Lab 2 Crawler

姓名：刘洺皓 学号：521030910014 班级：F2103001

1.实验概述

本实验分为两个部分：

- 本实验首先利用 *urllib.request* 和几个 *HTTP Base Handler类* 来实现网页登录，即表单数据提交，利用药智网来进行试验，过程中需要设置 *cookie* 来维持对话，设置 *header* 来模仿浏览器请求，传递data来实现表单提交，并且需要处理隐藏标签。
- 本实验旨在通过运用深度优先搜索(Depth-First-Search,DFS)和广度优先搜索(Breath-First-Search)算法来实现网址爬取，这里假设网络是一个连通性很好的图，并且这里可以将网络视为一棵树，并且可以用并查集来查询网址是否重复。DFS与BFS的主要区别就在于一个节点的叶节点放置在存储网络数据的线性结构的首端还是尾端。

2.实验环境

本实验具体使用的工具如下:

- Docker中的ee208镜像
- HTML解析器 *BeautifulSoup*
- Python(Vscode)

3.实验练习

3.1.1问题描述

> 自己先注册一下药智网账号（https://www.yaozh.com/)并填写自己的个人信息，接着仿照第9页PPT的做法，利用cookie登陆自己的个人主页（https://www.yaozh.com/member/basicinfo/)，并定位打印出自己个人主页中的真实姓名，用户名，性别，出生年月和简介。

3.1.2 问题分析

```
<form class="login_pc" id="login_pc">
    ...
    <input type="hidden" name="type" id="logintype" value="0">
    ...
    <input type="hidden" name="formhash" id="formhash" value="22F1C35A03">
    <input type="hidden" name="backurl" id="backurl" value="https%3A%2F%2Fwww.yaozh.com%2F">
</form>
```

这是一个 *HTML* 表单提交，当按下此元素
时， `<button id="button" class="in" type="submit" value="登录" style="cursor: pointer">登 录</button>` ， 表单将会以 *post* 的请求方法将表单数据以键值形式放在 *request body* 提交到服务器端,但是该网站的 `<input>...</input>` 数据中还有隐藏提交项，应该补
全 `{"formhash":"7B1DFE", "backurl":'https%3A%2F%2Fwww.yaozh.com%2F'}` 来模仿浏览器请求。这引导我们在爬取有Auth的网站时可以定位到表单中的各个具有 '*name*' 的属性进行对 '*value*' 属性的爬取并补充到表单提交数据中。但是审览网页元素时发现还有一
个 `<input type="hidden" name="type" id="logintype" value="0">` ,在爬取过程中发现在提交表单数据中并不需要补充 `{'type':'0'}` ，
从 `name="type" id="logintype"` 可以做出推测这里应该只是表征登录方式，为了方便后端服务器处理。

```
<form class="login_pc" id="login_pc">
  <input type="hidden" name="type" id="logintype" value="0">
  <br>
  <div class="pop_showmsg alert-error Mb_20 msg" id="J_login_msg" style="display: none;"></div>
  <div class="login-panel" style="display: block;">
    <li class="li1 user">
      <input type="text" name="username" id="username" class="checkitem" placeholder="手机号/用户名/邮箱">
    </li>
    <li class="li2 pass">…</li>
    <li class="li4" id="pc_show_vcode" style="display: none">…</li>
  </div>
  <div class="login-panel">…</div>
  <div class="clearfix">…</div>
  <button id="button" class="in" type="submit" value="登录" style="cursor: pointer">登 录</button>
  <div class="clearfix oways-w">…</div>
  <input type="hidden" name="formhash" id="formhash" value="22F1C35A03">
  <input type="hidden" name="backurl" id="backurl" value="https%3A%2F%2Fwww.yaozh.com%2F">
</form>
```

```
<dt>用户名：</dt>
<dd>
<input type="text" value="lmh209" name="nickname" readonly="true" class="Y_input" size="30">
</dd>
```

