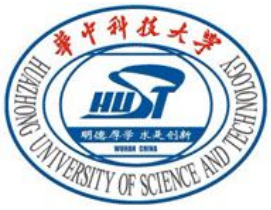


華中科技大學

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

电子信息与通信学院

实　验　报　告

（2017 / 2018 学年　第 1 学期）

课程名称	软件课设
实验名称	面向知乎 QAWEB 的网络爬虫设计与实现
指导教师	魏蛟龙
课程助教	聂锦燃
小组成员	游浩然　黎张帆　郭金城

姓名	游浩然	学号	U201515429

2018 年 1 月 4 日

目录

1 设计目的

- 加深对网络框架、协议及软件编程技术的理解
- 锻炼网络编程与解决实际问题的能力
- 熟练利用 Python 进行算法实现，培养将专业技能转换为实践的能力

2 开发平台

- 硬件平台：个人 PC，可连接 WEB 网络
- 软件环境：
  - 操作系统：WINDOWS 10 & Ubuntu
  - IDE：PyCharm Community Edition 2017.2.3
  - Python 版本：Python 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 03:27:45)
- 项目所需库
  - Wxpthon：Python GUI 库
  - Selenium：Python 模拟浏览器操作库
  - Phantom：基于 selenium 的 Python 工业化模拟浏览器操作库，需设置运行路径或将 bin 文件夹放入系统路径中。

3 项目描述

随着时代发展，网络成为大量信息的载体，如何有效地提取并利用这些信息成为一个巨大的挑战，定向抓取相关网页资源的“爬虫”也应运而生。爬虫不追求大的覆盖，而将目标定为抓取与某一特定主题内容相关的网页，为面向主题的用户查询准备数据资源。传统爬虫从一个初始网页的 URL 开始，获得初始网页上的 URL，在抓取网页的过程中，不断从当前页面抓取新的 URL 放入队列，直到满足系统的一定停止条件。爬虫程序主要有三个模块：

- 爬虫控制端：启动爬虫，停止爬虫，监视爬虫的运行情况
- 爬虫运行模块：包含三个小模块，URL 管理器、网页下载器、网页解析器
  - URL 管理器：对需要爬取的 URL 和已经爬取过的 URL 进行管理，可以从其中取出待爬取的 URL 传递给网页下载器。
  - 网页下载器：网页下载器将 URL 指定的网页下载下来，存储成一个字符串，传递给网页解析器。

  - 网页解析器：网页解析器解析传递的字符串，解析器不仅可以解析出需要爬取的数据，而且还可以解析出每一个网页指向其他网页的 URL，这些 URL 被解析出来会补充进 URL 管理器。

- 数据输出模块：存储爬取的数据

本课程实验借鉴其方法，对知乎中特定的问题（question）或者话题（topic）进行定向爬取。从知乎首页 (http://www.zhihu.com) 开始进入，输入关键词，以广度优先的方式，爬取相关联的网页显示的相关信息，包括问题本体、问题的提出者、浏览次数、点赞/差评次数、答案、答案的作者、答案的评论、答案获得的点赞/差评数，最后以 json 格式存取。

4 软件设计

4.1 整体框架设计

通过主程序调用知乎登陆函数，显示 GUI 界面，根据 GUI 界面所获取的搜索任务信息来爬取 URL，补充问题 URL 池，等待问题的 URL 爬取完毕，开始对每一个问题的 URL 爬取相应内容。
各模块之间的关系如下图所示：

