大连工业大学

实 验 报 告

题 目： 使用 Python 实现网络爬虫算法

专业： 计算机科学与技术

班级学号： 计算机 182-1805010207

学生姓名： 梁宇龙

指导教师： 任树华

实验室名称： 综 B410A

所属学院： 信息科学与工程学院

实验日期： 2021 年 6 月 27 日

实 验 内 容

实验项目 3

使用 Python 实现网络爬虫算法（Implementation of web crawler algorithm with Python）

1、实验目的

1）强化 Python 程序的设计和编程能力
2）学习网络爬虫算法的原理
3）学习使用 Python 语言实现网络爬虫算法

2、实验内容

1）理解网络爬虫算法的原理，并设计使用 Python 语言获取网页数据的程序。
2）用 Python 语言中的 threading 和 GetUrl 模块对网站中 URL 进行搜集。

3、实验原理

1）爬虫算法原理：

网络爬虫（又被称为网页蜘蛛，网络机器人，网页追逐者），是一种按照一定的规则， 自动的抓取万维网信息的程序或者脚本。很多搜索引擎都使用爬虫提供最新的数据，搜索引擎利用用户访问过页面的一个副本进行索引，以提供快速的访问。网络爬虫也可以在 web 上用来自动执行一些任务，例如检查链接，确认 html 代码；也可以用来抓取网页上某种特定类 型信息，例如抓取电子邮件地址。

本实验中使用的网络爬虫算法是广度优先搜索(BFS)。广度优先搜索策略是指在抓取过程中，在完成当前层次的搜索后，才进行下一层次的搜索。有很多研究将广度优先搜索策略应用于聚焦爬虫中。其基本思想是认为与初始 URL 在一定链接距离内的网页具有主题相关性的概率很大。另外一种应用是将广度优先搜索与网页过滤技术结合使用，先用广度优先策略抓取网页，再将其中无关的网页过滤掉。这些方法的缺点在于，随着抓取网页的增多，大量的无关网页将被下载并过滤，算法的效率将变低。

2）Python 语句在某一个网页上获取数据时，首先要分析网页的 HTML 源代码，我们以淘宝网页中的商品分类(http://list.taobao.com/browse/cat-0.htm)为例，获取所有一级类别和二级类别的标题。

本实验中要调用 urllib2 和 sgmllib.SGMLParser 模块，并使用语句 content = urllib2.urlopen('网页 URL').read()来获取网页的内容。并使用 list.feed(content)语句将网页内容存入列表，进行处理。

此实验部分实现代码如下

```
class ListName1(SGMLParser):
    def __init__(self):
        SGMLParser.__init__(self)
        self.is_h4 = ""
```

```
            self.name = []
        def start_h4(self, attrs):
            self.is_h4 = 1
        def end_h4(self):
            self.is_h4 = ""
        def handle_data(self, text):
            if self.is_h4 == 1:
                self.name.append(text)
                class ListName2(SGMLParser):
        def __init__(self):
            SGMLParser.__init__(self)
            self.is_h5 = ""
            self.name = []
        def start_h5(self, attrs):
            self.is_h5 = 1
        def end_h5(self):
            self.is_h5 = ""
        def handle_data(self, text):
            if self.is_h5 == 1:
            self.name.append(text)
    content = urllib2.urlopen('http://list.taobao.com/browse/cat-0.htm').read()
    listname1 = ListName1()
    listname2 = ListName2()
    listname1.feed(content)
    listname2.feed(content)
```

在该实验中，学生需用前述的爬虫算法实现语句实现写入读取淘宝网页商品分类标题的功能，并在此基础上，思考如何实现读取其他网页数据的方法，记录 Python 代码，并分析实验结果。