3.1.2 代码与运行结果

代码展示如下:

```
from urllib import request,parse,error
from bs4 import BeautifulSoup
import ssl
from http import cookiejar
import random
from lxml import etree
import re


login_url = 'https://www.yaozh.com/login/'
url = "https://www.yaozh.com/member/basicinfo/"
username = "lmh209"
pwd = "111111a"

def standard_request(url, headers = {}, required_login = False, args = {}):
    cookie = cookiejar.CookieJar()
    cookie_handler = request.HTTPCookieProcessor(cookie)
    opener = request.build_opener(cookie_handler)


    #防止服务器对请求头中不重要的信息进行检查来鉴别人还是虫
    # 但是还有诸如 refereer, authority, host和最重要的 cookie不具有普适性

    header = user_agent[random.randint(0,len(user_agent)-1)]
    if not headers:
        headers = {
                "Accept": "text/html, application/xhtml+xml, */*",
                "Accept-Language": "en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7",
                "Connection": "Keep-Alive",
                }

    opener.addheaders = [("User Agent", header)]

    data = None if not required_login else parse.urlencode(args).encode('utf8')
    open_Request = request.Request(url, data=data, headers = headers)

    try:
        content = opener.open(open_Request, timeout = 2).read()
        return opener, content
    except error.HTTPError as e:
        if (e.code == 418 or e.code == 400): #增加host请求再试一次
            host = parse.urlparse(url=url).scheme + "://" + parse.urlparse(url=url).netloc
            headers["host"] = host
            try:
                open_Request = request.Request(url, data, headers = headers)
                return opener.open(open_Request, timeout = 2).read()
            except error.HTTPError as e:
                print('HTTPError' + '\t' + str(e.code) + '\t' + str(e.reason))
                return
        print('HTTPError' + '\t' + str(e.code) + '\t' + str(e.reason))
    except error.URLError as e:
        if isinstance(e.reason, socket.timeout):
            print('TimeOut Error')
        else:
            print('URLError' + '\t' + str(type(e.reason)))


data = {"username":"lmh209","pwd":"111111a","formhash":"22F1C35A03", "backurl":"https%3A%2F%2Fwww.yaozh.com%2F"}

headers = dict()
headers['headers'] = user_agent[random.randint(0,len(user_agent)-1)]
opener, content = standard_request(login_url, required_login = True, args = data)
open_Request = request.Request(url, headers = headers)
response = opener.open(url).read()
soup = BeautifulSoup(response, "lxml")
target = soup.find("div", class_="U_myinfo clearfix").contents

print("真实姓名:  " + target[3].contents[2].next_element['value'])

print("用户名: " + target[5].contents[2].next_element['value'])

print("性别: " + target[7].contents[2].next_element['value'])

print("出生年月: " + target[9].contents[2].next_element['value'])

print("简介: " + target[11].contents[2].next_element.string)
```

```
login_url = 'https://www.zhihu.com/signin'
url = 'https://www.zhihu.com/people/liu-jia-zhu-54-65'
data = {
'username':'138188442238',
"password":'MAROUANEattend1'
}


headers = dict()
headers['headers'] = user_agent[random.randint(0,len(user_agent)-1)]
opener, content = standard_request(login_url, required_login = True, args = data)
open_Request = request.Request(url, headers = headers)
response = opener.open(url).read()


def findcontentbyXpath(xpath, document):
    element = document.xpath(xpath)
    tag = etree.tostring(element[0], encoding='utf-8').decode('utf-8')
    pat = re.compile('<.*(?<=>)(.*?)(?=[<>])', re.DOTALL)
    res = re.search(pat, tag)
    return "" if res is None else res.group(1)


content = opener.open(open_Request).read()

html = etree.HTML(content)

name = findcontentbyXpath('//*[@id="ProfileHeader"]/div/div[2]/div/div[2]/div[1]/h1/span[1]', html)
print("名字是: ", name)
info = findcontentbyXpath('/html/body/div[1]/div/main/div/div[1]/div/div[2]/div/div[2]/div[1]/h1/span[2]', html)
print("个人信息: ", info)
q1 = findcontentbyXpath('/html/body/div[1]/div/main/div/div[2]/div[1]/div/div[3]/div/div[2]/div[1]/div[2]/div/h2/div/a', html)
print('问题1: ' + q1)
a1 = findcontentbyXpath('/html/body/div[1]/div/main/div/div[2]/div[1]/div/div[3]/div/div[2]/div[1]/div[2]/div/div[2]/span/div/span', html)
print('回答1: ' + a1)
q2 = findcontentbyXpath('/html/body/div[1]/div/main/div/div[2]/div[1]/div/div[3]/div/div[2]/div[2]/div[2]/div/h2/span/div/a', html)
print('问题2: ' + q2)
# a2 = findcontentbyXpath(, html)
#print('回答2: ' + a2)
```

运行结果如下:

```
真实姓名:  刘德华
用户名: lmh209
性别: 1
出生年月: 2022-09-20
简介: 小舟从此逝，江海寄余生。
名字是:  editujan
个人信息:
问题1: 大一学习C/C++时是否需要大量刷题?
回答1: ： 看了你的老师给你们的建议，真的是非常认真和负责。作为一个以后想从事编程相关工作的大学生，真的，在大学里打下扎实的算法与数据结构基础，锻炼出自…
问题2: 请问madden nfl mobile怎么登进去? 我的一直闪退       ?
```

分析与反思:

1.