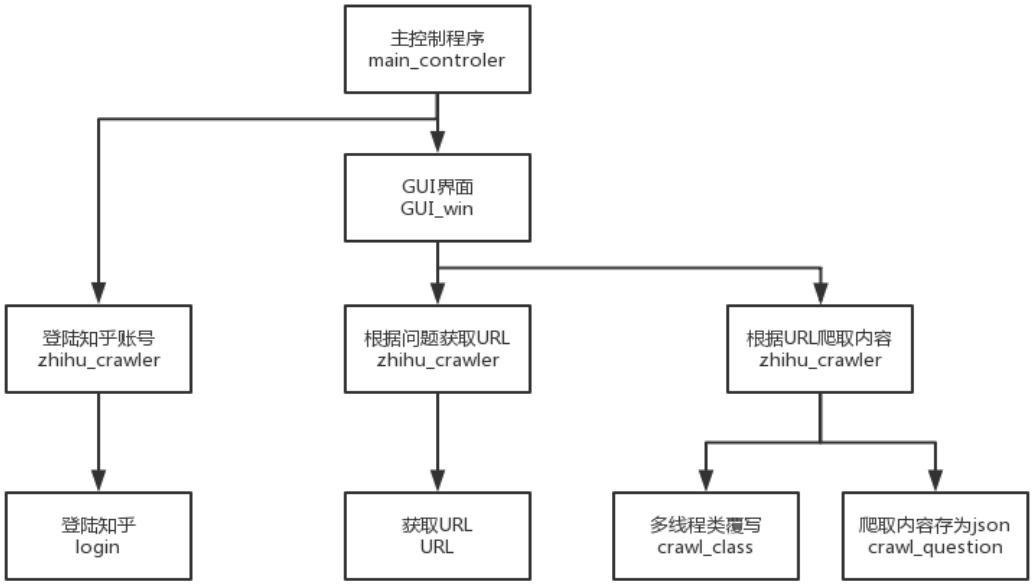


图 4.1: 各模块之间调用关系

程序运行的逻辑关系如下图所示：

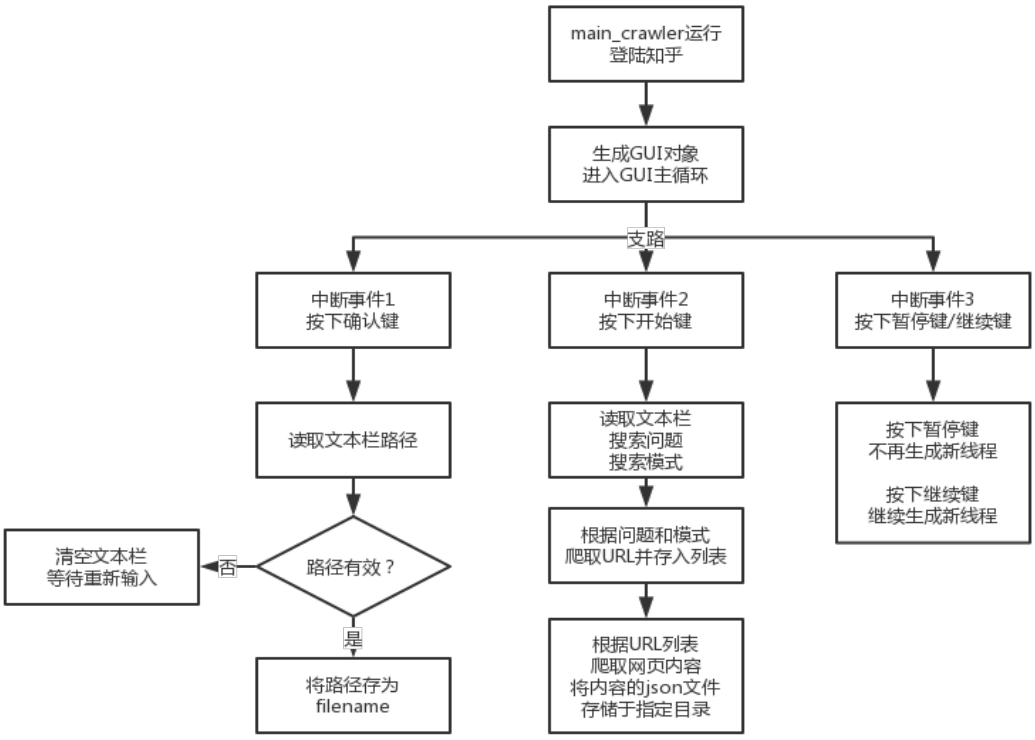


图 4.2: 程序逻辑关系

4.2 爬虫控制模块

4.2.1 wxpython GUI 设计

本程序使用 wxpython 实现 GUI 开发，同时使用基于 wxpython 的辅助平台 wxFormBuilder 进行 GUI 布局设计[1]。由于非正常操作可能导致程序报错，请按照以下顺序操作，确保程序的正确运行：

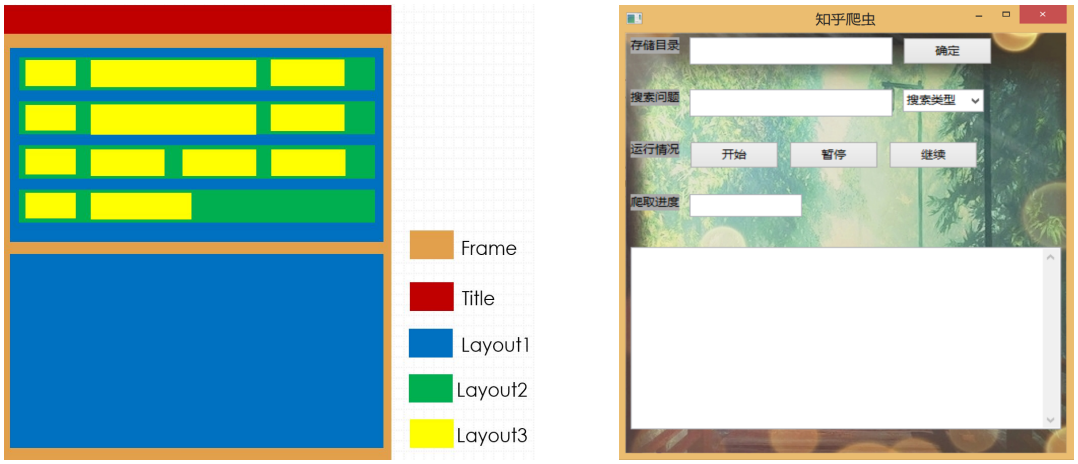
- 运行 main_controler.py，在 console 中登陆知乎，若登录成功，GUI 界面会弹出
- 输入 json 文件的存储目录，并按下确认键。如果该路径不存在，输入栏会清空，等待重新输入，如果路径存在，下方的信息显示栏会显示存储路径信息
- 输入搜索问题，并选择搜索类型是“综合问题”还是“话题”
- 按下开始键，程序开始运行。注意! 由于主机性能，在抓取 URL 阶段由于资源占用导致 CPU 满载，GUI 可能出现暂时的未响应状态，请等待 URL 爬取完毕，GUI 会恢复响应，并继续爬取具体内容并显示运行状态
- 在网页内容运行阶段，GUI 会显示当前爬取进去，下方信息显示栏会显示当前任务状态，总共 URL 数目和当前已进入运行线程的 URL，运行结束后会显示运行任务时间