3）在获取网站上的 URL 时，需要调用 GetUrl 模块。本实验中通过调用 threading 模块采用多线程算法实现网站 URL 的获取。

此实验部分实现代码如下

```
    def Craw(self,entryUrl): #这是一个深度搜索，到 g_toDlUrl 为空时结束
        g_toDlUrl.append(entryUrl)
        self.logfile.write('>>>Entry:\n')
        self.logfile.write(entryUrl)
        depth = 0
        while len(g_toDlUrl) != 0 and depth <= self.Maxdepth:
            depth += 1
            print 'Searching depth ',depth,'...\n\n'
            self.downloadAll()
            self.updateToDl()
            content = '\n>>>Depth ' + str(depth)+':\n'
            self.logfile.write(content)
            i = 0
            while i < len(g_toDlUrl):
                content = str(g_totalcount + i + 1) + '->' + g_toDlUrl[i] + '\n'
```

```
                self.logfile.write(content)
                i += 1
class CrawlerThread(threading.Thread):
    def __init__(self, url, fileName):
        threading.Thread.__init__(self)
        self.url = url #本线程下载的 url
        self.fileName = fileName
def run(self): #线程工作-->下载 html 页面
    global g_mutex
    global g_failedUrl
    global g_dledUrl
    try:
        f = urllib.urlopen(self.url)
        s = f.read()
        fout = file(self.fileName, 'w')
        fout.write(s)
        fout.close()
    except:
        g_mutex.acquire() #线程锁-->锁上
        g_dledUrl.append(self.url)
        g_failedUrl.append(self.url)
        g_mutex.release() #线程锁-->释放
        print 'Failed downloading and saving',self.url
        return None #记着返回!
g_mutex.acquire() #线程锁-->锁上
g_pages.append(s)
g_dledUrl.append(self.url)
g_mutex.release() #线程锁-->释放
```

在该实验中，学生需用上述网络爬虫算法和多线程控制语句实现获取某一网站所有 URL 的程序，并在此基础上，比较并分析采用不同线程数时算法的性能。记录 Python 代码，并分析实验结果。

4、实验步骤

1）设计某一个网页上获取数据的程序

- 分析实验要求
- 打印网页上获取的数据
- 记录程序代码
- 记录并分析实验结果

2）设计多线程的获取网站 URL 的程序

- 分析实验要求
- 打印网站上相关的 URL
- 比较不同线程数的算法性能
- 记录程序代码
- 记录并分析实验结果

5、实验记录

（一）设计某一网页上获取数据的程序

- 算法设计

（1）通过网页 URL，获取该网页的 html 文件。本部分使用 requests 模块，使用该模块 get 方法，将网页 URL 和回传最慢延时作为参数进行网页请求调用，并将调用回的结果设置为 utf-8 格式，并将网页内容返回给参数。