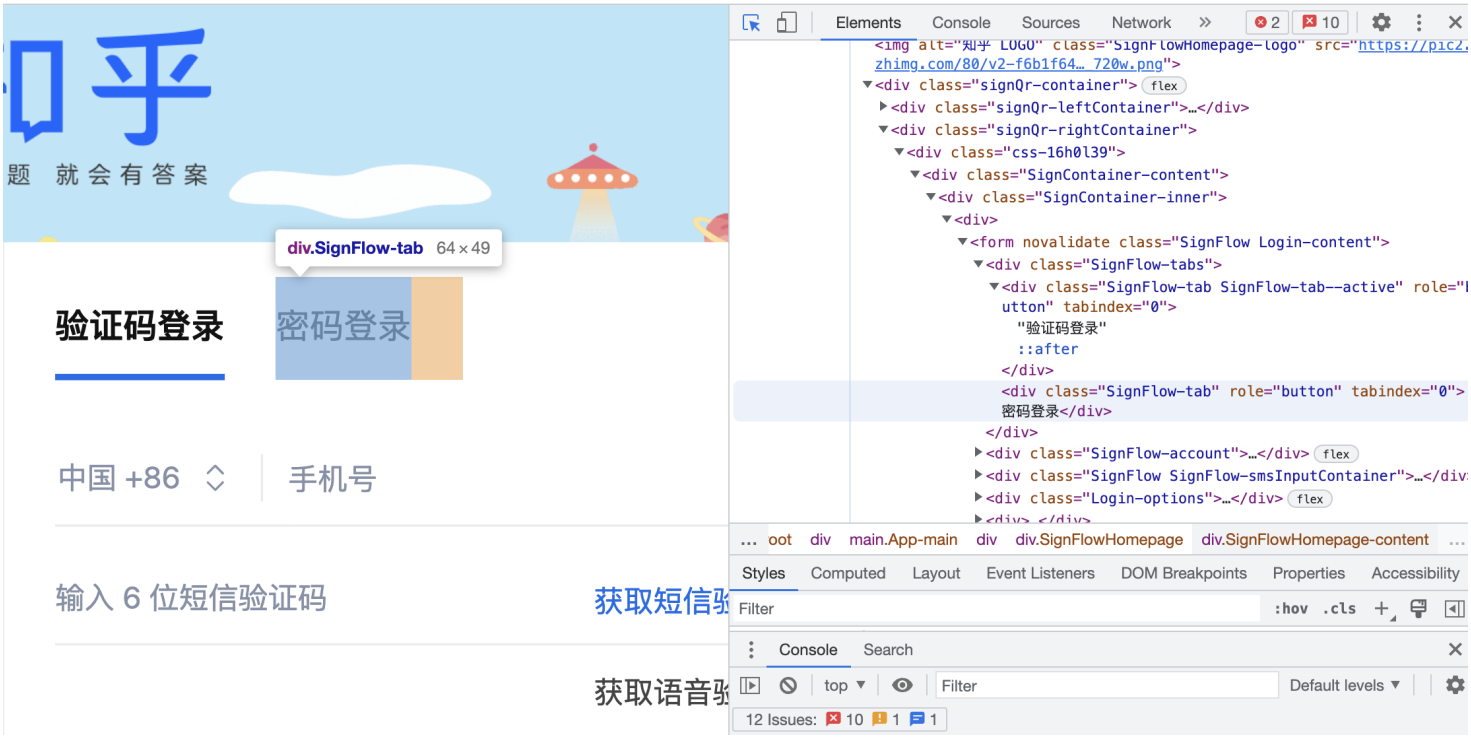
```
p = request.HTTPPasswordMgrWithDefaultRealm()
p.add_password(None, url, user=username, passwd=password)
auth = request.HTTPBasicAuthHandler(p)
cookie = cookiejar.CookieJar()
cookie_handler = request.HTTPCookieProcessor(cookie)
opener = request.build_opener(cookie_handler, auth)
opener.addheaders = [('User Agent', header)]

try:
    data = opener.open(login_url).read()
    response = opener.open(url).read()
    soup = BeautifulSoup(response, "lxml")
    print(soup.title.string)
except error.HTTPError as e:
    print("HTTPError", e.code, e.reason, e.headers)
except error.URLError as e:
    print("URLError", e.reason)
```

