是相同的。这里只粘贴了爬取金融学院教师信息时用到的两个主要函数。首先在 finance()

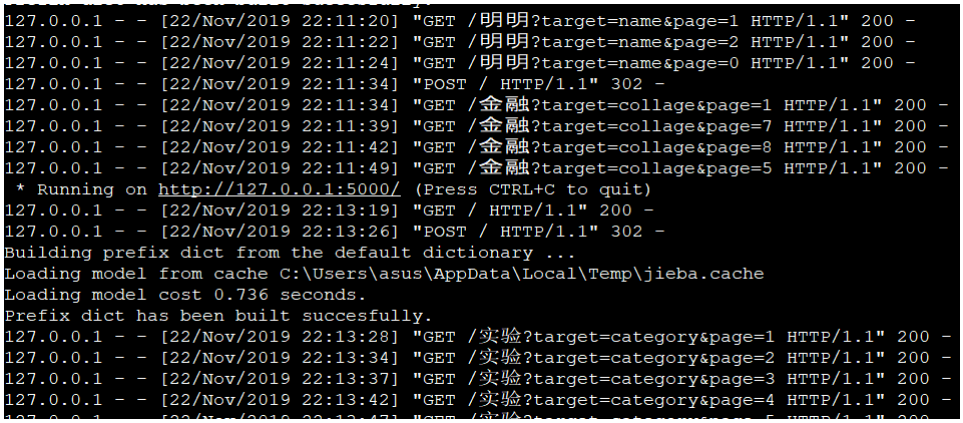

图 12: new.log

函数中使用 selenium 中的 webdriver 模拟火狐浏览器执行点击操作，根据指定的 URL 获取页面内容，存放在 browser 变量中。然后调用 get_fin_data(t_info, url) 函数，使用 BeautifulSoup 解析 HTML 文本，剖析器为 html.parser，根据网页的结构，找到需要信息的所在位置，对于金融学院的“师资队伍”页面，用 F12 开发者工具可以查看到我们需要的信息在“shiziListBox”中，调用 select() 函数取出信息。这里取出的是一些简要的信息，包含教师的个人网页链接，根据这个链接进行跳转，可以看到更详细的介绍在“teacherDetail”类中，可以使用 browser.find_element_by_class_name('teacherDetail').text，取出介绍的文字。由于个人简介的信息太长，并且不是关键信息，没必要全部存储，所以只取了前 100 个字。另外，在跳转的过程中，用 pagegraph 列表记录 PageRank 需要用到的顶点和边的信息。

```
def get_fin_data(t_info, url):
    urlset = set() # 用于判断是否重复
    try:
        # 使用剖析器为html.parser
        soup = BeautifulSoup(browser.page_source, 'html.parser')
        allList = soup.select('.shiziListBox')
        for teacher in allList:
            a = teacher.select('a')
            try: # 如果抛出异常就代表为空
                href = a[0]['href']
                browser.get('http://finance.nankai.edu.cn' + href)
                if href in urlset:
                    continue
```

```
            else:
                urlset.add(href)
            intro = browser.find_element_by_class_name('teacherDetail').text
            intro = intro[0:min(len(intro), 100)] + ' ……'
            sleep(1)
        except:
            href = ''
        try: # 如果抛出异常就代表为空
            e_mail = a[1].text
        except:
            e_mail = ''

        text = teacher.select('p')[0].text.split()
        name = text[0]
        if len(text) < 2:
            cate = ''
        else:
            cate = text[1]
        # 把爬取到的每条数据组合成一个字典用于数据库数据的插入
        new_dict = {
            "name": name,
            "cate": cate,
            "href": 'http://finance.nankai.edu.cn' + href,
            "e_mail": e_mail,
            "collage": '金融',
            "intro": intro
        }
        t_info.append(new_dict)
        pagegraph.append(
        [url, 'http://finance.nankai.edu.cn' + href, e_mail])
        print(new_dict)
    except Exception as e:
        print(e)


def finance():
    info = []
    urls = [r'http://finance.nankai.edu.cn/f/teacher/teacher/qzjs',
        r'http://finance.nankai.edu.cn/f/teacher/teacher/jzds',
```

```
        r'http://finance.nankai.edu.cn/f/teacher/teacher/rxjs']
    for url in urls:
        browser.get(url)
        get_fin_data(info, url)
    # 写入
    wf('TeacherInfo/finance.json', info)
```

