一个基于关键词爬虫生成词云的网络内容可视化工具

关键词：爬虫、词云、可视化

作者：

一、程序要解决的问题（目的/功能）

现如今，随着网络技术的不断发展以及网络普及水平的不断提高，海量的内容每天都在互联网上被产生和存储。当我们想要了解某个事物的时候，最先想到的就是上网搜索，而去哪里搜索，本身又有多种选择。面对各种搜索引擎呈现在用户面前的大量搜索结果，我们想要窥知这件事物在这些内容中的整体形象，要付出巨大的时间成本（一页一页的刷）；此外，不同的搜索引擎，由于其自身算法的差异，在搜索结果上往往会有所不同，而如果我们一页一页刷，也很难发现具体是什么样的不同。而这个程序所要解决的就是这样两个问题。

程序选取常见的几个搜索渠道：百度、必应（国内版）、谷歌、知乎（搜狗知乎搜索）、微信公众平台（搜狗微信搜索）和微博，针对用户在这四个平台进行搜索的结果内容进行爬取，并利用数据可视化的手段绘制出不同平台对应搜索结果的词云，将用户搜索的关键词对应结果的整体面貌和不同搜索渠道的这种面貌的差异，直观的呈现给用户。

二、程序设计思路

程序用到的重要第三方库包括：

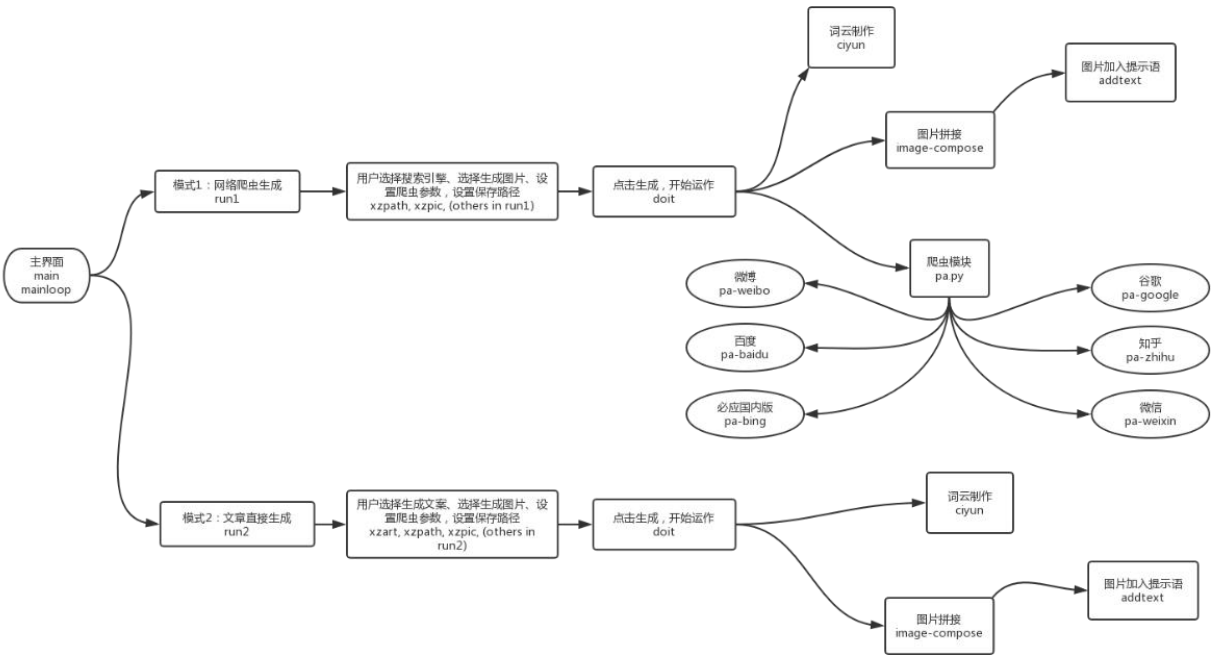
- ✓ selenium——模拟访问相应网页并获取源代码
- ✓ bs4（Beautifulsoup）——分析源代码提取内容
- ✓ wordcloud——生成词云
- ✓ PIL——对得到的词云图片进行处理

