教师搜索引擎

史恩扬 1711373

2020/1/2

摘要

通过爬虫爬取南开大学内网下的所有网页页，对其建立索引，并进行 PageRank 分析，通过 UI 界面进行关键词检索或语音问答。

关键字： Spider PageRank Search Engine

1 工程简介

本工程可划分为四个部分（见图1），分别为网络爬虫，索引构建及链接分析，UI 界面搭建以及语音处理。

最终是要实现一个可以对南开内网信息进行检索的搜索引擎，并且通过锚文本分析，以及 $BM25$ 算法对匹配的结果进行排序。

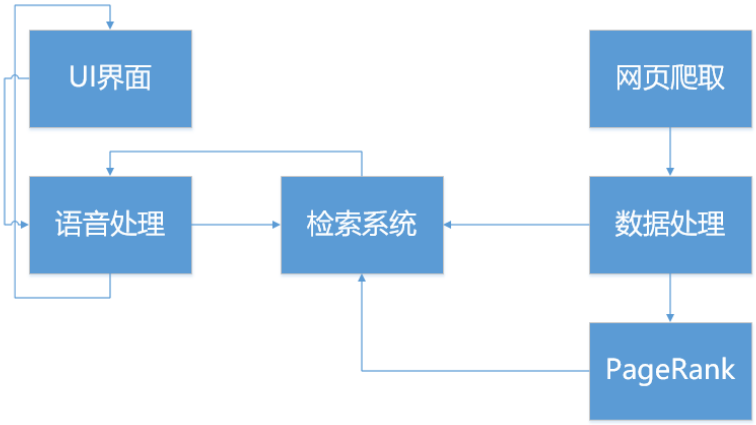

图 1: 工程结构

2 网页爬虫

通过 Python 支持的 Scrapy 框架编写网页爬虫。我们需要将南开内网中所有域名下的所有网页进行无差别的连续爬取，因此选用 Scrapy 的 Crawl 模板。

为了提高爬取的速率和准确率，将各个学院，南开新闻，南开主页的域名加入 $allow_set$ 中，将 bbs，软件之家等资源网站加入 $deny_set$ 中。制定爬取规则。

对每个符合规则的网页提取该网页的 title，url，纯文本，链接及链接附近的文本，将这些信息存入 item 中，并将 item 传给 pipeline。

```
1  # -*- coding: utf-8 -*-
2  import scrapy
3  from scrapy.linkextractors import LinkExtractor
4  from scrapy.spiders import CrawlSpider, Rule
```

```
5   from spider.items import SpiderItem
6
7
8   class InfoSpiderSpider(CrawlSpider):
9       name = 'info_spider'
10      allowed_domains = ["nankai.edu.cn"]
11      start_urls = ['http://news.nankai.edu.cn/']
12      allow_set = (
13                  "wxy\.nankai\.edu\.cn",
14                  "history\.nankai\.edu\.cn",
15                  "phil\.nankai\.edu\.cn",
16                  "law\.nankai\.edu\.cn",
17                  "zfxy\.nankai\.edu\.cn",
18                  "fcollege\.nankai\.edu\.cn",
19                  "cm\.nankai\.edu\.cn",
20                  "economics\.nankai\.edu\.cn",
21                  "bs\.nankai\.edu\.cn",
22                  "sms\.nankai\.edu\.cn",
23                  "physics\.nankai\.edu\.cn",
24                  "chem\.nankai\.edu\.cn",
25                  "sky\.nankai\.edu\.cn",
26                  "cc\.nankai\.edu\.cn",
27                  "ceo\.nankai\.edu\.cn",
28                  "env\.nankai\.edu\.cn",
29                  "medical\.nankai\.edu\.cn",
30                  "cs\.nankai\.edu\.cn",
31                  "hyxy\.nankai\.edu\.cn",
32                  "tas\.nankai\.edu\.cn",
33                  "teda\.nankai\.edu\.cn",
34                  "tedabio\.nankai\.edu\.cn",
35                  "iap\.nankai\.edu\.cn",
36                  "esd\.nankai\.edu\.cn",
37                  "pharmacy\.nankai\.edu\.cn",
38                  "finance\.nankai\.edu\.cn",
39                  "mse\.nankai\.edu\.cn",
40                  "stat\.nankai\.edu\.cn",
41                  "www\.riyan\.nankai\.edu\.cn",
42                  "ifd\.nankai\.edu\.cn",
43                  "cyber\.nankai\.edu\.cn",
44                  "ai\.nankai\.edu\.cn",
45                  'www\.nankai',
46                  'news\.nankai',)
47      deny_set = ('bbs\.nankai', 'career\.nankai', 'international\.nankai
    ', 'xxb\.nankai', 'std\.nankai',
48                  'soft\.nankai', 'weekly\.nankai', 'movie\.nankai', 'bt\.
    nankai', 'chinaeconomy\.nankai',
```

```
                    'paper\.lib\.nankai', 'soft\.nankai',
                    'jwc\.nankai', 'bt\.nankai', 'jw\.nankai', 'movie\.
    nankai', 'opac\.lib\.nankai', 'xxgk\.',
                    'nkuefnew\.nankai','lbs\.nankai','ibs\.nankai','lib\.
    nankai')
        rules = (
            Rule(LinkExtractor(allow=allow_set, deny=deny_set, unique=True),
     callback='parse_item',
                 follow=True),
        )

        def parse_item(self, response):
            item = SpiderItem()
            text_XPath = "//text()"
            title_XPath = "/html/head/title/text()"
            anchor = []
            x = LinkExtractor(allow=self.allow_set, deny=self.deny_set,
    unique=True, attrs=['href', 'text()'])
            links = x.extract_links(response)
            for link in links:
                anchor.append((link.url, link.text))
            item["my_url"] = response.request.url
            item["text"] = response.xpath(text_XPath).extract()
            item["title"] = response.xpath(title_XPath).extract()
            item["anchor"] = anchor
            return item
```