- 在运行阶段中，若按下暂停键，程序将不再创建新的爬取线程，同时下方信息显示栏会提示任务暂停；按下继续键，程序继续执行，下方信息显示栏会提示任务继续

wxpython 是一个最成熟的跨平台 GUI 工具包，通过 wxpython 我们可以比较方便地创建一个 GUI 界面并设置 GUI 事件，实现我们对爬虫程序的图形界面控制。如图，我们建立了一个 Frame 类，并定义了 GUI 事件。该类会在一个 Frame 上设置 layout 并添加部件。GUI 界面如下图所示：

Class MyFrame1
Parent：wx.Frame
Method： __init__(self, parent)　　#初始化 GUI 界面 __del__(self)　　#按关闭键后关闭 GUI savePathButtonClick(self, event)　　#按确认键后确认 json 存储路径 startButtonClick(self, event)　　#按开始键后开始运行程序 pauseButtonClick(self, event)　　#按暂停键后停止线程生成 resumeButtonClick(self, event)　　#按继续键后继续线程生成 processText(self)　　#显示 URL 池爬取进度 consoleText(self ,text)　　#在下方的 TextCtrl 显示运行信息 runTask(self)　　#执行爬取任务

图 4.3: Frame 子类 MyFrame1 类设计



GUI 设计　　GUI 界面

图 4.4: GUI

如上，左图是 GUI 的 layout 设计，右图是完成的 GUI 界面。该 GUI 界面的设计思路及方法是：在一个 Frame 上设置 layout，在 layout 上可以叠加子 layout 或部件，layout 可以选择水平排布和垂直排布，layout 上的部件会按照设置自动排布。

在布局设计上，本工程使用了基于 wxpython 的平台 wxFormBuilder 进行辅助设计，通过该平台，可以很方便地实现部件的添加和事件的绑定，下图是该 GUI 设计的分级结构图：

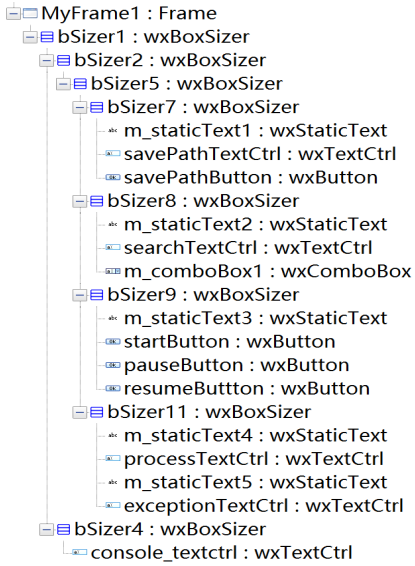


图 4.5: 分级结构图

在完成基础设计后，导出 python 代码，进行具体尺寸修改及 bitmap 背景设置，将爬虫控制程序绑定在 GUI 事件上，运行该类，即可实现爬虫程序的图形界面控制。

4.2.2 多线程实现

单线程爬取效率较低，对主机硬件资源的利用率较低，耗时也较长，因此，本工程使用 python 多线程库 threading 创建多线程任务提高爬取效率。

考虑到对于不同的搜索问题，本工程爬取的 URL 数不定，在创建新的爬取线程时我们直接创建匿名线程并启动。如下，调用 crawl_question 函数并传入参数创建新线程：

```
1 threading.Thread(target=cq.crawl_question,args=(url,filename,)).start()
```

但若直接使用 python 的 Thread 类创建新线程，会面临一个问题：一个异常的线程会直接停止，并打印 traceback 异常，但使用无法知道是那个 URL 线程出了问题。对此，我们建立 Thread 类的子类，对父类的 run 方法进行覆写：如上图，Crawler 类覆写了