经试验，该段代码也可以实现登录操作(QQMail)，但是他并不能处理药智网的登录，初步判断原因为 *HTTPBasicAuthHandler* 和 *HTTPPasswordMgr* 只能处理表单提交中只出现简单的 *Username* 和 *Password* 的 `<input/>` 标签时的情况，并不能处理隐藏标签，在处理表单数据提交时要视情况而分析。

还有一种常见的反爬机制称为为蜜罐，他是在HTML文档中定义了一些 `<input type="hidden">` 或者CSS属性设为 `display:none` 的HTML元素，这些表单元素在浏览器上不会显示，如果爬虫也去填写了这些表单元素，那就有证据证明这非人为，所以在处理表单提交时也要注意诸如此类的表单陷阱。

2.

在实现知乎的登录操作时，遇上了需要验证码登录与密码登录的转换，然后inspect，



发现以那种方式登录，那个 `<div></div>` 标签的class属性即是 `SignFlow-tab SignFlow-tab--active` , 另一个就是 `SignFlow-tab` ,在这个属性附近并未找到 `<input/>` 或者 `<button>` 类实现两种登录方式的转换，可能在 `<head></head>` 标签中 `<script type="text/javascript" ...>...</script>` 中存在一个事件监听器

```
<div>
  <div class="SignFlowHomepage">flex
    <div class="SignFlowHomepage-content">flex
      <img alt="知乎 LOGO" class="SignFlowHomepage-logo" src="https://pic2.zhimg.com/80/v2-f6b1f64… 720w.png">
      <div class="signQr-container">flex
        <div class="signQr-leftContainer">…</div>
        <div class="signQr-rightContainer">
          <div class="css-16h0l39">
            <div class="SignContainer-content">
              <div class="SignContainer-inner">
                <div>
                  <form novalidate class="SignFlow Login-content">
                    <div class="SignFlow-tabs">…</div>
                    <div class="SignFlow-account">…</div>flex
                    <div class="SignFlow SignFlow-smsInputContainer">
                      <div class="SignFlowInput SignFlow-smsInput">
                        <label class="Input-wrapper">flex
                          <input name="digits" type="number" class="Input username-input" placeholder="输入 6 位短信验证码" value>
                        </label>
                        <div class="SignFlowInput-errorMask SignFlowInput-requiredErrorMask SignFlowInput-errorMask--hidden">…</div>
                      </div>
                      <button type="button" class="Button CountingDownButton SignFlow-smsInputButton Button--plain">获取短信验证码</button>
```

发现两种方法的用户名都是 `<name="username">` , 但是验证码的登录对应 `<name="digits">` ,密码的登录对应 `<name="password">` ,但是不需要点击进入密码登录界面才能进行表单提交，两种界面只是一个显示在浏览器页面上，受 *JavaScript* 的一个事件监听器监听，来改变HTML页面的 *CSS* 和标签属性,并不妨碍表单提交。

3.2.1 问题描述

> 修改crawler_sample.py中的union_bfs函数，完成BFS搜索

3.2.1 问题分析

利用 `.insert()` 函数。

3.2.3 代码与运行结果

代码展示如下:

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

运行结果略

3.3.1 问题描述

> 修改crawler_sample.py中的crawl函数，返回图的结构。graph结构与crawler_sample.py中g的结构相同。完成后运行graph, crawled = crawl('A', 'bfs')。查看graph 中的图结构，以及crawled中的爬取结果顺序。

3.3.2 问题分析

所提供的g的结构满足一层一层从左向右标注英文字母的顺序，这是一个可以利用的很好的性质，我们知道从'A'到最大字母之间的所有字母一定都是该树的节点，故可以用查询键值这种效率较高的方法来访问，而且可以知道所有键的值里的元素是连续着排列的，所以只要每访问一个节点，将他的儿子节点依次放入队列，这样队列deque的顺序正好就按照 *'A''B''C''D'...* 进行。