为了提高爬取速度，对于各个网页信息的过滤，规范化操作在 pipeline 中并行进行。将纯文本中的特殊符号剔除，仅保留文字信息，将处理后的网页信息以及对应的 ID 存在 json 文件中。为了后续的数据处理，将 id 与 url 对应的哈希表也存在 json 文件中。

```
    class SpiderPipeline(object):
    # id = 67597
    # url_dic = {}

    def __init__(self):
        self.id = 0

    def __del__(self):
        json_str = json.dumps(self.url_dic, ensure_ascii=False)
        path = "data/url2id.json"
        with open(path, "w", encoding="utf-8") as json_file:
            json_file.write(json_str)

    def process_item(self, item, spider):
```

```
86          text = ""
87          for i in item["text"]:
88              x = i.strip()
89              if (x != "" and x.find("    ") == -1 and x.find("javascript") ==
    -1):
90                  text += x + ' '
91          dic = {}
92          dic["id"] = self.id
93          dic["my_url"] = item["my_url"]
94          dic["text"] = text
95          dic["title"] = item["title"]
96          dic["anchor"] = item["anchor"]
97          self.url_dic[item["my_url"]] = self.id
98          self.id += 1
99          json_str = json.dumps(dic, ensure_ascii=False)
100         path = "data/doc/" + str(dic["id"]) + ".json"
101         with open(path, "w", encoding="utf-8") as json_file:
102             json_file.write(json_str)
103         return item
```

3 数据处理

3.1 索引建立

借助 Python 提供的 Whoosh 模块结合 jieba 模块提供的分词工具快速构建索引。索引的 Schema 包含网页对应的 ID，url，text，title，anchor text，netloc，date。其中 text 和 anchor text，netloc 是作为检索部分，url，title，date 会作为结果返回。

