

实验报告2 爬虫1

518030910303 纪喆

2019 年 9 月 23 日

1 实验准备

1.1 实验环境

本次实验在 **Ubuntu 14.04 LTS**系统中进行，使用的是**Python3**编程语言。本次实验主要用到的工具有：

- vim
- Python3

本次实验使用的库有：

- sys
- urllib
- BeautifulSoup4
- requests
- re
- os

1.2 实验目的

1. 使用程序模拟登陆交大BBS网站，并且修改个人信息
2. 将网络爬虫的工作过程抽象化，在较为简单的情况下实现BFS搜索
3. 修改代码，实现网络关系图的构建
4. 完善第三个代码文件，实现网络爬虫

1.3 实验原理

1.3.1 练习一：模拟登录，修改个人说明档

登陆网页的原理 为了成功模拟登陆网页，首先需要理解现实生活中用浏览器登陆网站的原理。首先，使用GET方法获得网站首页，在用户填写用户名和密码后，将表单以POST方式提交到服务器，服务器验证登陆后，则会与客户端建立一个会话，而唯一标识这个会话的ID会以cookie的形式储存在浏览器中。浏览器在后续的发送请求时会自动携带cookie发送，从而使服务器得以识别会话，进行用户个人的操作。

程序模拟 本实验中，使用了 *requests* 库中的 *session* 类，用来模拟会话。*session* 类的实体创建后，会自动保存服务器返回的各类信息，包括cookie。因此，在后续的访问中使用该实体访问即可模拟登陆后的过程。

修改个人说明档 在实现登陆后，根据在浏览器中获得的信息，用程序仿造相同内容，以POST的方式发送给服务器，从而达到修改目的。

1.3.2 练习二：完成BFS搜索

为实现广度优先搜索，可以让待爬取URL按照队列的模式“入列”和“出列”。只要爬取的时候遵循队列的先入先出原则，就可以实现广度优先的要求。

1.3.3 练习三：完善函数，返回图的结构

按照个人对代码示例的理解，此函数返回图的结构，意在生成网络拓扑结构，对网络有更深理解。其实现较为简单，只需要在最一开始创立一个空字典，每次获取页面URL时向字典中增加一个记录即可。

1.3.4 练习四：完成网页爬虫

在练习三构建的模型基础上，加入实际的信息网络内容，用实际网络代替图模型，用实际发送HTTP请求代替字典取运算等，即可使爬虫实战。

2 实验过程

本部分将以文件（而非练习题目）为单位，采用代码+解释的方式展示本实验。

2.1 文件一：bbs_set

```
id = sys.argv[1]
pw = sys.argv[2]
text = sys.argv[3].encode('gbk')
```

首先，借助`sys.argv` 获取从命令行输入的三个参数，分别为用户名、密码、个人说明档更新内容，传入`bbs_set`函数。其中，对于第三个参数，因为内容有可能为中文，为了避免乱码，本实验使用`GBK`重新编码在传入。

```
bbs_session = requests.session()
bbs_session.post("https://bbs.sjtu.edu.cn/bbslogin",{ 'id':
                                                    id, 'pw':pw, 'submit':
                                                    login'})
bbs_session.post("https://bbs.sjtu.edu.cn/bbsplan",{ 'type':
                                                    update', 'text':text})
```

函数首先创建了一个`session`对象，该对象可以看作是程序与BBS服务器建立的一个临时会话。函数第二行实现了登陆操作，以POST方式将用户名和密码发送给服务器，服务器返回的相关内容则会被储存在 `bbs_session` 对象中，因此在第三行修改个人说明档的时候，由于请求中会带有`session`的`cookie`内容，服务器可以顺利识别用户，修改内容。