✓ jieba——对中文文本进行分词

此外用到的一些库还包括 time, urllib, tkinter, re, numpy, matplotlib 等。

程序采用自顶向下的设计思路，将负责各个功能的代码模块化逐层封装。程序还构建了一个图形化界面，方便用户进行操作。

程序整体上划分为三个模块：图形界面模块、网络爬虫模块和词云生成模块。图形界面模块通过图形化界面提示和引导用户输入相关信息，包括选择哪种模式、选择哪种搜索引擎、图片所在路径等等，之后将这些信息存储起来，进行网络爬虫和词云生成工作。之后是网络爬虫模块，负责网络爬虫工作并将信息存储在一个文本文件里头。最后是词云生成模块，需要将指定的文本文件里的内容，经过分词后，生成词云。



图表 1 自顶向下的程序设计图

整体分工： [ ] 主要负责网络爬虫和数据提取，[ ] 主要负责图形界面和词云生成，两人共同负责模块之间的联合测试和优化。

程序除了关键词网络爬虫生成词云外，也提供了文章生成模式，这两种模式最终都会生成一张词云图，只不过信息来源不同：第一种模式采用网络爬虫的方式，用户选择想要的搜索引擎，输入想搜索的关键词，程序会自动生成网页并且搜集信息，将搜集的信息经过分词之后生成词云；第二种方式根据用户选择本地文章的目录，用其本地的文章来生成词云。

程序源代码以 `pic.py` 作为**主文件**，实现词云生成，图形界面功能；以 `pa.py` 作为**附文件**，实现网络爬虫及数据提取功能。`pa.py` 以在 `pic.py` 中 import 的方式起作用，同时 `pa.py` 中的函数运行时生成 read.txt 文件存储爬取的数据，供 `pic.py` 中词云生成函数使用。

`pic.py` 代码行数为 235 行，`pa.py` 代码行数为 208 行（均不含空行）。考虑到这个项目是两人协作完成而且代码不同函数之间存在一些相似内容，我们认为源代码的行数可以算是符合要求的。

三、代码分析与阐述

网络爬虫模块

网络爬虫模块针对每个平台封装了相应的爬取页面数据的函数和分析提取文本的函数。

这一部分需要面对的难点主要在于不同的平台其源代码的风格不同，因此在搜索框和按钮、内容呈现和翻页这几部分的定位方式都具有相当的差异，爬虫程序要做的就是根据不同网站的 html 源代码差异调整策略，灵活运用 id 查找、class name 查找、css selecor 查找、link_text 查找等多种定位手段，精确定位需要自动操作的元素。并处理可能影响访问搜索结果的因素（如 google 的结果自动省略）。配合 selenium 提供的 webdriver，最终实现根据用户的查找词，自动在搜索框内输入并搜索，自动翻页，自动爬取源代码。用户也可以实时看到这个自动操作的过程。