程序会遍历网页爬虫获取到的所有 json 文件，收集所有锚文本储存在一个字典中，再次遍历，通过自身 ID 获取本网页的锚文本，通过对自身的 url 进行分析得到对应的 netloc ，通过对文章纯文本中的日期进行正则匹配获取文章的发布日期，还有 json 文件中的其余信息添加到索引中。

```
104 def anchor_init(rootdir):
105 analyzer = ChineseAnalyzer()
106 schema = Schema(id=NUMERIC(stored=True, unique=True), url=ID(stored=True
    ), text=TEXT(analyzer=analyzer),
107                 anchor_text=TEXT(analyzer=analyzer), title=TEXT(stored=
    True, analyzer=analyzer),
108                 netloc=KEYWORD(scorable=True), date=DATETIME(stored=True
    ))
109 counter = 0
```

```
110     with open('spider/data/url2id.json', encoding='utf-8') as file:
111         url_dic = json.load(file)
112     if not os.path.exists('searching_engine/index'): # 如果目录index不存在
        则创建
113         os.mkdir('searching_engine/index')
114     create_in("searching_engine/index", schema) # 按照schema模式建立索引目
        录
115     anchor = {}
116     print("begin init anchor dic")
117     for root, dirs, files in os.walk(rootdir):
118         for name in files:
119             # 收集所有anchor
120             fp = os.path.join(root, name)
121             fp = fp.replace('\\', '/')
122             with open(fp, encoding='utf-8') as file:
123                 dic = json.load(file)
124             if(dic["anchor"].__len__!=0):
125                 for a in dic["anchor"]:
126                     id = url_dic.get(a[0])
127                     if(id):
128                         t = anchor.get(id)
129                         if(t):
130                             t += a[1]
131                         else:
132                             anchor[id] = a[1]
133                 # 统计link
134                     to_url = url_dic.get(a[0])
135             counter += 1
136             if (counter == 1000):
137                 counter = 0
138                 print(fp)
139     with open("searching_engine/anchor.json", 'w', encoding="utf-8") as
        anchor_file:
140         anchor_json = json.dumps(anchor, ensure_ascii=False)
141         anchor_file.write(anchor_json)
142     print("anchor init done")
143
144     def add_doc(ROOTDIR):
145     index = open_dir("searching_engine/index")  # 打开该目录一遍存储索引文
        件
146     writer = index.writer()
147     with open("searching_engine/anchor.json", encoding="utf-8") as
        anchor_file:
148         anchor = json.load(anchor_file)
149     print("begin init index")
150     counter = 0
```

```
151     for root, dirs, files in os.walk(ROOTDIR):
152         for name in files:
153             #  遍历所有文件
154             fp = os.path.join(root, name)
155             fp = fp.replace('\\', '/')
156             with open(fp, encoding='utf-8') as file:
157                 dic = json.load(file)
158             my_anchor = anchor.get(dic["id"])
159             t = parse.urlparse(dic["my_url"])
160             netloc = netloc_dic.get(t[1])
161             match = re.search(pattern, dic["text"], re.M | re.I)
162             # 构建index
163             try:
164                 if(my_anchor):
165                     if(netloc):
166                         if(match):
167                             writer.add_document(id=dic["id"], url=dic["
    my_url"],
168                                                 text=dic["text"], title=dic
    ["title"], anchor_text=my_anchor,
169                                                 date=datetime.datetime.
    strptime(match[1], "%Y-%m-%d"), netloc=netloc)
170                         else:
171                             writer.add_document(id=dic["id"], url=dic["
    my_url"],
172                                                 text=dic["text"], title=dic
    ["title"], anchor_text=my_anchor, netloc=netloc)
173                     else:
174                         writer.add_document(id=dic["id"], url=dic["my_url"],
175                                             text=dic["text"], title=dic["
    title"], anchor_text=my_anchor)
176                 else:
177                     if (netloc):
178                         if (match):
179                             writer.add_document(id=dic["id"], url=dic["
    my_url"], text=dic["text"], title=dic["title"],
180                                                 date=datetime.datetime.
    strptime(match[1], "%Y-%m-%d"), netloc=netloc)
181                         else:
182                             writer.add_document(id=dic["id"], url=dic["
    my_url"],
183                                                 text=dic["text"], title=dic
    ["title"], netloc=netloc)
184                     else:
185                         writer.add_document(id=dic["id"], url=dic["my_url"],
186                                             text=dic["text"], title=dic["
```

```
    title"])
187             except BaseException:
188                 continue
189             counter += 1
190             if(counter==1000):
191                 counter = 0
192                 print(fp)
193     writer.commit()  # index构建完成
```