值得一提的是，两次POST提交的数据格式，全部来自从浏览器的检查。具体如图。信息在图片左下角的Form Data中。

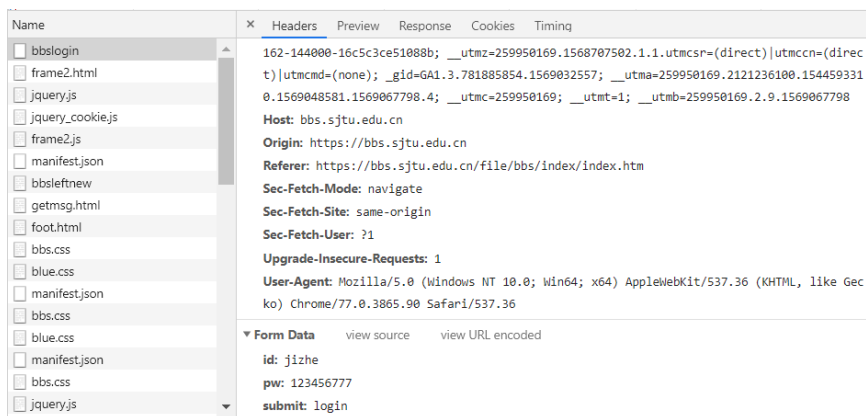


图 1: 点击登陆后的Network情况

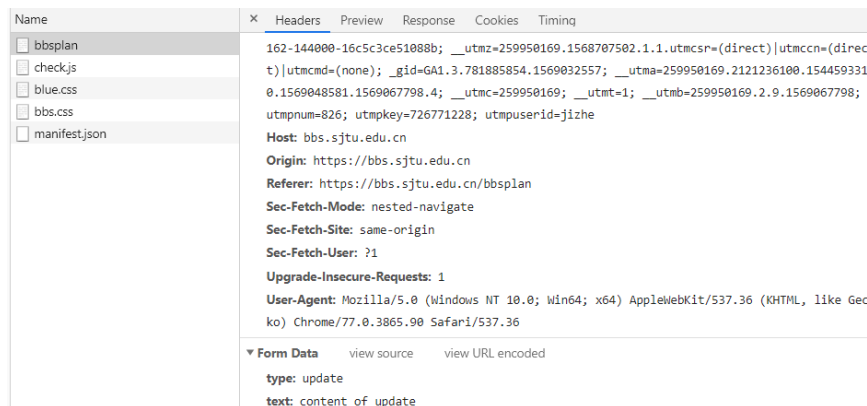


图 2: 点击个人说明档中的存盘后

下面的这三行程序是为了验证修改个人说明档是否成功。在重新获取修改个人说明档的页面后，检索“textarea”部分，并将个人档内容打印到屏幕上。

```
content = bbs_session.get('https://bbs.sjtu.edu.cn/bbsplan')
soup = BeautifulSoup(content, 'html.parser')
print(soup.find('textarea').string.strip())
```

运行命令及结果如图。

```
(base) jizhe@ubuntu:~/Desktop$ python3 bbs_set.py jizhe 123456777 hello,SH
hello,SH
```

图 3: 文件一运行结果1

注：因为实验环境的系统中没有安装中文输入法，因此无法演示中文。但代码本身支持中文输入，为说明此点，这里展示一张在Windows10系统中的运行结果

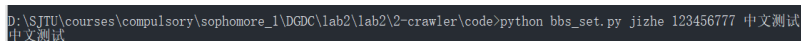
A terminal window with a dark background. The command prompt shows the file path 'D:\SJTU\courses\compulsory\sophomore_1\DGDC\lab2\lab2\2-crawler\code>' followed by the command 'python bbs_set.py jizhe 123456777 中文测试'. The output of the script is '中文测试'.

图 4: 文件一运行结果2

2.2 文件二：crawler_sample

本文件中人为设定一个字典来储存图的结构，以此模拟网络结构。定义过程如下：

```
g = {'A': ['B', 'C', 'D'], \
     'B': ['E', 'F'], \
     'D': ['G', 'H'], \
     'E': ['I', 'J'], \
     'G': ['K', 'L']}
```

与此字典模型相对应，获取网页内容的功能被模型化成字典操作中的`get`。具体代码为

```
def get_page(page):
    return g.get(page, [])
```

而由于此模型中“网页”的内容就是所有的“URL”，因此提取URL的函数直接返回全部内容即可，也就是：

```
def get_all_links(content):
    return content
```