图 4.6: Thread 子类 Crawler 类设计

父类的 ___init___ 方法和 run 方法，在 run 方法中调用 _run 方法运行线程创建的目标函数，若运行期间出现异常，则抛出异常并打印异常信息及异常 URL，代码如下：

```
import threading
import traceback
import sys
#Thread类的子类Crawler
class Crawler(threading.Thread):
    #class_lock = threading.Lock()

    def ___init___(self,funcName=None,args=()):
        threading.Thread.___init___(self)
        self.args = args
        self.funcName = funcName
        self.exitcode = 0
        self.exception = None
        self.exc_traceback = ''

    #覆写run方法
    def run(self):
        try:
            self._run()
        except Exception as e:
            self.exitcode = 1
            self.exception = e
            self.exc_traceback = ''.join(traceback.format_exception(*sys.exc_info()))
            print("URL Crawler Exception: ", self.args[0])
            print(" self.exc_traceback")

    def _run(self):
        try:
            self.funcName(self.args[0],self.args[1])
        except Exception as e:
            raise e
```

在上面的代码中，创建 Crawler 类，父类为 Thread 类，覆写 run 方法，执行目标函数，若运行中出现问题，线程抛出异常，并打印出错误 URL 信息及 traceback 信息。

将 Thread 类改为 Crawler 类，匿名线程创建如下：

```
def crawl_web_into_json(url,filename):
   crawl_class.Crawler(funcName=cq.crawl_question,args=(url,filename,)).start()
```

在 GUI 中，当程序开始后，通过执行 runTask 方法调用 crawl_web_into_json 创建爬取线程：

```
def runTask(self):
        lock = threading.Lock()
        start = time.clock()
        while self.url_list:
                if (len(threading.enumerate())<11)&self.thread_flag :
                        lock.acquire()
                        url = self.url_list.pop(0)
                        lock.release()
                        zhihu_crawler.crawl_web_into_json(url, self.filename)
        elapsed = (time.clock() - start)
```

在程序运行中，当 url_list 不为空，当程序总线程数小于 11（最大多线程数为 5）且程序未被暂停，url_list 弹出新 url，并调用 crawl_web_into_json 函数创建新线程。在 url_list 弹出新 url 过程中，为了防止资源竞争，对该步骤上锁，新线程创立后释放资源锁。

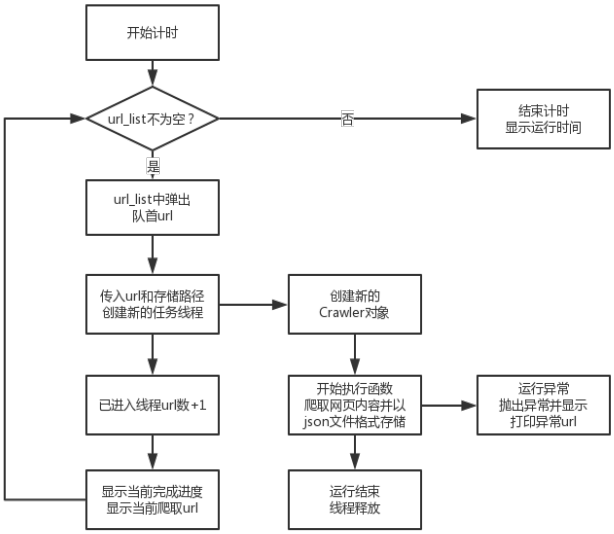


图 4.7: 线程创建流程图

经过测试，同一运行环境下，对于 80 条 URL，单线程运行接近 40 分钟，而多线程运行可以把运行时间缩短到 10 分钟左右，大大提高了运行效率。

4.3 爬虫运行模块

本模块主要实现了爬虫的基本运行，分为两个主要部分：

- 知乎登陆
- 建立问题 URL 库
  - 针对具体问题（Question）搜索知乎的“综合”模块
  - 针对具体话题（Topic）搜索知乎的“话题”模块

其中对问题（Question）的爬取采用 PhantomJS 库[2] 来实现动态加载，对话题（Topic）的爬取采用模拟翻页来实现动态加载。其主要原理框图如下图所示：