3.2 PageRank

具体实现同上次实验，在此不再赘述

3.3 搜索端口

对于索引进行读取，并对传入的信息在 text，title 以及 anchor text 部分检索，并对结果按照 $BM25$ 算法进行排序。搜索结果可能日期并不相符，那么要对返回的 date 属性再次过滤

```
194     def search_for(content):
195     # 查询端口
196     index = open_dir("searching_engine/index")
197     with index.searcher() as searcher:
198         parser = MultifieldParser(['title', 'anchor_text', 'text', 'id', '
     netloc', 'date'], index.schema, fieldboosts={'anchor_text': 1.5})
199         query = parser.parse(content)
200         print(query)
201         corrected = searcher.correct_query(query, content)
202         # 纠错机制
203         if corrected.query != query:
204             q = parser.parse(corrected.string)
205         else:
206             q = query
207         match = re.search(pattern, content, re.M | re.I)
208         d = None
209         if(match):
210             d = datetime.datetime.strptime(match[1], "%Y-%m-%d")
211         result = searcher.search(q, limit=None)
212         res = []
213         for r in result:
214             x = dict(r)
215             if(d):
216                 try:
217                     date = x["date"]
```

```
                    if(date==d):
                        res.append([x["title"], x["url"]])
                except BaseException:
                    continue
            else:
                res.append([x["title"], x["url"]])
        return res
```

3.3.1 $BM25$ 算法

$BM25$ 算法是一种常见用来做相关度打分的公式，思路比较简单，主要就是计算一个 query 里面所有词和文档的相关度，然后再把分数做累加操作。公式如下：

$$
\begin{aligned}
Score(Q,d) &= \sum_{i}^{n} W_i \cdot R(q_i,d) \\
W_i &= IDF(q_i) = log\frac{N+0.5}{n(q_i)+0.5} \\
R(q_i,d) &= \frac{f_i \cdot (k_1+1)}{f_i+K} \cdot \frac{qf_i \cdot (k_2+1)}{qf_i+k_2} \\
K &= k_1 \cdot (1-b+b \cdot \frac{dl}{avgdl})
\end{aligned}
\tag{1}
$$

其中 $R(q_i,d)$ 是查询语句 query 中每个词 q_i 和文档 d 的相关度值，W_i 是该词的权重，一般情况下为 IDF(InverseDocumentFrequency) 值，即逆向文档频率,N 是文档总数，$n(q_i)$ 是包含该词的文档数，0.5 是调教系数，避免 $n(q_i)=0$ 的情况。log 函数是为了让 IDF 的值受 N 和 $n(q_i)$ 的影响更加平滑。k_1,k_2,b 都是调节因子，一般 $k_1=1,k_2=1,b=0.75$。式中 qf_i 为词 q_i 在查询语句 query 中的出现频率，f_i 为 q_i 在文档 d 中的出现频率。由于绝大多数情况下一条简短的查询语句 query 中，词 q_i 只会出现一次，即 $qf_i=1$。dl 为文档 d 的长度，$avgdl$ 为所有文档的平均长度。

4 语音处理

借助讯飞提供的语音听写以及语音合成功能完成与用户的交互，通过 websocket 与讯飞服务器建立连接，完成文字和语音的传输。

借助 Python 提供的 pyaudio 进行录音和播放

4.1 语音听写

借助 pyaudio，进行 10 秒的录音，将录好的 pcm 文件通过 websocket 上传到讯飞服务器，收到返回的 json 文件，其中包含了对于语音识别结果的分词及分析

```
225
226     def audio_record(out_file, rec_time):
227     CHUNK = 1024
228     FORMAT = pyaudio.paInt16  # 16bit编码格式
229     CHANNELS = 1  # 单声道
230     RATE = 16000  # 16000采样频率
231
232     p = pyaudio.PyAudio()
233     # 创建音频流
234     stream = p.open(format=FORMAT,  # 音频流wav格式
235                     channels=CHANNELS,  # 单声道
236                     rate=RATE,  # 采样率16000
237                     input=True,
238                     frames_per_buffer=CHUNK)
239
240     print("Start Recording...")
241
242     frames = []  # 录制的音频流
243     # 录制音频数据
244     for i in range(0, int(RATE / CHUNK * rec_time)):
245         data = stream.read(CHUNK)
246         frames.append(data)
247
248     # 录制完成
249     stream.stop_stream()
250     stream.close()
251     p.terminate()
252
253     print("Recording Done...")
254
255     # 保存音频文件
256     wf = wave.open(out_file, 'wb')
257     wf.setnchannels(CHANNELS)
258     wf.setsampwidth(p.get_sample_size(FORMAT))
259     wf.setframerate(RATE)
260     wf.writeframes(b''.join(frames))
261     wf.close()
262
263
264 def speechcapture():
```

```
265     t = []
266     with open("SpeechRecognizer/audio/data/voice2text.json", "w") as file:
267         json.dump(t, file)
268     audio_record("SpeechRecognizer/audio/question.pcm", 10)
269     # 测试时候在此处正确填写相关信息即可运行
270     time1 = datetime.now()
271     websocket.enableTrace(False)
272     wsUrl = wsParam.create_url()
273     ws = websocket.WebSocketApp(wsUrl, on_message=on_message, on_error=
        on_error, on_close=on_close)
274     ws.on_open = on_open
275     ws.run_forever(sslopt={"cert_reqs": ssl.CERT_NONE})
276     time2 = datetime.now()
277     print(time2 - time1)
278
279
280 def caputre():
281     speechcapture()
282     result = ""
283     with open("SpeechRecognizer/audio/data/voice2text.json", "r") as file:
284         t = json.load(file)
285         for i in t:
286             for w in i["cw"]:
287                 result += w["w"]
288         print(result)
289         return result
```