搜索引擎以爬取百度搜索结果为例。注释中是对代码逻辑的详细阐述。

```
    # 百度
    def analyze_baidu(html, file):  # 分析页面源代码并提取文本
        h = BeautifulSoup(html, 'lxml')
        title = h.find_all('div', class_='c-abstract')  # 提取搜索结果中内容摘要对应的部分
        for item in title:
            file.write(item.get_text())  # 获取的文本输出至文件中
        file.write(line)

    def pa_baidu(string, pagelen):  # 访问爬取页面并获得源代码
        brs = webdriver.Chrome()
        brs.get('https://www.baidu.com/')
        WebDriverWait(brs, 10).until(EC.presence_of_element_located((By.ID, "kw"))) # 等待网页加载完毕，10 秒超时
        # 初次搜索并写入数据
        fbaidu = open('read.txt', 'w', encoding='utf-8')
        text = brs.find_element_by_id('kw')
        text.send_keys(string)
        time.sleep(0.2)
        search = brs.find_element_by_id('su')
        search.click()
        hcode = brs.page_source
        analyze_baidu(hcode, fbaidu)
        # 翻页循环搜索
        time.sleep(1)
        for i in range(eval(pagelen)):
            WebDriverWait(brs, 45).until(EC.visibility_of_element_located((By.LINK_TEXT, "下一页>")))
            nextpage = brs.find_element_by_link_text('下一页>')
            try:  # 在测试中这里的翻页不稳定，因此采用 css 选择器查找作为备用翻页按钮查找方式
                nextpage.click()
            except:
                time.sleep(0.3)
                nextpage = brs.find_element_by_css_selector('#page > a:nth-child(12)')
                nextpage.click()
            time.sleep(0.5)
            hcode = brs.page_source
            analyze_baidu(hcode, fbaidu)
        brs.close()
```

必应国内版和谷歌的代码与百度类似。

不过，由于必应的翻页按钮没有文字可供查找，而采用类名查找又会报错，因此采用 css 选择器进行定位，并根据切换页面那一行元素个数的变化动态调整 css 选择器里参数的值：

```
    for i in range(eval(pagelen)):
        if i == 0:
            j = 6
        elif i == 1:
            j = 8
        else:
            j = 9
        # 下面一行是等待页面上翻页按钮加载出来，45 秒后超时
        WebDriverWait(brs, 45).until(EC.visibility_of_element_located(
(By.CSS_SELECTOR, '#b_results > li.b_pag > nav > ul > li:nth-
child({}) > 'a'.format(j))))
        nextpage = brs.find_element_by_css_selector('#b_results > li.b
_pag > nav > ul > li:nth-child({}) >a'.format(j))
        nextpage.click()
        time.sleep(0.5)
        hcode = brs.page_source
        analyze_bing(hcode, fbing)
    brs.close()
```

而对于谷歌，主要的问题在于其会自动省略显示搜索引擎认为相关度最高的结果，并在这些结果的最后一页才进行提示，并提供搜索完整结果的链接。在实际测试中，省略后显示的范围从 11 页至 26 页不等。这相当于限制了我们获取搜索结果的范围，因此需要尽快翻到最后一页解除该限制，再重新搜索，这里使用了 try-except 结构判断页面中是否出现最后一页展示完整搜索结果的链接。

```
# 检索关闭搜索结果省略
if eval(pagelen) > 10:
    while 1:
        try:
            brs.find_element_by_link_text('下一页')
            n = brs.find_element_by_css_selector('#nav > tbody > tr > td:nth
-child(11) > a')
            if eval(n.text) > eval(pagelen):  # 如果此时跳转到的页面已经超出了
搜索范围，则直接重新开始搜索
                break
```

```
            n.click()
        except:
            showall = brs.find_element_by_link_text('重新搜索以显示省略的结果
')
            showall.click()
            break
hcode = brs.page_source
analyze_google(hcode, fgoogle)
```

对于**社交平台**的爬取以微博为例。如果没有登录，微博只能展示第一页搜索结果；我们当然可以自动填充用户名和密码框，然后自动点击登录，但是测试发现微博有时会要求用户输入验证码，于是这一条道路的可行性便大大降低。