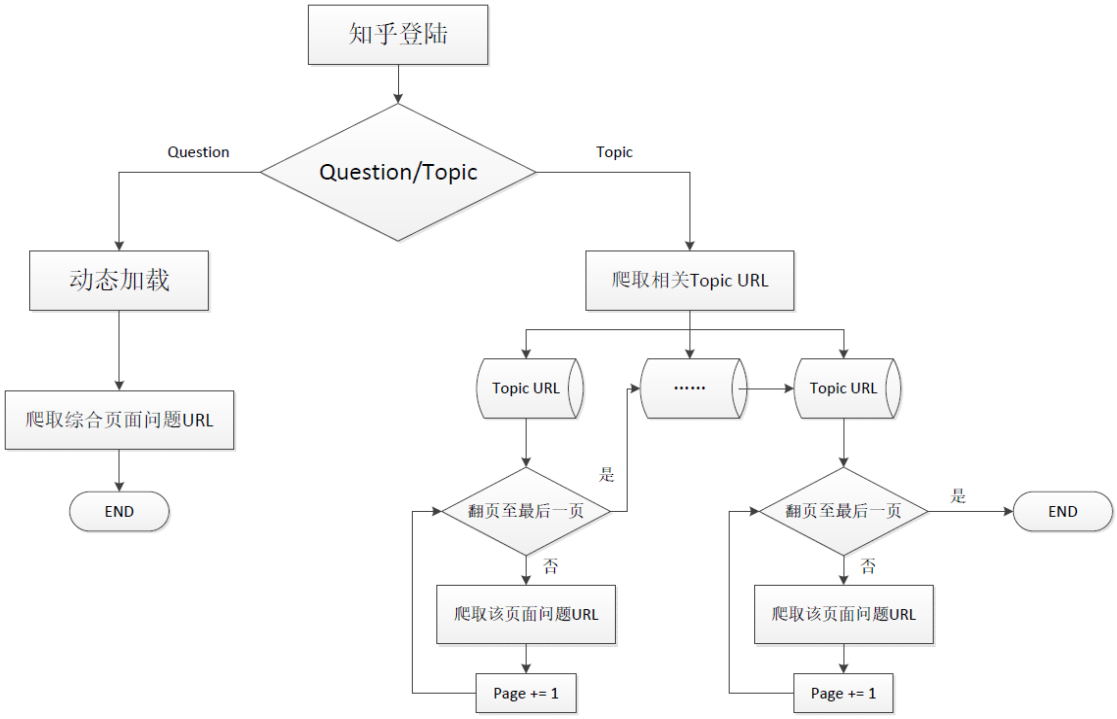


图 4.8: 爬虫运行框图

4.3.1 知乎登陆

知乎登陆是一个重要的环节，只有登录之后才可以捕捉到 topic 下面的所有页面信息。然而使用电脑浏览器登陆知乎需要点击倒立的文字，如果用程序模拟则十分不方便，速度也得不到保障；经尝试，我们可以使用手机的代理（User-Agent）来模拟登陆，此时 request 请求的头文件信息如下所示：

```
1 headers = {
2     "Host": "www.zhihu.com",
3     "Referer" : url,
4     'X-Requested-With' : 'XMLHttpRequest',
5     'User-Agent' : 'Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N)
6                     AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Mobile Safari/537.36'
7 }
```

这时我们就可以用验证码来登陆，首先获取验证码的链接，然后使用 request 库访问链接抓取验证码图片并显示出来，等待用户输入。我们获取验证码的函数如下所示：

```
session = requests.session()
def get_captcha():
    t = str(int(time.time()*1000))
    captcha_url = url + 'captcha.gif?r=' + t + "&type=login"
    r = session.get(url=captcha_url, headers=headers)
    with open('captcha.gif', 'wb') as f:
        f.write(r.content)
        f.close()
    im = Image.open('captcha.gif')
    im.show()
    im.close()
    captcha = input("please input the captcha:")
    return captcha
```

登陆函数如下所示，可以使用手机号码或者是邮箱通过模拟 post 请求来登陆，如果返回状态码为 200 则说明登陆成功，我们会把登陆的信息打印出来以便测试。首次完成登陆后会保存用户的 cookie，下次登陆可以直接加载 cookie 而不需要再次输入账号和验证码。

```
def login(secret, account):
    if re.match(r"^1\d{10}$", account):
        # login with phone number
        post_url = 'https://www.zhihu.com/login/phone_num'
        postdata = {
            'phone_num': account,
            'password' : secret
        }
    else:
        # login with email
        post_url = url + 'login/email'
        postdata = {
            'email' : account,
            'password': secret
        }
    # with captcha
    postdata["captcha"] = get_captcha()
    login_page = session.post(url=post_url, headers=headers, data=postdata)
    print('the status code returned by server:', login_page.status_code)
    if login_page.status_code != 200:
        print('login error!')
    else:
        print(login_page.json())
        while login_page.json()['r'] == 1:
            postdata["captcha"] = get_captcha()
            login_page = session.post(url=post_url, headers=headers, data=postdata)
            print('the status code returned by server:', login_page.status_code)
            print(login_page.json())

    session.cookies.save()
```

登陆函数测试结果如下：

```
C:\Users\dell-pc\Anaconda3\python.exe C:/Users/dell-pc/Desktop/zhihu_crawler_py3/login.py
Cookie cannot be load!
your name: None
please input your account:******
please input your secret:******
please input the captcha:f8dk
the status code returned by server: 200
{'r': 0, 'msg': '登录成功'}
your name: <span class="name">游浩然</span>
```

4.3.2 网页下载 & 解析

网页下载函数即为下代码所示，使用 requests 库来抓取指定 URL 的内容：

```
def getHTMLText(url):
    try:
        r = requests.get(url, headers=headers)
        # r.status_code  200
        # r.encoding  'utf-8'
        return r.content
    except:
        return "Error"
```