4.2 语音合成

通过将需要转换的文本发送到讯飞服务器，得到返回的 pcm 文件，再将其转化为 wav 格式，通过 pyaudio 播放出来

```
290     def speech(text):
291     wsParam.set_text(text)
292     speechanswer()
293     with open("SpeechRecognizer/audio/answer.pcm", 'rb') as pcmfile:
294         pcmdata = pcmfile.read()
295     with wave.open("SpeechRecognizer/audio/answer.wav", 'wb') as wavfile:
296         wavfile.setparams((1, 2, 16000, 0, 'NONE', 'NONE'))
297         wavfile.writeframes(pcmdata)
298     play_audio("SpeechRecognizer/audio/answer.wav")
```

5 UI 界面

借助 Python 支持的 tkinter，搭建一个简易的 UI 界面，提供文字输入的关键词搜索（见图2），以及语音问答搜索（见图3）。

对于语音问答目前支持询问某个学院某一天的新闻，当天的日期，当前的时间。

```
299 class MyUI(object):
300     # 将界面封装为类
301     def __init__(self):
302         print("begin init UI")
303         self.command = -1  # 记录命令类型
304         self.result = []
305         self.window = tk.Tk()
306         self.window.title("Xiao Kai")
307         self.window.geometry('800x500')
308         self.l2 = tk.Label(self.window, text='Please enter the information
    you search for',
309                            font=('Arial', 10), width=40, height=2)
310         self.msg = tk.Entry(self.window, show=None)
311         self.fm = tk.Frame(self.window)
312         self.b1 = tk.Button(self.fm, text='word search', width=10,
313                             height=2, command=self.search_by_word)
314         self.b2 = tk.Button(self.fm, text='voice search', width=10,
315                             height=2, command=self.search_by_speech)
316         self.text = tk.Text(self.window, height=10)
317         self.l2.pack()
318         self.msg.pack()
319         self.b1.pack(side='left')
320         self.b2.pack(side='left')
321         self.fm.pack()
322         self.text.pack()
323         self.date_pattern = r'((((1[6-9]|[2-9][0-9])([0-9]{2}))年
    (0?[13578]|1[02])月(0?[1-9]|[12][0-9]|3[01])日)|' \
324                             r'(((1[6-9]|[2-9][0-9])([0-9]{2}))年(0?[469]|11)月
    (0?[1-9]|[12][0-9]|30)日)|' \
325                             r'(((1[6-9]|[2-9][0-9])([0-9]{2}))年(0?2)月
    (0?[1-9]|1[0-9]|2[0-8])日)|' \
326                             r'(((1[6-9]|[2-9][0-9])([13579][26]))年(0?2)月(29)日)
    |' \
327                             r'(((1[6-9]|[2-9][0-9])([2468][048]))年(0?2)月(29)日)
    |' \
328                             r'(((1[6-9]|[2-9][0-9])(0[48]))年(0?2)月(29)日)|' \
329                             r'((([13579]6)(00))年(0?2)月(29)日)|' \
330                             r'((([2468][048])(00))年(0?2)月(29)日)|' \
```

```
331                     r'((([3579]2)(00))年(0?2)月(29)日)|' \
332                     r'((0?[13578]|1[02])月(0?[1-9]|[12][0-9]|3[01])日))'
333         self.netloc_dic = ['软件学院', '计算机学院', '生命科学学院', '生命科学学院', '文学院', '经济学院', '经济与社会发展学院', '经济与社会发展学院', '物流', '泰达学院',
334                            '泰达生物技术研究院', '滨海学院', '滨海研究院', '金融发展学院', '日本研究学院', '统计与科学学院', '医学院', '商学院', '网络空间安全学院', '物理学院',
335                            '药学院', '应用物理学院',
336                            '材料科学与工程学院', '电子信息与光学工程学院', '环境科学与工程学院', '环境科学与工程学院', '化学院', '汉语言文化学院', '旅游与服务学院', '旅游与服务学院',
337                            '医学院', '旅游与服务学院',
338                            '历史学院', '哲学院', '马克思主义学院', '法学院', '外国语学院', '商学院', '数学科学学院', '周恩来政府管理学院', '数学科学学院', '外国语学院',
339                            '人工智能学院', '住房管理',
340                            '经济学院', '智能计算系统研究室', '天津物理学会', '科学技术研究部', '就业指导中心', '南开', '教务', '新闻']
341         self.pattern = [
342             '今天(.)*(几号|日期)',
343             '现在(.)*(几点|时间)',
344             '新闻'
345         ]
346         self.text.tag_config('link', foreground='blue', underline=True)
347 
348         self.window.mainloop()
349 
350     def search_by_word(self):