3.3.3 代码与运行结果

代码展示如下:

```
import queue

def crawl(seed, method):
    tocrawl = [seed]
    crawled = []
    graph = dict()

    q = queue.Queue(26)
    q.put(seed)

    while not q.empty():
        deque = q.get()
        if deque in g.keys():
            graph[deque] = g[deque]
            for item in g[deque]:
                q.put(item)
        else:
            graph[deque] = ""

    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
            content = get_page(page)
            outlinks = get_all_links(content)
            globals()['union_%s' % method](tocrawl, outlinks)
            crawled.append(page)
    return graph, crawled
```

运行结果如下:

```
[Running] python3 -u "/Users/liliu/Desktop/lab/lab2/code/crawler_sample.py"
graph_dfs: {'A': ['B', 'C', 'D'], 'B': ['E', 'F'], 'C': '', 'D': ['G', 'H'], 'E': ['I',
'J'], 'F': '', 'G': ['K', 'L'], 'H': '', 'I': '', 'J': '', 'K': '', 'L': ''}
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']
```

分析与反思：

这一题相当于给定树的结构来进行DFS与BFS遍历，在经典的解法中，我们时常在DFS中运用栈，在BFS中运用队列，这与把新得到的儿子节点放在线性结构的首端或尾端如出一辙。

上述利用 *queue* 来访问g以此来刻画graph的结构的方法类似于树的前序遍历，只是传统意义上的前序遍历讲究用递归实现，故效果中带有回溯，这里如果把每个键值放入队列的行为看成是访问该节点，那么每次deque的对象所组成的队列在字母序上是连续的也就容易理解了。

3.4.1 问题描述

> 进一步修改函数，完成网页爬虫（修改crawler.py）

3.4.2 问题分析

已经完善好DFS和BFS后，关键的问题就是爬虫陷阱的处理和请求网页错误的异常处理了。也即是实现 `standard_request(url, required_login = False, args = {})` 函数了。

- 在请求网页错误的异常处理中，可以利用 `urllib.error` 来实现返回错误，其中 `error.HTTPError` 是继承 `error.URLError` 的类，在异常处理时具有优先级。考虑到 `opener.open(open_Request, timeout = timeout)` 中可以设置 *timeout* 参数，一个错误 *urllib.error* 对象有三个属性(e.reason(返回错误原因), e.code（返回错误状态码（参考HTTP状态码[1]）, e.headers（返回请求头)），而超时错误是 *URLError* 的子类，且其reason是一个 *urllib* 实现底层的 *socket* 库的 *socket.timeout* 实例，故也可将超时错误单独处理。
- 在使爬虫更加具有真实性的考量上，最实用的办法在于请求头的伪造上，一般 "*User Agent*" 是服务器检查最频繁的HTTP属性，但是某些网页也可能将: '*authority*','*method*','*accept*','*accept-encoding*','*accept-language*','*connection*' 等鲜少人注意的属性作为判别人与机的依据，但是除了 "*User Agent*" 外最常见的检查目标还是集中在 '*Host*','*Referrer*' 前者表示主域名(netloc), 可以用 `parse.urlparse().scheme` 和 `parse.urlparse().netloc` 的concat() 来构造一个合法的 '*Host*' 域名，而 '*Referrer*' 目前网页是从哪里翻转过来的，而为了使 '*Referrer*' 更具有真实性，可以在参数中加上一个parent, 意为目标网页的父节点。
- 在爬虫陷阱的处理上，有可能有 `<iframe>` 动态网页限制， 也有可能有 php/asp 文件，也有可能是一个 *HTTP 302 Redirects 重定向* 所构成的闭环回路等类型的无限循环，这在分析与思考与拓展思考中有部分提及。