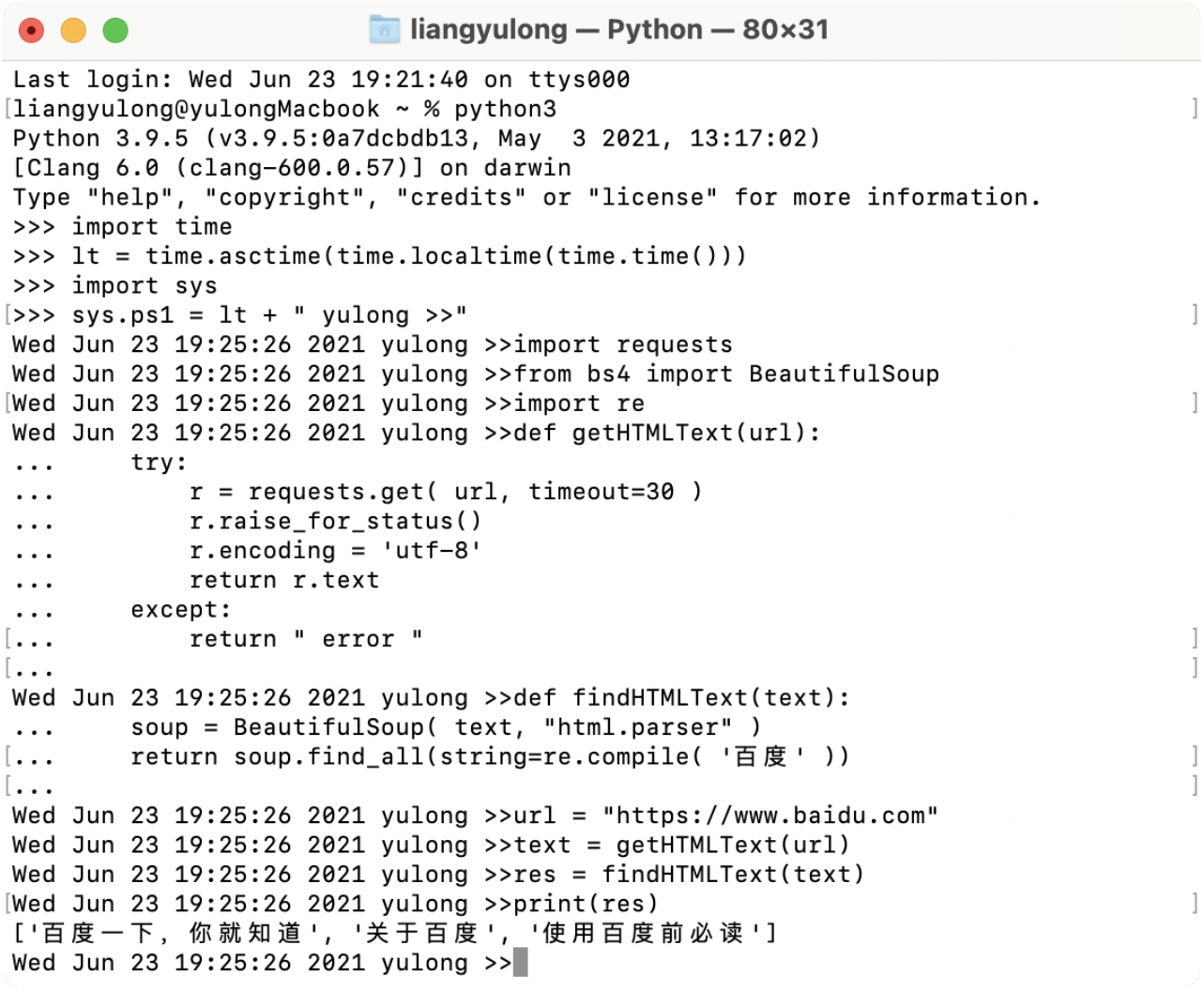
由于此部分涉及网页请求，因此要设置抛出异常的情况，输出 error 退出函数。

（2）获取网页 html 信息后，对网页进行格式化处理，并按照关键字进行检索。本部分使用 beautifulsoup 模块，在获取 html 模块以后，可以通过 soup 中参数“html.parser”进行网页文件格式化处理。

（3）而后通过该模块方法 find_all 可以进行关键字内容检索，方法采用正则表达式 re 模块的 compile 方法对关键字内容进行检索。

（4）将检索后的内容打印输出，验证设计。

- 程序与结果记录



- 结果分析

实验中通过抓取百度首页中关于百度为关键字的信息，对网页进行爬取。检查爬取结果，均含有百度关键字。此外，检查 html 网页，其余项均不包含百度字样，程序执行结果正确。

（二）设计多线程的获取网站 URL 的程序

- 算法设计

Python 标准库模块 threading 提供了相关的操作。通过创建 threading，Thread 对象实例可以创建线程；通过调用 Thread 对象的 start 方法可以启动线程，也可以创建 Thread 派生类，重写 run()方法，然后通过创建实例来创建线程。

本次实验中，将所有要爬取的网页 url 以列表形式进行存放，并调用爬取函数。在函数中，每调用一次要创建一个新的线程，该线程用于对对应网页进行爬取。爬取结束后线程关闭，当所有线程都关闭的情况下调用返回，打印输出结果。