351         my_msg = self.msg.get()
352         self.msg.delete(0, 'end')
353         print(my_msg)
354         self.search(my_msg, 1)
355         ans = "以下是对于这些关键词的搜索结果"
356         speech(ans)
357 
358     def search_by_speech(self):
359         my_msg = caputre()
360         self.search(my_msg, 0)
361 
362     def show_hand_cursor(self, event):
363         self.text.config(cursor='arrow')
364 
365     def show_arrow_cursor(self, event):
366         self.text.config(cursor='xterm')
367 
```

```
368     def click(self, event, x):
369         webbrowser.open(x)
370 
371     def handlerAdaptor(self, fun, **kwds):
372         return lambda event, fun=fun, kwds=kwds: fun(event, **kwds)
373 
374     def search(self, my_msg, type):
375         # type 为0时，代表是对话；为1时，代表是关键词搜索
376         self.text.delete('1.0', 'end')
377         if(type==0):
378             # 处理对话
379             result = self.pattern_match(my_msg)
380         elif(type==1):
381             result = search_for(my_msg)
382             print(result)
383         if(result):
384             self.text.insert('end', "Here are the results of searching
    "+"\""+my_msg+"\"")
385             self.text.insert('end', '\n')
386             self.text.insert('end', '\n')
387             k = 0
388             for i in result:
389                 if(k > 10):
390                     break
391                 self.text.tag_config(k, foreground='blue', underline=True)
392                 self.text.tag_bind(k, '<Enter>', self.show_hand_cursor)
393                 self.text.tag_bind(k, '<Leave>', self.show_arrow_cursor)
394                 self.text.insert('end', i[0], k)
395                 self.text.insert('end', '\n')
396                 self.text.tag_bind(k, '<Button-1>', self.handlerAdaptor(self
    .click, x=i[1]))
397                 k += 1
398             self.text.pack()
399 
400     def pattern_match(self, content):
401         i = 0
402         for p in self.pattern:
403             match = re.search(p, content, re.M | re.I)
404             if (match):
405                 print(match)
406                 break
407             i += 1
408         if (i == 0):
409             a = datetime.datetime.now()
410             ans = "今天是" + str(a.year) + '年' + str(a.month) + '月' + str(
    a.day) + '日'
```

```
411             speech(ans)
412             return None
413         elif (i == 1):
414             a = datetime.datetime.now()
415             ans = "现在是" + str(a.hour) + '点' + str(a.minute) + '分'
416             speech(ans)
417             return None
418         elif (i == 2):
419             match1 = re.search(self.date_pattern, content, re.M | re.I)
420             netloc = None
421             for t in self.netloc_dic:
422                 n = content.find(t)
423                 if (n != -1):
424                     netloc = t
425                     break
426             if (netloc and match1):
427                 if (match1[1].find('年') == -1):
428                     # date = datetime.datetime.strptime(datetime.date.today
    ().year+match1[1], '%Y年%m月%d日')
429                     date = datetime.datetime.strptime("2019年" + match1[1],
    '%Y年%m月%d日')
430                 else:
431                     date = datetime.datetime.strptime(match1[1], '%Y年%m月%d
    日')
432                 msg = "text:" + date.isoformat()[:10] + " netloc:" + netloc
433                 print(msg)
434                 res = search_for(msg)
435                 print(res)
436             ans = "以下是对于问题：" + content + "的搜索结果"
437             speech(ans)
438             return res
439         else:
440             ans = "对不起，我还不会这类问题"
441             speech(ans)
442             return None
```