而扫码登录虽然需要用户手机配合操作，但是程序的操作相对简单而且登录速度快。并且，几个社交平台都有扫码登录的选项，而且微信搜索更是仅支持扫码登录，登录前后页面的元素也基本不变。综合考虑，选择了在程序中留给用户 30 秒左右时间扫码登录的方式，并配以相关的提示信息提醒用户进行登录操作。下面展示了微博爬虫函数的前半段：

```
def pa_weibo(string, pagelen):
    print("微博搜索要求登录后才能查看完整的搜索结果。扫码登录是唯一的方式，这让我们无法在这一步实现自动化。\n 请打开微博扫一扫界面，静待二维码出现！")
    brs = webdriver.Chrome()
    brs.get('https://weibo.com/')
    WebDriverWait(brs, 10).until(EC.presence_of_element_located((By.CLASS_NAME, "gn_search_v2 ")))
    # 初次搜索并写入数据
    fweibo = open('read.txt', 'w', encoding='utf-8')
    # 处理登录的问题
    login = brs.find_element_by_link_text('安全登录')
    login.click()
    print('请扫描二维码登录以继续。请在 30 秒内完成操作！')
    # 留时间在手机上操作，检测到已经登录后继续下一步，45 秒后超时
    WebDriverWait(brs, 45).until(EC.presence_of_element_located((By.CLASS_NAME, "nameBox")))
.......
```

微信和知乎的代码逻辑与微博类似。而知乎和微信的搜索均为通过搜狗附带的搜索

引擎进行搜索。代码中也使用了类似手段引导用户进行登录，并及时检测用户登录状态。

图形界面模块

我们采用 tkinter 库来进行图形化界面的生成，这个库可以采用面向过程的模式来编程，虽然说有些缺点但是对于我们还没有接触过 Python 面向对象编程，这个库显得非常友好，同时编程思路也相对清晰。

最开始我们需要生成一个主页面，用 tkinter 库里的 Tk 函数即可，取名为 root，之后设置大小和标题。设置一个提示语，放在合适的地方。再设置两个选项，代表两种模式，点击之后执行相应的函数，并把这两个选项放在合适的地方，再用 root.mainloop() 运行 root 界面即可。这部分代码放在主函数里头：

```
1.  #主函数
2.  root = Tk()
3.  root.geometry('460x240')
4.  root.title("模式选择")
5.  lb1 = Label(root, text="欢迎使用本工具，请选择一种方式生成词云：")
6.  lb1.place(relx=0.1, rely=0.1, relwidth=0.8, relheight=0.1)
7.  var = IntVar()
8.  rd1 = Radiobutton(root, text="关键词爬虫生成
    ", variable=var, value=0, command=run1)
9.  rd1.place(relx=0.1, rely=0.4, relwidth=0.3, relheight=0.1)
10. rd2 = Radiobutton(root, text="本地文章生成
    ", variable=var, value=1, command=run2)
11. rd2.place(relx=0.6, rely=0.4, relwidth=0.3, relheight=0.1)
12. root.mainloop()
```

之后就要写 run1 和 run2 函数了，前者代表关键词爬虫生成模式，后者代表本地文章生成模式。

这两个函数大同小异，都是通过弹出一个子窗口来提示用户输入相关信息，同时也路径选择按钮帮助用户选择路径，再将用户所输入的参数传递到全局变量中，通过全局变量的参数来生成词云。这些功能通过 tk 库非常容易实现，且代码较长，这里不再对代码进行阐述。

词云生成模块

用图形化界面和爬虫模式将相关参数确定好之后，就可以用词云生成模块了，该模块只用到了一个函数，但用到了 re（正则表达式）、wordcloud（词云生成库）、PIL 库、numpy（多维运算）库、jieba 库等多个第三方库。