5.2 建立索引相关代码

具体的流程已经在实现思路中说明，这里不再赘述，只展示关键部分。首先是创建 schema, stored 为 True 表示能够被检索。

```
schema = Schema(
    name=TEXT(stored=True, analyzer=analyzer),
    e_mail=ID(stored=True),
    href=ID(stored=True),
    collage=TEXT(stored=True, analyzer=analyzer),
    category=TEXT(stored=True, analyzer=analyzer),
    direction=TEXT(stored=True, analyzer=analyzer),
    department=TEXT(stored=True, analyzer=analyzer),
    intro=TEXT(stored=True, analyzer=analyzer),
)
```

之后存储 schema 信息至 indexdir 目录，按照 schema 定义信息，增加需要建立索引的文档。allLists 加载了所有教师的信息，每一条信息以字典方式存储。

```
# 存储schema信息至indexdir目录
indexdir = 'indexdir/'
if not os.path.exists(indexdir):
    os.mkdir(indexdir)
ix = create_in(indexdir, schema)

# 按照schema定义信息，增加需要建立索引的文档
writer = ix.writer()

for i in allLists:
    name = i.get('name', '')
    e_mail = i.get('e_mail', '')
    cate = i.get('cate', '')
```

```
    href = i.get('href', '')
    collage = i.get('collage', '')
    intro = i.get('intro', '')
    direction = i.get('direct', '')
    department = i.get('depart', '')
    writer.add_document(name=name, e_mail=e_mail, href=href, collage=collage,
        category=cate, intro=intro,
                    direction=direction, department=department)
writer.commit()
```

5.3 PageRank 算法代码

使用 PageRank 算法时，需要将网页间的指向关系用图的形式表示，pagegraph 中加载了在爬取过程中存储的 PRInfo/pageGraph.json，由 init_graph() 函数将它转换为图。

```
def init_graph():
    graph = nx.DiGraph() # 初始化一个空的图
    anchor = {}
    for t in pagegraph:
        graph.add_edge(t[0], t[1])
        # 记录锚文本及其对应的链接
        an = t[2]
        if an in anchor:
            anchor[an].append(t[1])
        else:
            anchor[an] = [t[1]]
    wf('PRInfo/anchor.json', anchor)
    return graph
```

对于 init_graph() 得出的有向图，在 cal_pr() 函数中，计算各个网页的 PR 值，将结果存入 PRInfo/pageRank.json。

```
def cal_pr():
    G = init_graph()
    pr = nx.pagerank(G, alpha=0.85) # 计算pr
    result = {}
    for node, value in pr.items():
        result[node] = value
    print(result)
```

```
    wf('PRInfo/pageRank.json', result)
```

5.4 内容检索相关代码

记录查询日志的代码：

```
logging.basicConfig(level=logging.DEBUG, # 控制台打印的日志级别
                    filename='new.log',
                    filemode='a', #
                        模式，有w和a，w就是写模式，每次都会重新写日志，覆盖之前的日志
                    # a是追加模式，默认如果不写的话，就是追加模式
                    format='%(message)s'
                    # 日志格式
                    )
```

重定义列表排序方式，将搜索结果按照 PR 值重新排序的代码：

```
def func(x, y):
    # 让每一项的大小比较转换为他们各自的按照PR值比较
    if pg[x['href']] < pg[y['href']]:
        return -1
    if pg[x['href']] == pg[y['href']]:
        return 0
    else:
        return 1


def resort(buf, f):
    if f == 0:
        global pg
        with open('PRInfo/pageGraph.json', 'r', encoding='UTF-8') as f:
            pg = json.load(f)
        f.close()
        # 进行重载
        sorted(buf, func)
```