6 试用反馈

汪珂航：能较好的返回所需结果

孙晓辉：希望增加更多搜索模式

江玥：语音处理比较准确

任冠铭：搜索结果比较准确，但不够全面

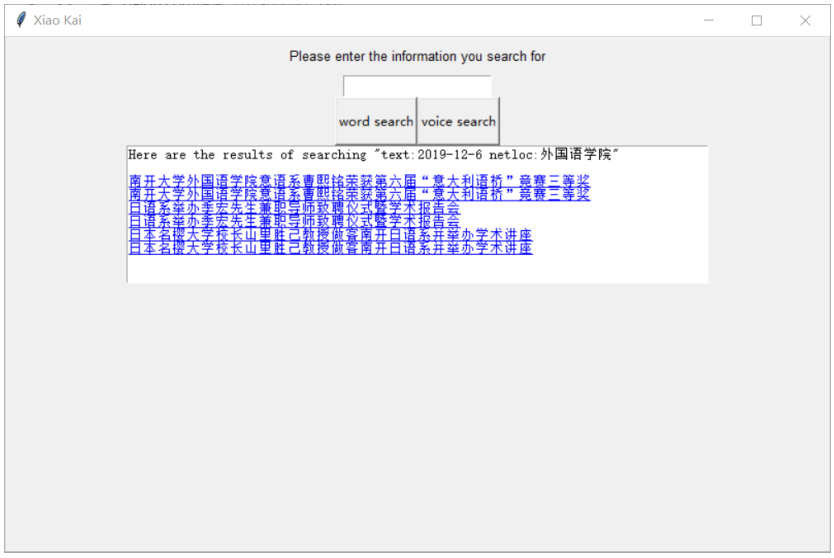

图 2: 关键词搜索

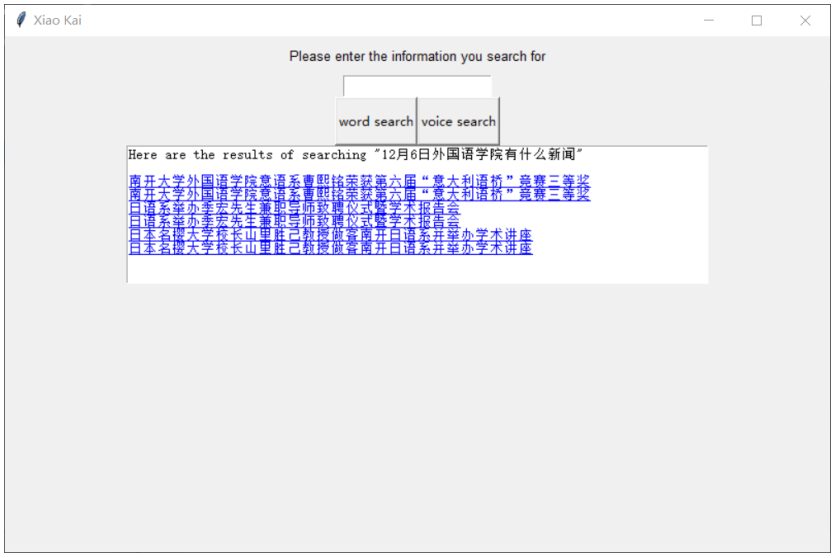

图 3: 语音问答