没有因为模型而简化功能的两个函数是`union_dfs`与`union_bfs`，这两个函数完成的的就是检查网址是否已被爬取，别难过且根据爬取顺序确定待爬取URL的插入方式。具体地，由于在爬取时每次使用`list.pop`弹出列表最后一个元素，因此在找到未爬取的网址后：在广度优先方式中，`union_bfs`将新发现的URL插入列表头部，配合爬取函数完成的是队列操作，先入先出；深度优先方式中，`union_dfs`将新发现的URL插入列表的尾部，配合爬取函数完成的是栈操作，先入后出。两个函数代码如下：

```
def union_dfs(a,b):
    for e in b:
```

```

        if e not in a:
            a.append(e)

def union_bfs(a,b):
    for e in b:
        if e not in a:
            a.insert(0,e)

```

辅助函数就是以上这些，下面介绍爬取函数代码：

```

def crawl(seed, method):
    tocrawl = [seed]
    crawled = []
    graph = {}
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
            content = get_page(page)
            outlinks = get_all_links(content)
            graph[page] = outlinks
            globals()['union_' + method](tocrawl, outlinks)
            crawled.append(page)
    return graph, crawled

```

函数接口： 爬取函数接受两个参数，第一个为爬取种子，第二个有 *dfs* 与 *bfs* 两种选择，分别对应深度优先与广度优先。其返回值为一个元组，前一个元素为“网络”的结构，类型为一个字典，后一个元素是已爬取的“网页”。

核心变量：

tocrawl 类型为列表，储存待爬取网页连接，需要时每次弹出列表最后一个连接。

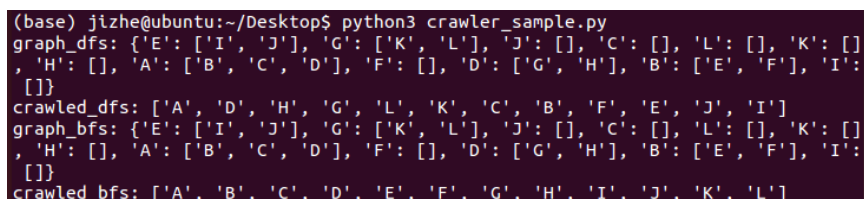
crawled 类型为列表，储存爬取过的网页链接。其目的为避免重复爬取相同网页浪费资源。

graph 类型为字典，储存各网页之间的联系，及它们的图结构。

工作流程： 函数通过`while`循环，持续工作直到待爬取网页连接为空。每次爬取从`tocrawl`中弹出最后一个链接，判断是否已爬取。若未曾被爬取，则进入爬取工作。先通过`get_page`获得网页内容，然后用`get_all_links`获取网页内容中所有的网络链接，在变量`graph`中新增一个键值对，储存该网页所提及的所有链接，根据参数`method`确认使用哪一种`union`函数，将新出现的链接加入到`tocrawl`中。爬取后将本网页链接加入到`crawled`中，避免后续重复爬取。

本文件的最后几行代码为测试代码。以`g`图中的`A`为种子，进行对模型`g`的爬取，分别使用了深度优先与广度优先的方式，并将函数的两个返回值分别打印出来。

运行结果如下：



```
(base) jizhe@ubuntu:~/Desktop$ python3 crawler_sample.py
graph_dfs: {'E': ['I', 'J'], 'G': ['K', 'L'], 'J': [], 'C': [], 'L': [], 'K': [], 'H': [], 'A': ['B', 'C', 'D'], 'F': [], 'D': ['G', 'H'], 'B': ['E', 'F'], 'I': []}
crawled_dfs: ['A', 'D', 'H', 'G', 'L', 'K', 'C', 'B', 'F', 'E', 'J', 'I']
graph_bfs: {'E': ['I', 'J'], 'G': ['K', 'L'], 'J': [], 'C': [], 'L': [], 'K': [], 'H': [], 'A': ['B', 'C', 'D'], 'F': [], 'D': ['G', 'H'], 'B': ['E', 'F'], 'I': []}
crawled_bfs: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L']
```

图 5: 文件二运行结果

2.3 文件三：crawler

本文件加入了实际网络爬取，将文件二建立的爬虫模型落成实体。

2.3.1 辅助函数的增加

`valid_filename`函数的作用是把网页URL的内容通过过滤字符变成合法文件名。

```
def valid_filename(s):
    import string
    valid_chars = "-_(). %s%s" % (string.ascii_letters, string.digits)
    s = ''.join(c for c in s if c in valid_chars)
    return s
```