使用 flask 框架创建网页，首页编写了 search() 用于处理浏览器发来的表单信息。然后调用 search_result() 函数。

```
app = Flask(__name__)


@app.route('/', methods=['GET', 'POST'])
def search():
    if request.method == 'POST':
        sent = request.form['query']
        target = request.form['type']
        try:
            isLogic = request.form['check']
        except:
            isLogic = '0'
        return redirect(url_for('search_result', sent=sent, target=target,
             isLogic=isLogic, page=1))
    else:
        return render_template('hello.html')
```

search_result() 中，buffer 列表用于缓存上一次搜索的结果，以便在换页时提高反应速度。题目要求中说可以查询 filetype，但对于教师搜索系统，并没有爬取到什么特殊类型的文件，所以这里只用图片 icon.png 做一个示例。接着判断本次查询是否是新的查询，如果不是新的查询，直接从 buffer 中取数据；如果是新的查询，需要加载索引，创建查询向量，进行查询。对于一个新的查询，具体地，先要判断是否是逻辑查询，如果是逻辑查询，需要用关键字“OR”进行拆分。用 QueryParser(target, schema=myindex.schema).parse(sent) 构造查询向量，使用 whoosh 已有的查询器进行查询，将结果放入 buffer。将对应的具体信息存入 details 并使用 PageRank 算法进行排序，将结果展示在 search_result.html 上。

```
@app.route('/<sent>')
# <sent>会被传来的参数替换，组成函数的路由
def search_result(sent):
    global buffer
    if sent == 'favicon.ico':
        with open("./static/images/icon.png", 'rb') as f:
            image = f.read()
        return Response(image, mimetype='image/png')
    page = int(request.args.get('page'))
    details = []
    target = request.args.get('target')
    isLogic = request.args.get('isLogic')
```

```
if page != 1: # 说明已经查询过，直接从缓存取数据
    if page < 1:
        page = 1
    max_page = min(int(math.ceil(len(buffer) / 3.0)), 7)
    if page > max_page:
        page = max_page
    tmp = max(0, page - 1)
    print('一共发现%d份文档。' % len(buffer), page)
    for i in range(tmp * 3, min(min(tmp * 3 + 3, 20), len(buffer))):
        details.append(buffer[i].fields()) #
            details中的每一项是一个词典，包含一个人的除简介外的所有信息
        print(i, " : ", json.dumps(buffer[i].fields(), ensure_ascii=False))
else:
    myindex = index.open_dir("indexdir")
    # 按域搜索，搜的类型为target，值为sent
    qp = QueryParser(target, schema=myindex.schema)

    if isLogic == '1':
        # 逻辑查询
        if 'OR' in sent:
            or_terms = sent.split('OR')
            q = Or([Term(target, or_terms[0]), Term(target, or_terms[1])])
    else:
        q = qp.parse(sent)
    searcher = myindex.searcher()
    buffer = searcher.search(q, limit=20)
    print(buffer)
    for i in range(min(3, len(buffer))):
        buffer[i].highlights("intro")
        details.append(buffer[i].fields()) #
            details中的每一项是一个词典，包含一个人的所有信息
        print(json.dumps(buffer[i].fields(), ensure_ascii=False))

    # PageRank算法排序
    resort(buffer, 1)
max_page = min(int(math.ceil(len(buffer) / 3.0)), 7)
if page > max_page:
    page = max_page
```

```
return render_template('search_result.html', sent=sent, target=target, page=page,
    max_page=max_page,details=details)
```