- 程序记录

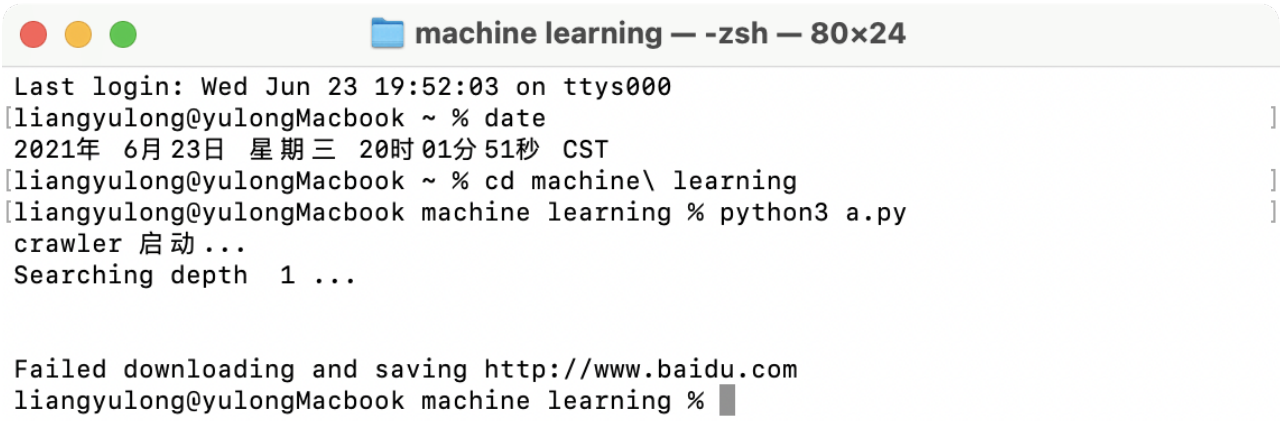
```
import threading
import urllib
import re
import time
g_mutex = threading.Condition()
g_pages = []  # 从中解析所有 url 链接
g_queueURL = []  # 等待爬取的 url 链接列表
g_existURL = []  # 已经爬取过的 url 链接列表
g_failedURL = []  # 下载失败的 url 链接列表
g_totalcount = 0  # 下载过的页面数

class Crawler:
    def __init__(self, crawlername, url, threadnum):
        self.crawlername = crawlername
        self.url = url
        self.threadnum = threadnum
        self.threadpool = []
    def craw(self):
        global g_queueURL
        g_queueURL.append(url)
        depth = 0
        print(self.crawlername+" 启动...")
        while(len(g_queueURL) != 0):
            depth += 1
            print('Searching depth ', depth, '...\n\n')
            self.downloadAll()
            self.updateQueueURL()
```

```
            content = '\n>>>Depth '+str(depth)+':\n'
            i = 0
            while i < len(g_queueURL):
                content =
str(g_totalcount+i)+'->'+g_queueURL[i]+'\n'
                self.logfile.write(content)
                i += 1

    def downloadAll(self):
        global g_queueURL
        global g_totalcount
        i = 0
        while i < len(g_queueURL):
            j = 0
            while j < self.threadnum and i+j < len(g_queueURL):
                g_totalcount += 1
                threadresult = self.download(
                    g_queueURL[i+j], str(g_totalcount)+'.html', j)
                if threadresult != None:
                    print('Thread started:', i+j,
                          '--File number =', g_totalcount)
                j += 1
            i += j
            for thread in self.threadpool:
                thread.join(30)
            threadpool = []
        g_queueURL = []

    def download(self, url, filename, tid):
        crawthread = CrawlerThread(url, filename, tid)
        self.threadpool.append(crawthread)
        crawthread.start()

    def updateQueueURL(self):
        global g_queueURL
        global g_existURL
        newUrlList = []
        for content in g_pages:
            newUrlList += self.getUrl(content)
        g_queueURL = list(set(newUrlList)-set(g_existURL))

    def getUrl(self, content):
```

```
        reg = r'"(http://.+?)"'
        regob = re.compile(reg, re.DOTALL)
        urllist = regob.findall(content)
        return urllist


class CrawlerThread(threading.Thread):
    def __init__(self, url, filename, tid):
        threading.Thread.__init__(self)
        self.url = url
        self.filename = filename
        self.tid = tid

    def run(self):
        global g_mutex
        global g_failedURL
        global g_queueURL
        try:
            page = urllib.urlopen(self.url)
            html = page.read()
            fout = file(self.filename, 'w')
            fout.write(html)
            fout.close()
        except Exception:
            g_mutex.acquire()
            g_existURL.append(self.url)
            g_failedURL.append(self.url)
            g_mutex.release()
            print('Failed downloading and saving', self.url)
            return None
        g_mutex.acquire()
        g_pages.append(html)
        g_existURL.append(self.url)
        g_mutex.release()


if __name__ == "__main__":
    g_queueURL = ['http://www.baidu.com', 'http://www.qq.com',
           'http://www.taobao.com', 'http://www.sina.com.cn']
    threadnum = 10
    crawlername = "crawler"
    for i in range(len(list)):
```

```
            crawler = Crawler(crawlername, g_queueURL[i], threadnum)
            crawler.craw()
```

- 结果记录



- 结果分析

通过将要爬取的网页以列表形式写入 url，在多次调用 crawler 类进行网页爬取时候触发多线程机制，促使在爬取多个网页时候可以进行并发执行。

6、实验体会

本次实验通过使用网络爬虫技术和多线程技术，进一步了解了 python 编程语言在数据获取和数据分析领域的作用，也同时了解了 python 第三方模块库对 python 的重要性。

同时本次实验的难度较前两次变得更大，其主要原因就是爬取网页的不规范性导致爬取难度变大，如例程中判断标签为 h4 或 h5 属性的方法在大部分网页不适用。另一个原因就是模块库的版本问题，由于 python3 和 python2 的系统自带模块库差异，导致有多种模块不可以同时使用，加大了开发难度。在接下来学习中，在进行实验之前会对其进行系统学习和开发文档阅读，避免类似情况发生。