函数先引入`string`库，然后构建了文件名合法字符集，包括：连字符、下划线、句点、圆括号、大小写字母、阿拉伯数字。按照此合法字符集，过滤URL，把过滤后的字符串返回作为储存网页的文件名。

`add_page_to_folder`函数的作用是生成一个index.txt文件与一个html文件夹。其中前者储存网页URL与网页文件名的对应关系，后者以文件的形式储存爬取的网页。注意，在将网页内容写入文件时，使用的时‘wb’模式，即二进制写入。原因为，有些URL可能是word文档、图片等二进制文件，如果使用‘w’模式并且使用‘utf-8’编码，则会写入错误，导致下载的文件无法正常打开。

```
def add_page_to_folder(page, content):
    index_filename = 'index.txt'
    folder = 'html'
    filename = valid_filename(page)
    index = open(index_filename, 'a')
    index.write(page.encode('ascii', 'ignore').decode('ascii')
                + '\t' + filename + '\n')

    index.close()
    if not os.path.exists(folder):
        os.mkdir(folder)
    f = open(os.path.join(folder, filename), 'w', encoding='utf-8')

    f.write(content)
    f.close()
```

函数接受两个参数，第一个参数为URL，第二个参数为网页内容。先利用`valid_filename`生成合法文件名，然后将URL和文件名写入index.txt。然后判断是否已经存在文件夹，没有则创建。之后把网页内容保存在以合法文件名命名的文件中。

2.3.2 函数的修改

真正让爬虫从模型到实例的修改，是获取网页内容以及提取网页中链接的两个函数，以及`crawl`函数的少许修改。

`get_page`此函数为获取网页内容的函数。

```
def get_page(page):
    try:
        r = requests.get(page, timeout = 1, headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)'})
```



```
        r.raise_for_status()
        return r.text
    except:
        return ''
```

函数使用`requests`库提供的`get`函数获取网页内容，并且为了提高爬虫效率，本着有舍才有的思想，本实验使爬虫自动放弃响应缓慢的网页，设置等待响应最长时间为1s。在调整内容编码格式之后返回网页内容。如果响应时间超过1s或由于其他原因网页没有被正常的获取，则会由`r.raise_for_status()`抛出错误，最后函数返回空字符串。

`get_all_links`函数是提取URL的函数。本函数与上机实验1的练习一相同，使用了`beautifulsoup4`库进行解析，提取链接标签。再次就不做过多展示。

`crawl`函数相对于文件二做了些许调整。

首先，新增了一个最大限制参数，限制爬取网页的数量最大值，原因在于本函数牵涉下载网页占用储存空间，设置最大值防止将硬盘占满。实现方式是`while`语句增加第二个条件判断已爬取网页是否到达最大值。其次，增加了存储环节，使用`add_page_to_folder`写入文件。

2.3.3 文件使用的变化

为了让代码使用更加灵活，本代码接受命令行输入的三个参数，分别定义了爬虫的种子、搜索优先方式、最大爬取量。实现方式是使用`sys`中的`argv`获取参数内容。具体如下：

```
if __name__ == '__main__':

    seed = sys.argv[1]
    method = sys.argv[2]
    max_page = sys.argv[3]

    graph, crawled = crawl(seed, method, max_page)
```