最后在主函数入口调用 app.run()，运行整个系统。

5.5 网页设计 HTML 代码

下面是 html 代码中与搜索和数据相关的部分。hello.html 上面主要有一个下拉框，用于选择查询域；有一个输入框，用于输入短语查询或逻辑查询；有一个语音输入的图标，点击之后可以利用 Web Speech API 的控制接口 SpeechRecognition 实现语音输入；有一个 CheckBox 用于选择是否进行逻辑查询；还有一个按钮用于提交表单。hello.html 关键代码如下所示：

```
<form action="{{ url_for('search') }}" method=post >
  <div class="styled-select">
  <select name="type">
   <option value="href"> Website</option>
   <option value="filetype">Filetype</option>
   <option value="name"> Name</option>
   <option value="e_mail"> Email</option>
   <option value="department">Department</option>
   <option value="category">JobTitle</option>
   <option value="direction">Direction</option>
   <option value="collage">Collage</option>
  </select>
  </div>
  <div class="div-d" >
  <input type="text" placeholder="Search here..." class="speech-input" name="query"
      required>
  <button type="submit"></button>
  </div>
  <div class="div-d" >
  <label style="font-size:12px;right: 20px;">Logic Quiry<label/>
  <input type="checkbox" name="check" value="1" style="zoom:40%;display:inline;" />
  </div>
</form>
```

search_result.html 的头部是与 hello.html 相同的搜索功能，不再粘贴代码，主体是搜索结果的展示，如果搜索结果为空，系统会给出提示：“众里寻 Ta 千百度，蓦然回首，那

人不在数据库”；如果结果不为空，将搜索到的字典放入表格，并在 pagination 类中进行分页，每页最多展示三个表格。代码如下：

```
<ul class=articles>
  {% if details|length > 0 %}
     {% for person in details %}
       <li><a href="{{person['href']}}"
           target="_blank"><h2>{{person['name']}}</h2></a>
       <table id="customers">
        {%for key in person if key !='name' and key !='href'%}
         <tr class="alt">
         <th> {{ key }} </th>
         <td><p> {{ person[key] }} </p></td>
         </tr>
        {% endfor %}
       </table>
     {% endfor %}
  {% else %}
    <li font-size:20px><em>众里寻Ta千百度，蓦然回首，那人不在数据库</em>
  {% endif %}
  </ul>

  {% if details|length > 0 %}
  <div align="center">
  <ul class="pagination">
    <li><a href={{url_for('search_result', sent=sent, target=target,
        page=page-1)}}>&laquo;</a></li>
    {% for i in range(1,max_page+1)%}
      {% if i == page %}
        <li class="active"><a href={{url_for('search_result', sent=sent, target=target,
            page=i)}}>{{i}}</a></li>
      {% else %}
        <li><a href={{url_for('search_result', sent=sent, target=target,
            page=i)}}>{{i}}</a></li>
      {% endif %}
    {% endfor %}
    <li><a href={{url_for('search_result', sent=sent, target=target,
        page=page+1)}}>&raquo;</a></li>
  </ul>
```

```
  </div>
{% endif %}
```

