第二次实验报告

一、 实验概览

本实验旨在于理解 HTTP 协议的具体内容，包括 HTTP 请求的过程，提交的 HTML 表单包含内容，并利用 python 代码模拟实现发送 HTTP 请求。实验的另一部分旨在于理解爬虫的概念，学习使用爬虫技术应遵守的原则，实现爬虫抓取网页的两种策略。

二、 实验环境

Docker: SJTU-EE208

三、 解决思路

1. 练习一

基于 urllib 中的 request 类，利用 add_header 方法添加 header 模拟浏览器对网络进行访问；利用 cookiejar 库构建登录所需的 cookie 信息加入 request 中实现登录并获得网页。

```python
# 1. 构建一个CookieJar对象实例来保存cookie
cookie = cookiejar.CookieJar()
# 2. 使用HTTPCookieProcessor()来创建cookie处理器对象，参数为CookieJar()对象
cookie_handler = urllib.request.HTTPCookieProcessor(cookie)
# 3. 通过build_opener()来构建opener
opener = urllib.request.build_opener(cookie_handler)
# 4. addheaders接受一个列表，里面每个元素都是一个headers信息的元组，opener附带headers信息
opener.addheaders = [("User-Agent", 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.71 Safar
# 5. 需要登陆的账号和密码，此处需要使用你们自己注册的账号密码
data = {"username": "plwu0064",
        "pwd": "Wu_1654088009",
        # "formhash": "4A95C39D38",
        "backurl": "https%3A%2F%2Fwww.yaozh.com%2F"}
sign_in_url = "https://www.yaozh.com/login/"
info_url = "https://www.yaozh.com/member/basicinfo/"

# 6. 通过urlencode转码
postdata = urllib.parse.urlencode(data).encode("utf8")
# 7. 构建Request请求对象，包含需要发送的用户名和密码
request = urllib.request.Request(sign_in_url, data=postdata,headers = dict(Referer = sign_in_url))
# 8. 通过opener发送这个请求，并获取登陆后的Cookie值
opener.open(request)
# 9. opener包含用户登陆后的Cookie值，可以直接访问那些登陆后才能访问的页面
response = opener.open(info_url).read()
```

基于 bs4 对 html 网页源码进行解析，通过开发者模式查看网页发现所需信息格式均在 中，以<dl>标签分段排列，利用 BeautifulSoup 类的 findAll 方法查找对应标签得到信息。

```
soup = BeautifulSoup(response, 'html.parser')

try:
    div = soup.findAll("div", {"class": "U_myinfo clearfix"})[0]
    dls = div.findAll("dl")
    for dl in dls:
        dt = dl.contents[1].string
        if dt == "真实姓名: ":
            input = dl.contents[2].contents[0]
            name = input['value']
        elif dt == "用户名: ":
            input = dl.contents[2].contents[0]
            username = input['value']
        elif dt == "性别: ":
            input = dl.contents[2].find('input', {"checked": "checked"})
            sex = input['value']
        elif dt == "出生年月: ":
            input = dl.contents[2].contents[0]
            date = input['value']
        elif dt == "简介: ":
            input = dl.contents[2].contents[0]
            resume = input.string

    print(f"真实姓名: {name} \n 用户名: {username} \n 性别: {sex} \n 出生年月: {date} \n 简介: {resume}")
except:
    pass
```

在具体实施过程中，发现登录次数一定后网页会有验证码需求，无法通过 urllib 直接登录，故考虑使用 selenium 库模拟网页图形界面解决验证码问题。

```
chrome.get(sign_in_url)
input = chrome.find_element(By.ID, "username")
input.send_keys("plwu0064")
input = chrome.find_element(By.ID, "pwd")
input.send_keys("Wu_1654088009")
time.sleep(5)
chrome.find_element(By.ID, "button").click()
time.sleep(5)

chrome.get(info_url)
response = chrome.page_source
```

之后的解析过程一致

2. 练习二

利用 list 类的 insert 操作实现 BFS

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

3. 练习三

分析代码，确定 outlinks 变量包含了每个节点下子节点，赋值实现输出 graph

```python
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_%s' % method](tocrawl, outlinks)
            crawled.append(page)
    return graph, crawled
```

4. 练习四

基于 urllib 库获取网页信息并进行异常处理

```python
def get_page(page):
    content = ''

    header = ("User-Agent", "Mozilla/5.0 (X11; Fedora; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110Safari/537.36")
    request = urllib.request.Request(page)
    request.add_header(*header)
    try:
        content = urllib.request.urlopen(request, timeout=5).read()
    except:
        pass
#
    return content
```

基于 BeautifulSoup 类和正则表达式查找超链接子网页，根据是否以 http 开头判断相对网页绝对网页并得到最终网页路径。

```python
def get_all_links(content, page):
    links = []

    soup = BeautifulSoup(content, 'html.parser')
    hypers = soup.findAll('a', {'href' : re.compile('^http|^/')})
    for hyper in hypers:
        href = hyper["href"]
        if href[:4] == 'http':
            links.append(href)
        else:
            links.append(urllib.parse.urljoin(page, href))

    return links
```

四、 代码运行结果

练习一：执行 python example1.py，如需输入验证码在图形界面中输入，在终端中显示结果

```
真实姓名：吴沛琳
 用户名：plwu0064
 性别：2
 出生年月：2004-05-01
 简介：简介测试
```

练习二、三：运行 python crawler_sample.py 在终端中得到结果

```
graph_dfs: {'A': ['B', 'C', 'D'], 'D': ['G', 'H'], 'H': [], 'G': ['K', 'L'], 'L': [], 'K': [], 'C': [], 'B': ['E', 'F'], 'F': [], 'E': ['I', 'J'], '
J': [], 'I': []}
crawled_dfs: ['A', 'D', 'H', 'G', 'L', 'K', 'C', 'B', 'F', 'E', 'J', 'I']
graph_bfs: {'A': ['B', 'C', 'D'], 'B': ['E', 'F'], 'C': [], 'D': ['G', 'H'], 'E': ['I', 'J'], 'F': [], 'G': ['K', 'L'], 'H': [], 'I': [], 'J': [], '
K': [], 'L': []}
crawled_bfs: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L']
```

练习四：运行 python crawler.py 直接得到对应 index.txt 和文件夹，附图为 index.txt，具体结果请参考对应文件

```
'http://www.sjtu.edu.cn'    httpwww.sjtu.edu.cn
'https://vs.sjtu.edu.cn/'   httpsvs.sjtu.edu.cn
'https://vs.sjtu.edu.cn/jtdx/index/imagesList'  httpsvs.sjtu.edu.cnjtdxindeximagesList
'https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=51'   httpsvs.sjtu.edu.cnjtdxindeximagesListthemeId51
'https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=41'   httpsvs.sjtu.edu.cnjtdxindeximagesListthemeId41
'https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=49'   httpsvs.sjtu.edu.cnjtdxindeximagesListthemeId49
'https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=45'   httpsvs.sjtu.edu.cnjtdxindeximagesListthemeId45
'https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=54'   httpsvs.sjtu.edu.cnjtdxindeximagesListthemeId54
'https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=40'   httpsvs.sjtu.edu.cnjtdxindeximagesListthemeId40
'https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=42'   httpsvs.sjtu.edu.cnjtdxindeximagesListthemeId42
```

五、 分析与思考：网络爬虫的合法性？

网络爬虫需遵循礼貌性原则，爬虫对于浏览器访问频率过高，导致服务器崩溃，可能要承担对应法律责任；同时针对爬取信息应遵守 robot.txt 文件，爬取公民敏感信息则可能承担对应民事乃至刑事责任。