这个函数我们取名为 ciyun，设定五个参数：article（文本文件路径）、picture（图片路径）、word_max（最大生成词数）、stopwords（跳过的词列表）、result_path（图片保存路径）。首先打开 article 文件，将其内容用字符串存在一个变量 word 里，之后去除空格。之后用 numpy 打开图片 picture，再选择字体，我们用雅黑粗体，系统路径 C:\\Windows\\Fonts\\msyhbd.ttc。再用正则表达式将 word 里的英文数字字符等东西删去，之后用 jieba 分词，再转化成字符串。之后导入 stopwords，用 WordCloud 生成图片，最后用 plt 显示出来之后保存即可。代码如下：

```
1.  def ciyun(article, picture, word_max, stopwords, result_path1):
2.      with open(article, 'r', encoding='utf-8') as f:
3.          word = f.read()
4.          f.close()
5.      word=word.replace(' ','')
6.      image1 = np.array(image.open(picture))
7.      font = "C:\\Windows\\Fonts\\msyhbd.ttc"
8.
9.      resultword = re.sub("[A-Za-z0-
    9\[\`\~\!\@\#\$\^\&\*\(\)\=\|\{\}\'\:\;\'\,\[\]\.\<\>\/\?\~\。
    \@\#\\\&\*\%]", "", word)
10.     wordlist_after_jieba = jieba.cut(resultword)
11.     wl_space_split = " ".join(wordlist_after_jieba)
12.     ls = stopwords
13.     sw = set(STOPWORDS)
14.     for i in range(len(ls)):
15.         sw.add(ls[i])
16.
17.     my = WordCloud(
18.         scale=4,
19.         font_path=font,
20.         mask=image1,
21.         stopwords=sw,
22.         background_color='white',
```

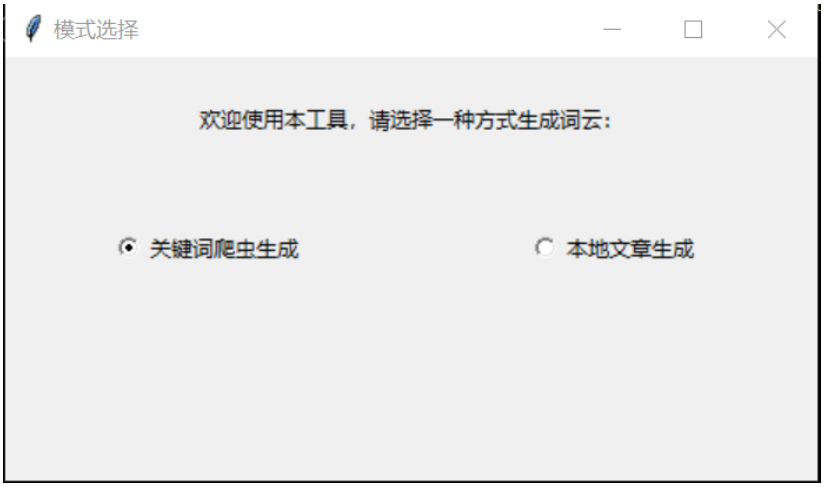
```
        max_words=word_max,
        max_font_size=60,
        random_state=20
    ).generate(wl_space_split)

    plt.imshow(my)
    plt.axis("off")
    plt.show()
    my.to_file(result_path1)
    return
```

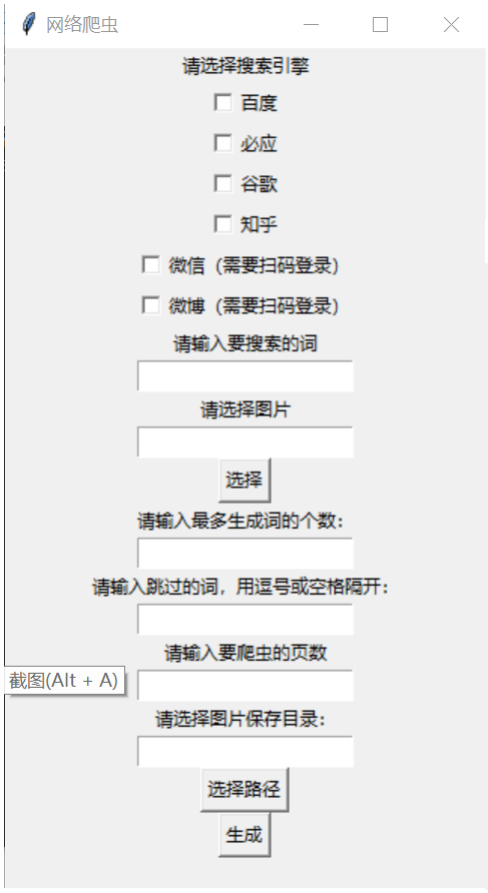
接下来对于网络爬虫生成的图片，需要在里头加上文字信息，提示是通过什么搜索引擎，用什么关键字生成的。我们可以通过 PIL 库轻松实现加入文字。