6 结果演示

1) 点击图标，进行语音输入，图 13

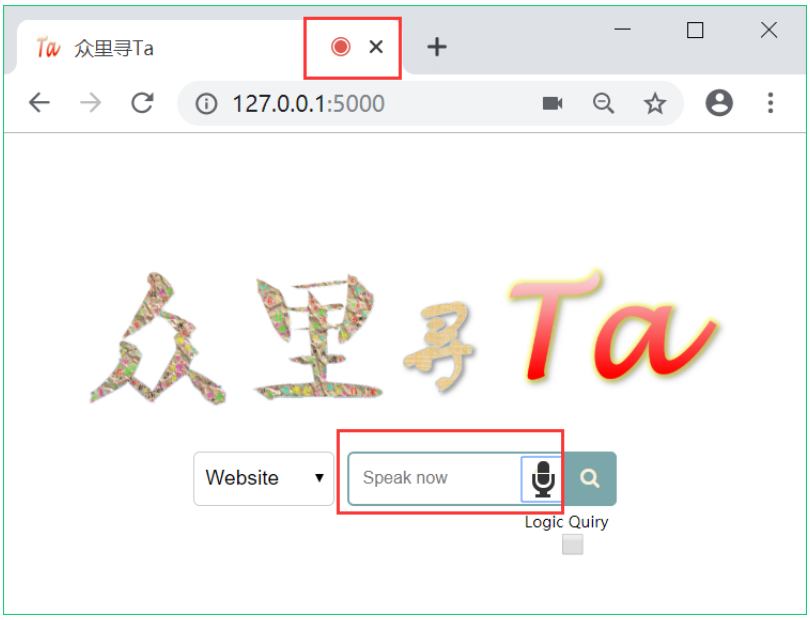

图 13: 语音输入

2) 搜索名字中含有“明明”的老师，图 14

3) 搜索研究方向是“人工智能”和“计算机”的老师，图 15

4) 搜索学院是“计算机”或“历史”的老师（逻辑查询），图 16

5) 搜索结果为空，图 17

6) 点击教师姓名，跳转到教师的个人网页，图 18

7) 搜索 favicon.ico，可以查看图片文件，图 19

8) 搜索职称中含有“实验”的老师，图 20

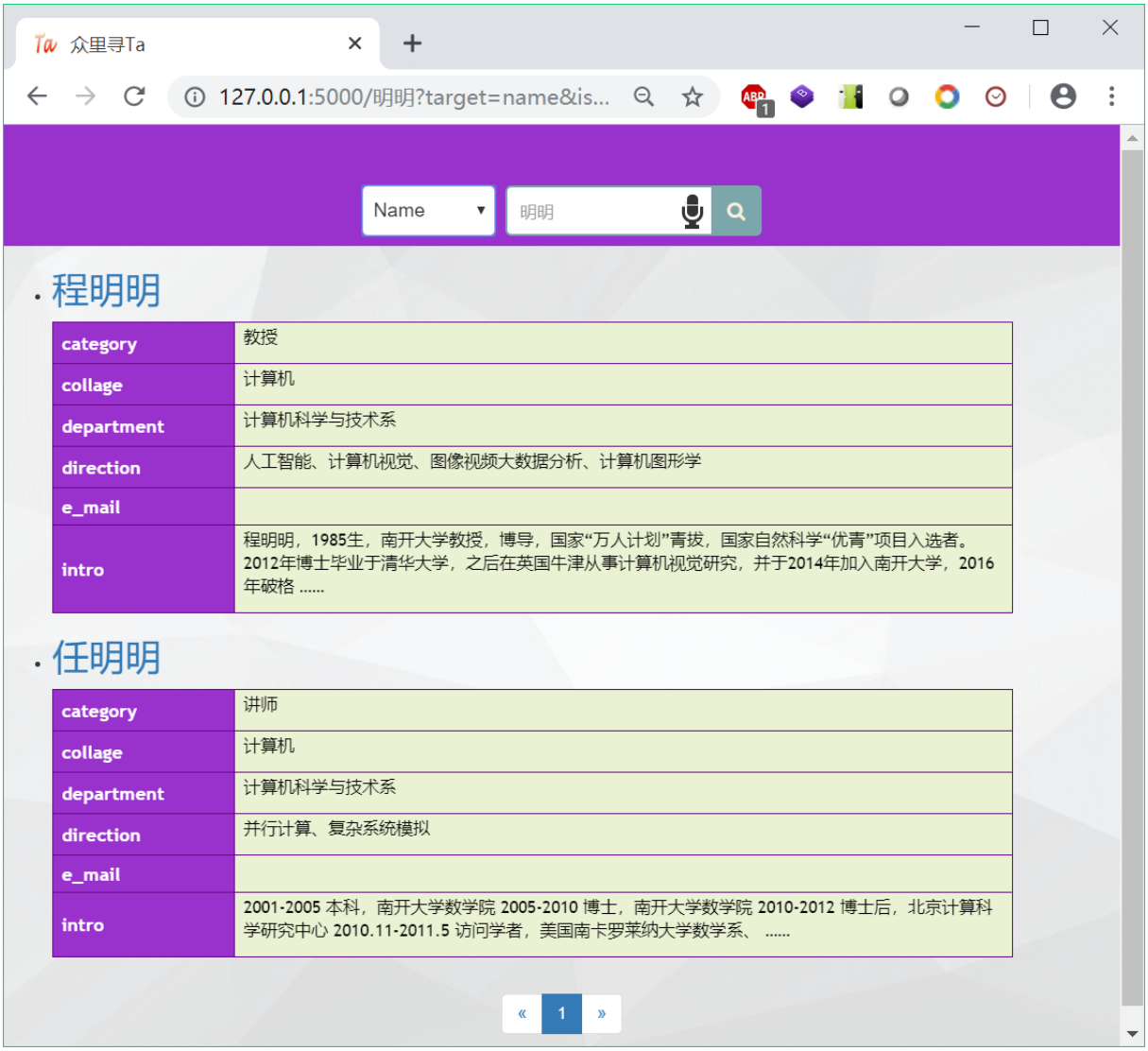

图 14: 名字中含有“明明”的老师

7 其他

记录一下实验中学到的比较零碎的东西以及踩的坑。

1) Seacher 不要过早地关闭，result 中取出来的并不是真的文字信息，而是在索引中的位置信息，searcher 关掉了就找不到文字信息了，浏览器会报 500.