3.4.3 代码与运行结果

代码展示如下:

```
import os
import queue
import re
import socket
import string
import sys
from urllib import request,parse,error
from http import cookiejar
import ssl
from bs4 import BeautifulSoup
import random

#全局取消SSl证书
ssl._create_default_https_context = ssl._create_unverified_context

user_agent= [
    # Opera
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36 OPR/26.0.1656.60",
    "Opera/8.0 (Windows NT 5.1; U; en)",
    "Mozilla/5.0 (Windows NT 5.1; U; en; rv:1.8.1) Gecko/20061208 Firefox/2.0.0 Opera 9.50",
    "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; en) Opera 9.50",
    # Firefox
    "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0",
    "Mozilla/5.0 (X11; U; Linux x86_64; zh-CN; rv:1.9.2.10) Gecko/20100922 Ubuntu/10.10 (maverick) Firefox/3.6.10",
    # Safari
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.57.2 (KHTML, like Gecko) Version/5.1.7 Safari/534.57.2",
    # chrome
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.71 Safari/537.36",
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.64 Safari/537.11",
    "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Chrome/10.0.648.133 Safari/534.16",
    # 360
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.101 Safari/537.36",
    "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko",
    # 淘宝浏览器
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.11 TaoBrowser/2.0 Safari/536.11",
    # 猎豹浏览器
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.71 Safari/537.1 LBBROWSER",
    "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center
    "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; QQDownload 732; .NET4.0C; .NET4.0E; LBBROWSER)",
    # QQ浏览器
    "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center
    "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; QQDownload 732; .NET4.0C; .NET4.0E)",
    # sogou浏览器
    "Mozilla/5.0 (Windows NT 5.1) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.84 Safari/535.11 SE 2.X MetaSr 1.0",
    "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; SV1; QQDownload 732; .NET4.0C; .NET4.0E; SE 2.X MetaSr 1.0)",
    # maxthon浏览器
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Maxthon/4.4.3.4000 Chrome/30.0.1599.101 Safari/537.36",
    # UC浏览器
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 UBrowser/4.0.3214.0 Safari/537.36",

    # 各种移动端

    # IPhone
    "Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_3 like Mac OS X; en-us) AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2 Mobile/8J2 Safari/6533.18.5",
    # IPod
    "Mozilla/5.0 (iPod; U; CPU iPhone OS 4_3_3 like Mac OS X; en-us) AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2 Mobile/8J2 Safari/6533.18.5",
    # IPAD
    "Mozilla/5.0 (iPad; U; CPU OS 4_2_1 like Mac OS X; zh-cn) AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2 Mobile/8C148 Safari/6533.18.5",
    "Mozilla/5.0 (iPad; U; CPU OS 4_3_3 like Mac OS X; en-us) AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2 Mobile/8J2 Safari/6533.18.5",
    # Android
    "Mozilla/5.0 (Linux; U; Android 2.2.1; zh-cn; HTC_Wildfire_A3333 Build/FRG83D) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1",
    "Mozilla/5.0 (Linux; U; Android 2.3.7; en-us; Nexus One Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1",
    # QQ浏览器 Android版本
    "MQQBrowser/26 Mozilla/5.0 (Linux; U; Android 2.3.7; zh-cn; MB200 Build/GRJ22; CyanogenMod-7) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile S
    # Android Opera Mobile
    "Opera/9.80 (Android 2.3.4; Linux; Opera Mobi/build-1107180945; U; en-GB) Presto/2.8.149 Version/11.10",
    # Android Pad Moto Xoom
    "Mozilla/5.0 (Linux; U; Android 3.0; en-us; Xoom Build/HRI39) AppleWebKit/534.13 (KHTML, like Gecko) Version/4.0 Safari/534.13",
    # BlackBerry
    "Mozilla/5.0 (BlackBerry; U; BlackBerry 9800; en) AppleWebKit/534.1+ (KHTML, like Gecko) Version/6.0.0.337 Mobile Safari/534.1+",
    # WebOS HP Touchpad
    "Mozilla/5.0 (hp-tablet; Linux; hpwOS/3.0.0; U; en-US) AppleWebKit/534.6 (KHTML, like Gecko) wOSBrowser/233.70 Safari/534.6 TouchPad/1.0",
    # Nokia N97
    "Mozilla/5.0 (SymbianOS/9.4; Series60/5.0 NokiaN97-1/20.0.019; Profile/MIDP-2.1 Configuration/CLDC-1.1) AppleWebKit/525 (KHTML, like Gecko) BrowserNG/7.
    # Windows Phone Mango
    "Mozilla/5.0 (compatible; MSIE 9.0; Windows Phone OS 7.5; Trident/5.0; IEMobile/9.0; HTC; Titan)",
```

```
    # UC浏览器
    "UCWEB7.0.2.37/28/999",
    "NOKIA5700/ UCWEB7.0.2.37/28/999",
    # UCOpenwave
    "Openwave/ UCWEB7.0.2.37/28/999",
    # UC Opera
    "Mozilla/4.0 (compatible; MSIE 6.0; ) Opera/UCWEB7.0.2.37/28/999"

    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/22.0.1207.1 Safari/537.1",