最后我们需要将爬虫生成的图片拼接在一起，这样方便对比搜索引擎对于同一个关键词搜索内容的不同。我们可以用 PIL 库根据要拼接图片的数量，生成相应大小的空白图片，并且分割成不同块，再将词云图片分别复制到相应的块上，这样就可以实现图片的拼接。

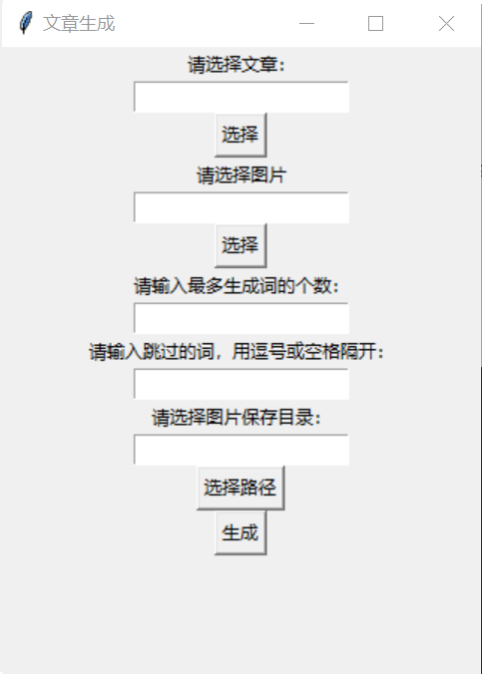
四、使用效果示例



图表 2 图形化界面一——模式选择



图表 3 图形化界面二——爬虫模式配置



图表 4 图形化界面三——文章模式配置

图表 5 生成的单张词云图片——“华为”在微信公众平台的搜索结果

图表 6 合成图片示例——“新疆”在全部六个平台的搜索结果对比

五、应用与展望

由于时间和开发水平的限制，目前这个工具在某些功能的实现上还有一些改进的空间：

- ✓ 整体运行时间较长，这是因为没有考虑应对可能出现的反爬虫机制的手段，因此只能采用 sleep 一定时间的方式保证程序不会被要爬取的网页识别；
- ✓ 微信、微博这样一些网络平台的内建搜索无法自动登录，采用了用户手动扫码登录的手段以获取完整搜索结果；
- ✓ 图形化界面的 UI 设计和交互逻辑还有待优化；
- ✓ 词云生成时目前只支持中文词，对无意义词的过滤还有优化的空间。

如果有机会，我们后续会继续针对上面几点进行改进。

不过，这依然不妨碍该工具已经可以产生的的实用价值。回顾该工具可以实现的功能：

- ✓ 让用户直观感受到关键词搜索结果的整体面貌；
- ✓ 让用户直观体会到不同搜索渠道得到的搜索结果的差异。

对于个人用户来说，如果想要了解自己的名字，自己所在的学校/工作单位，抑或是一些新闻热点在网络平台上的整体内容面貌，这个工具无疑是相当实用的。上面的示例就是很好的证明。此外，如果想要在不同的搜索引擎间做出选择，也可以用这个工具来判断哪一种搜索渠道更适合自己的口味。数据可视化和不同搜索渠道的对比，本身就可以产生极大的乐趣。

而对于单位用户来说，这个程序可以方便地了解单位在各大网络平台上的整体形象，是一个实用的了解舆情和协助处理公共关系的小工具。