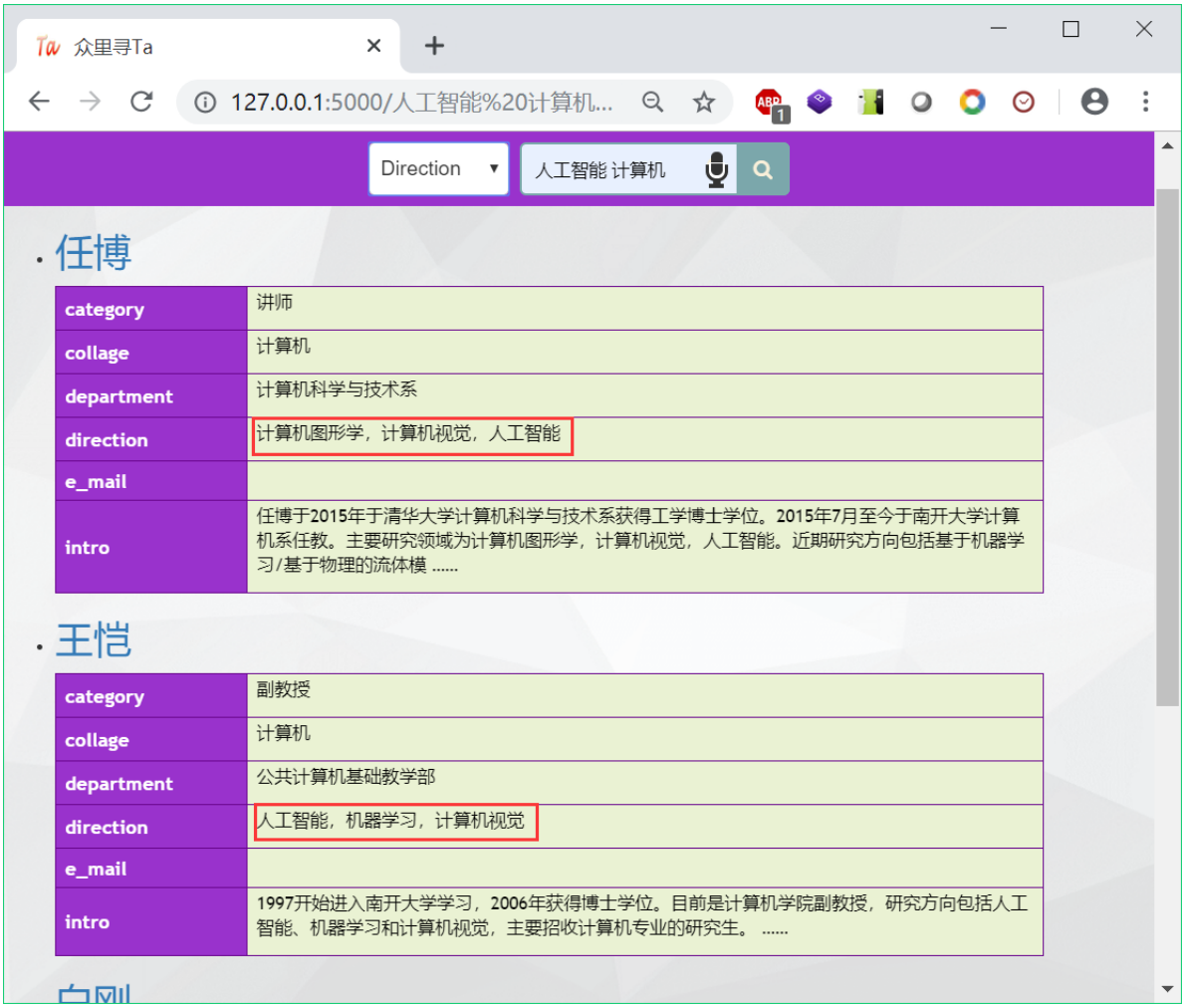

图 15: 研究方向是“人工智能”和“计算机”的老师

2) u/U: 表示 unicode 字符串，不是仅仅是针对中文, 可以针对任何的字符串，代表是对字符串进行 unicode 编码。

3) 字符创前面加’r’ 表示在原始字符串里，所有的字符都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。

4) 使用 LaTeX 的 lstlisting 宏包时，代码块中有中文注释就会编译报错，解决方案：以 TeXstudio 为例，选中 Options 的 Configure 截面，再选 build 截面将 Default Compile 改成 XeLaTeX 即可。