网页解析则依赖于 BeautifulSoup 库，比如下代码中 demo 即为爬取的网页内容，通过 BeautifulSoup 解析过后以网页标签来爬取相应问题的 URL。

```
demo = demo_temp.pop()
soup = BeautifulSoup(demo, 'html.parser')
for meta in soup.find_all('meta', attrs={'itemprop': 'url'}):
    url_temp = meta.get('content')
    url_list.append(url_temp)
```

4.3.3 URL 爬取 & 管理

- 对知乎“综合”模块进行爬取
  采用 PhantomJS 来模拟对页面的下拉操作来实现动态加载，之后获取该 URL 的 HTML 页面，使用 BeautifulSoup 来解析，利用网页中的 <meta> 标签的 content 属性来抓取页面上的所有问题链接，补充入地址池。

```
def get_content(url, url_list):
    driver = webdriver.PhantomJS(executable_path=r"E:/phantomjs/bin/phantomjs.exe")
    driver.implicitly_wait(10)
    driver.maximize_window()
    driver.get(url)

    demo_temp = []
    time.sleep(1)
    while True:
        if len(demo_temp) > 10:
            demo_temp = demo_temp[-10:]
            if demo_temp[0] == demo_temp[9]:
                break
        driver.execute_script('window.scrollTo(0,document.body.scrollHeight)')
        time.sleep(1)
        demo_temp.append(driver.page_source)

```

```
    demo = demo_temp.pop()
    soup = BeautifulSoup(demo, 'html.parser')
    for meta in soup.find_all('meta', attrs={'itemprop': 'url'}):
        url_temp = meta.get('content')
        url_list.append(url_temp)
    return url_list

    if DType == 'question':
    # get a question url
    content = {'' : content}
    url = "https://www.zhihu.com/search?type=content&q" + urllib.parse.urlencode(content)
    # search url in this page
    url_list = URL.get_content(url, [])
```

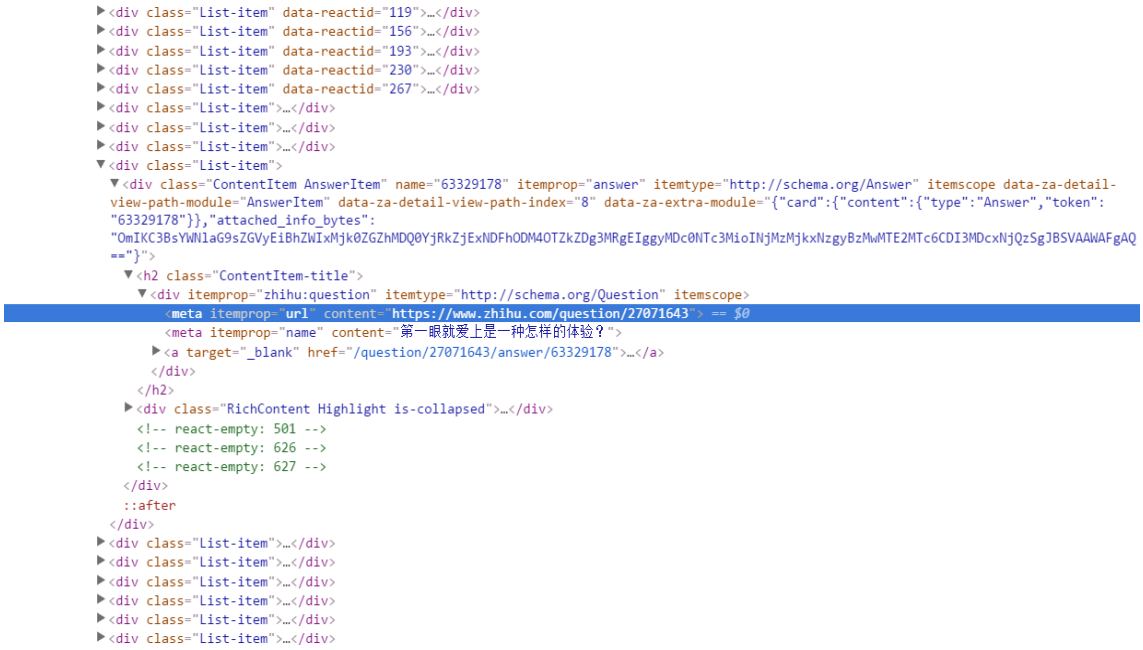


图 4.9: HTML 文件中的 <meta> 标签

- 对知乎“话题”模块进行爬取
  首先由种子 URL 开始，爬取相关 topic 的 URL，如下函数：

```
def get_topic_id(url, url_list):
    demo = getHTMLText(url)
    soup = BeautifulSoup(demo, "html.parser")
    for link in soup.find_all('a'):
        url_temp = link.get('href')
        if not url_temp:
            pass
        elif '/topic' in url_temp and not 'http' in url_temp:
            topic_id = url_temp[7:15]
            url_list.append('https://www.zhihu.com/topic/' + topic_id)
    return url_list
```