文件运行示例如下：

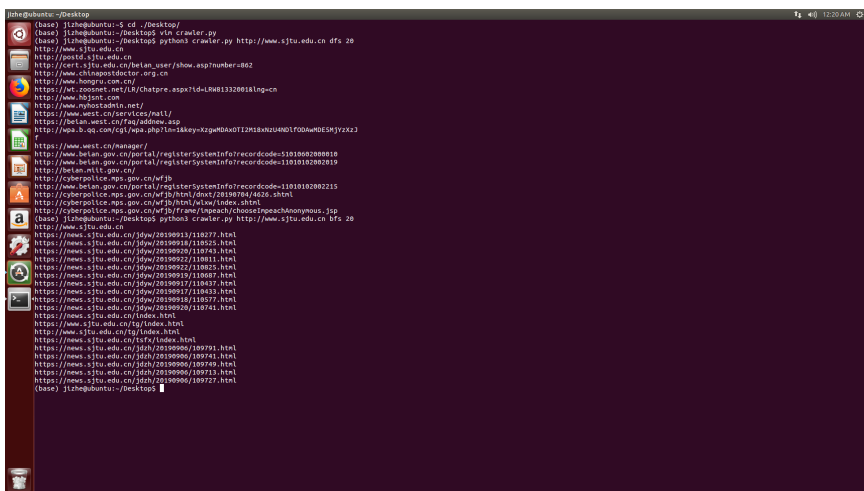


图 6: 当前命令行界面

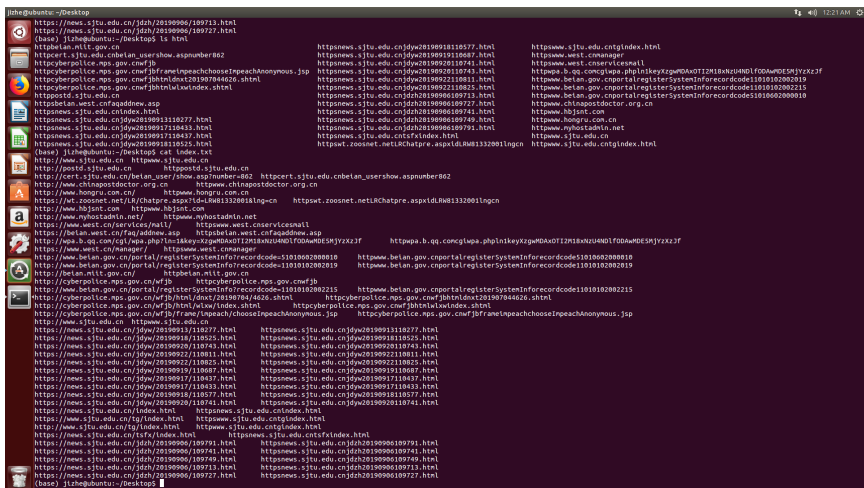


图 7: 文件更改情况

注：在爬取一些网站时，有链接会指向二进制文件，如word文档。在遇到这些URL的时候，代码会打印出一行警告，内容为

Some characters could not be decoded, and were replaced with REPLACEMENT CHARACTER.

其原因是，在用beautifulsoup解析文件提取URL时，*BeautifulSoup*的构建过程极有可能遇到无法解码的内容，代码会自动用*REPLACEMENT CHARACTER*替换这些二进制码，不会对爬取过程造成伤害，可以无视此

警告。

3 实验总结

3.1 概述

本次实验可分为两个大部分，一个是模拟登录的操作，一个是构建网络爬虫。第一个部分对应的是练习一。在练习一的操作中，对HTTP协议下的“会话”概念有了清晰的认知，大致上理解了“session”的含义，并成功通过程序模拟登录。第二个部分对应后三个联系，也就是后两个文件。其中前一个文件是后一个文件的代码模型，或者说是模板。实验深度由浅入深，先理解爬虫工作流程，再加入网络内容，实现小规模爬虫功能，能给人较大的成就感。

3.2 问题与解决

问题 爬虫爬取的二进制文件，比如word文档，无法正常打开。情况如图。



解决 在用浏览器打开相应URL下载文件发现文件源没有问题后，我开始寻找问题所在。我发现在原先的代码中，采用了encoding = utf8的编码格式，以‘w’的方式将网页内容记录在文件中。在网页为文本文件的时候这样做固然是没有问题的，但如果URL对应的为二进制文件，便会出错。解决方法就是把‘w’方式改为‘wb’方式，直接将二进制文件原封不动地写入新建的文件中。解决就此成功。

3.3 感想

经历了本次实验，我终于亲手写出了一个小型的，还不太完善的网络爬虫。虽然实现了网络爬虫的基本功能，但是对于网络爬虫应该有的礼貌性还没有实现。比如，这个初步爬虫会高频率地访问同一个服务器上的资源，而且未能读取robot协议，如果不慎爬取了禁爬内容，会引起不必要的麻烦。

需要学习的东西还有很多，要走的路还有很长，愿在这门课中学习更多的东西，让自己能够走得更远。