图 16: 学院是“计算机”或“历史”的老师

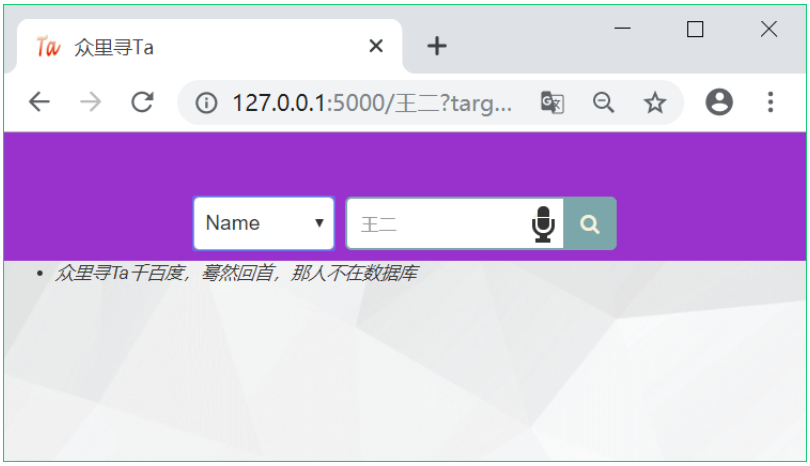

图 17: 搜索结果为空

图 18: 跳转