针对前述的每一个 URL，爬取该 topic 下的所有问题，只需要解析出其问题的总页数，通过 page+=1 的操作模拟翻页过程，爬取该 topic 下面的所有问题的 URL 补充进地址池。

图 4.10: Topic 下所有问题页数

```
def get_question_id(url, url_list):
    page_url = url + '/questions?page=1'
    demo = getHTMLText(page_url)
    soup = BeautifulSoup(demo, "html.parser")
    # get total page
    num_page = 1
    for link in soup.find_all('a'):
        url_temp = link.get('href')
        if '?page=' in url_temp:
            num_page = max(num_page, int(url_temp[6:len(url_temp)]))
    print('the number of page in topic ' + url[28:len(url)] + ' is : ' + str(num_page))
    # get all questions under topic
    for page in range(1, num_page+1):
        page_url = url + '/questions?page=' + str(page)
        demo = getHTMLText(page_url)
        soup = BeautifulSoup(demo, "html.parser")
        for link in soup.find_all('a'):
            url_temp = link.get('href')
            if '/question' in url_temp and not 'http' in url_temp:
                question_id = url_temp[10:19]
                url_list.append('https://www.zhihu.com/question/' + question_id)
    return url_list
```

对知乎“话题”模块所有问题 URL 进行爬取并管理的代码如下所示：

```
    if DType == 'topic':
    # get a topic url
    url = "https://www.zhihu.com/search?type=topic&q=" + content

    # search url in this page
    url_temp = URL.get_topic_id(url, [])
    if len(url_temp) > 0:
        url_list = url_list + url_temp

    # duplicate removal
    url_list = list(set(url_list))

    # search url based on BFS
    if len(url_list) > 0:
        url = url_list[0]
    while '/topic' in url and len(url_list) > 0:
        url = url_list.pop(0)
        url_temp = URL.get_question_id(url, [])
        if len(url_temp) > 0:
            url_list = url_list + url_temp

    # duplicate removal
    url_list = list(set(url_list))
    return url_list
```

4.4 数据输出模块

4.4.1 Question 页面动态加载

首先，由于知乎问题页面采用了动态加载技术，直接通过 url 获取的网页内容只有一小部分，其次，虽然通过浏览器请求可以得到回答的 json 数据，但由于问题的完整描述需要执行点击操作，故最终考虑使用 selenium 模拟浏览器操作，在实际爬取中，我们采用了 PhantomJS（基于 webkit 的没有界面的浏览器），相较于 Chrome 和 FireFox 有更快的速度。

实现难点: 由于 selenium 是通过模拟浏览器来获取网页内容，所以存在一个加载以及渲染过程，也就是说在执行每步操作后，需要让程序等待一段时间。另外在回答加载上，需要不断将鼠标下滑到页面底部，由于存在加载的时间，故要控制该行为的频率，以免造成过多的无效操作。

代码实现:

```
# 点击操作
try:
    click_btn = driver.find_element_by_xpath('//button[@class="Button QuestionRichText-more Button--plain"]')
    ActionChains(driver).click(click_btn).perform()
    time.sleep(0.5)
    html_text = driver.page_source
except:
    html_text = driver.page_source
# 下滑操作
while(True):
    driver.execute_script('window.scrollTo(0,document.body.scrollHeight)')
    time.sleep(0.8)
    if(html_text == driver.page_source):
        break
    html_text = driver.page_source
```

4.4.2 Question 网页解析

网页解析要求从 HTML 中提取所需的信息，我们选取的工具是 beautifulsoup4，相较于采用 xpath，css 以及正则表达式，更加方便简洁。

实现难点: 需要通过浏览器检查元素来确定元素的位置，以及如何利用条件来提取该位置的元素。

代码实现:

```
# 获取问题标题
question_title = soup.select_one('h1[class="QuestionHeader-title"]').get_text()
question_dict['question_title'] = question_title

# 获取回答
nodes = soup.find_all('div', class_="List-item")
```

4.4.3 Question 数据存储

由于 json 能较好地的反映数据的从属关系，并且更加易读，所以我们决定将爬取的数据保存成 json 格式[3]。

实现难点:Python 对于中文字符的支持不够好，在保存时需要转换成 unicode 编码。

- 代码实现：

```
    with open(file_name, 'wb') as json_file:
        for info in info_list:
            json_file.write(json.dumps(info,ensure_ascii=False).encode('utf-8'))
            json_file.write('\n'.encode('utf-8'))
```

效果展示：

- 问题信息

```
{
  "question_followers": "15,436",
  "question_title": "如何在四小时内学会用 Ai 做 UI？ ",
  "answer_number": "86 个回答",
  "question_text": "整个问题的初衷在于看到了@黎敏 的回答，产品经理新人是否有必要学习Photoshop？ 我这个问题主要在于希望一些新人能速度的进入AI操作，而不必花时间去找教程，需要这个问题回答的人不少。再次谢谢 @黎敏。",
  "question_commet": "15 条评论",
  "brower_number": "989,872",
  "question_id": "21378038"
}
```

- 回答信息

```
{
  "answer_votes": "0",
  "answer_author": "mac mico",
  "answer_text": "工欲善其事必先利其器，效率一定是首要的。UI设计之效率为王 - 设计与开发之效率 - 知乎专栏",
  "answer_id": 61575458,
  "answer_comment": "添加评论"
}
```

4.5 爬虫测试效果

- 爬虫正常运行!