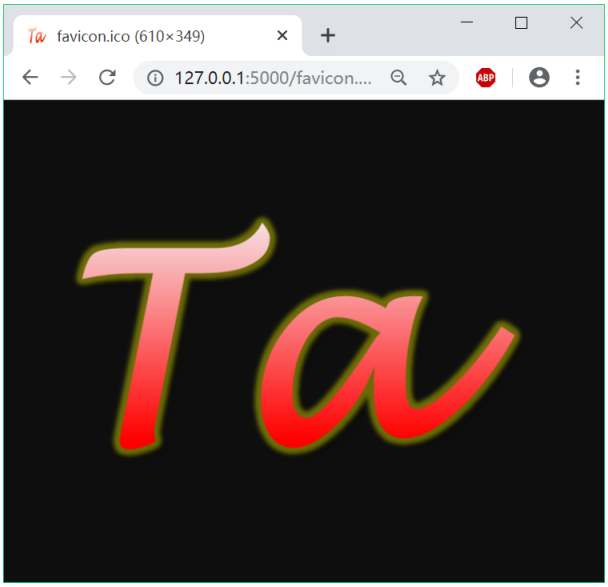

图 19: favicon.ico

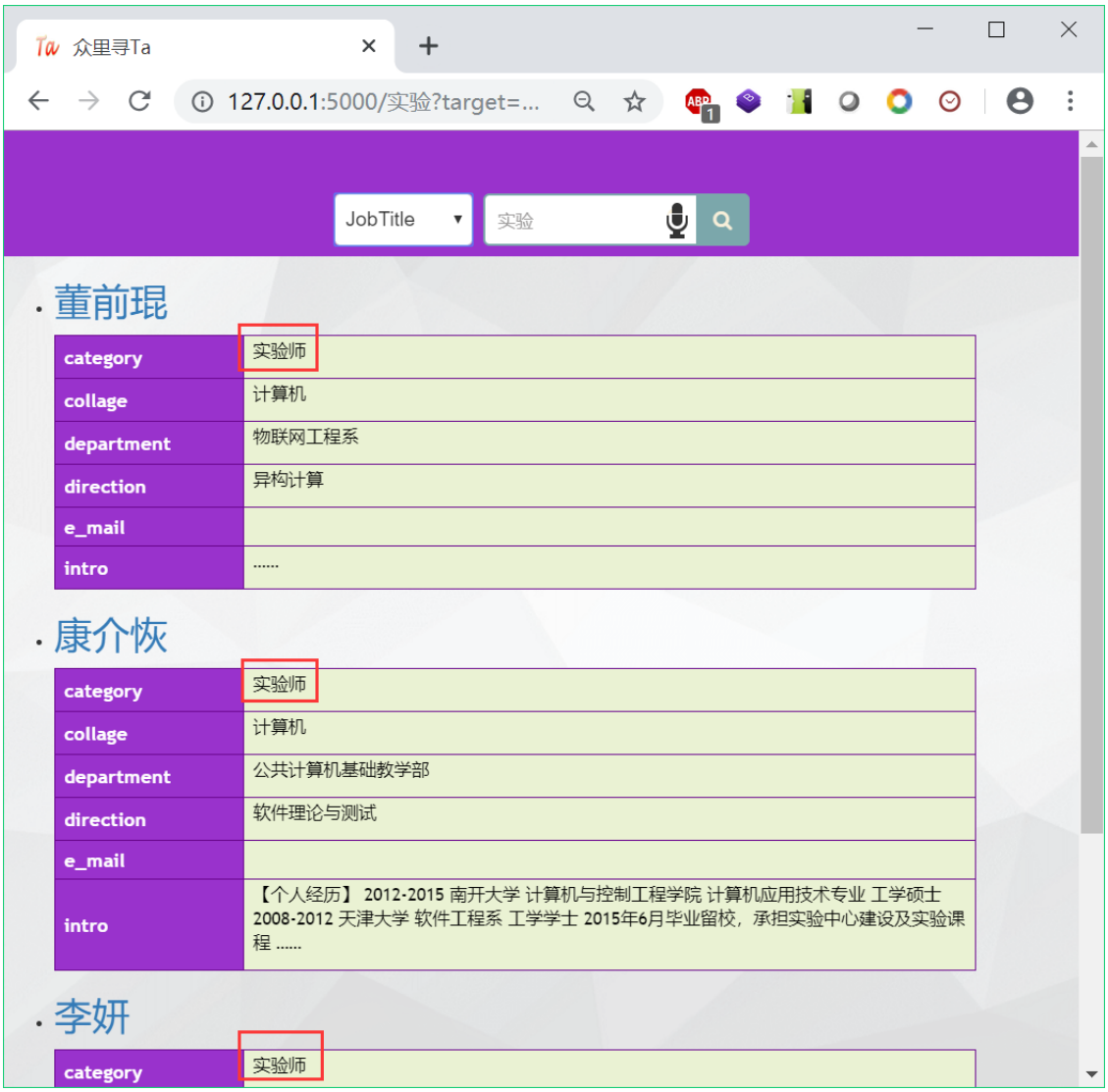

图 20: 职称中含有“实验”的老师