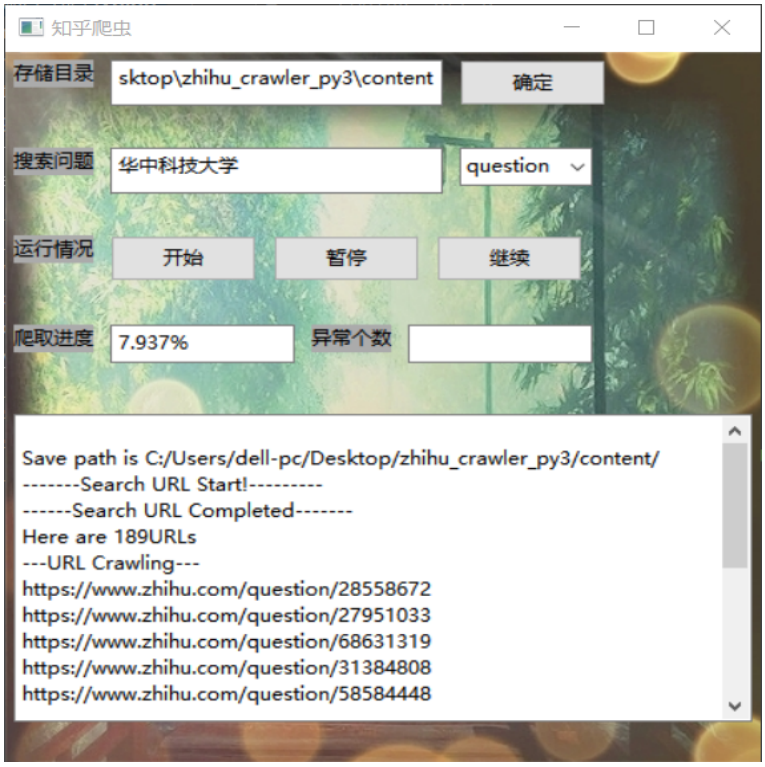


图 4.11: 爬取程序运行效果

5 项目分工

姓名	负责模块	主要任务
黎张帆	爬虫控制器	启动、停止、监视爬虫的运行情况；实现多线程爬取。GUI 设计
游浩然	爬虫运行模块	知乎登陆；URL 管理器、网页下载器、网页解析器；动态加载
郭金城	数据输出模块	通过网页标签抓取问题 URL 下的相关信息；动态加载

6 设计总结

通过这次软件课程设计，我更好地理解了爬虫运行的原理，包括模拟登陆、网页下载、网页解析以及动态加载，更好地了解了 Python 中的 requests、BeautifulSoup、re、http 等等爬虫常用的库。

在实验过程中也遇到了许多问题，比如知乎登陆过程中如果使用 PC 端的 User-Agent 则会有点击倒立的验证码的操作，影响实验效率，我为此专门尝试其他浏览器代理登陆，发现手机端登陆只需要输入验证码即可，节约了时间；还比如在处理搜索“question”时综合页面动态加载的过程中，开始的时候积极尝试各种方法，比如通过浏览器抓包获得页面数 total 和当前页面 page 的内容以及下一页面的 url 链接，这样就可以更快地去抓取所有页面，但是由于知乎的反爬机制，使用 requests 库不能获取到服务器内部信息，最后选择了使用 selenium 来模拟浏览器下滑的过程实现动态加载；但在处理搜索“topic”时爬取全部问题时，网页是存在具体的页面的，这时只需要翻页即可！实现动态加载的过程也教会了我“分而治之”的思想，对于具体的问题要有相对应的解决方法，不能拘泥于已有的了解一概而论。

亲自动手实践让我巩固了 python 的基础应用、掌握了爬虫的工作模式以及领会到“分而治之”的灵活思想，受益匪浅！

参考文献

[1] [美]Mark Summerfield. Python3 程序开发指南. 人民邮电出版社, 北京, 2 2015.

[2] 班歌静听. Phantomjs 中文文档. https://www.cnblogs.com/bangejingting/p/6907628.html, 5 2017. [Online;accessed 13-Dec-2017].

[3] W3school. Json 教程. http://www.w3school.com.cn/json/. [Online;accessed 13-Dec-2017].