    "Mozilla/5.0 (X11; CrOS i686 2268.111.0) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.57 Safari/536.11",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.6 (KHTML, like Gecko) Chrome/20.0.1092.0 Safari/536.6",
    "Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.6 (KHTML, like Gecko) Chrome/20.0.1090.0 Safari/536.6",
    "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/19.77.34.5 Safari/537.1",
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/19.0.1084.9 Safari/536.5",
    "Mozilla/5.0 (Windows NT 6.0) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/19.0.1084.36 Safari/536.5",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3",
    "Mozilla/5.0 (Windows NT 5.1) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_0) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1062.0 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1062.0 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.0 Safari/536.3",
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.24 (KHTML, like Gecko) Chrome/19.0.1055.1 Safari/535.24",
    "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/535.24 (KHTML, like Gecko) Chrome/19.0.1055.1 Safari/535.24"
]


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

def standard_request(url, required_login = False, args = {}, timeout = 2):
    cookie = cookiejar.CookieJar()
    cookie_handler = request.HTTPCookieProcessor(cookie)
    opener = request.build_opener(cookie_handler)

    header = user_agent[random.randint(0,len(user_agent)-1)]

    #防止服务器对请求头中不重要的信息进行检查来鉴别人还是虫
    # 但是还有诸如 refereer, authority, host和最重要的 cookie不具有普适性
    headers = {
            "User Agent": header,
            "Accept": "text/html, application/xhtml+xml, */*",
            "Accept-Language": "en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7",
            "Connection": "Keep-Alive"
            }


    data = None if not required_login else bytes(parse.urlencode(args).encode())
    open_Request = request.Request(url, data, headers = headers)

    try:
        return opener.open(open_Request, timeout = timeout).read()
    except error.HTTPError as e:
        if (e.code == 418 or e.code == 400):
            host = parse.urlparse(url=url).scheme + "://" + parse.urlparse(url=url).netloc
            headers["host"] = host
            try:
                open_Request = request.Request(url, data, headers = headers)
            except error.HTTPError as e:
                print('HTTPError' + '\t' + str(e.code) + '\t' + str(e.reason))
        print('HTTPError' + '\t' + str(e.code) + '\t' + str(e.reason))
    except error.URLError as e:
        if isinstance(e.reason, socket.timeout):
            print('TimeOut Error')
        else:
            print('URLError' + '\t' + str(type(e.reason)))
    except TimeoutError as e:
        print("TimeOut Error")
        print(e)


def get_page(page):
    return str(standard_request(page))
```

```
def get_all_links(content, page):
    links = []
    data = BeautifulSoup(content, "lxml")

    for item in data.find_all('a', href=re.compile(r'^http|^/')):
        if item['href'][0] == 'h':
            links.append(item['href'])
        else:
            links.append(parse.urljoin(page, item['href']))
    return links


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)


def add_page_to_folder(page, content):  # 将网页存到文件夹里，将网址和对应的文件名写入index.txt中
    index_filename = 'index.txt'  # index.txt中每行是'网址 对应的文件名'
    folder = 'html'  # 存放网页的文件夹
    filename = valid_filename(page)  # 将网址变成合法的文件名
    index = open(index_filename, 'a')
    index.write(page + '\t' + str(filename) + '\n')
    index.close()
    if not os.path.exists(folder):  # 如果文件夹不存在则新建
        os.mkdir(folder)
    f = open(os.path.join(folder, filename), 'w')
    f.write(content)  # 将网页存入文件
    f.close()

def display(page):  # str->null
    if isinstance(page, str):
        pat = re.compile(r'(.+(\\x[0-9a-zA-Z]{2})+.+)+|(.+(%[0-9a-zA-Z]{2})+.+)+', re.DOTALL)
        if re.match(pat, page.__repr__()) is not None:
            # print('Contains Chinese!')
            s = page.encode('raw_unicode_escape')
            ss = s.decode('utf-8').replace('\\x', '%')
            print(parse.unquote(ss))
        else:
            print(page)

def crawl(seed, method, max_page):
    tocrawl = [seed]
    crawled = []
    graph = {}
    count = 0
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:

            display(page)

            content = get_page(page)
            add_page_to_folder(page, content)
            outlinks = get_all_links(content, page)

            globals()['union_%s' % method](tocrawl, outlinks)
            crawled.append(page)
            graph[page] = outlinks
            if (len(crawled) > int(max_page)):
                return graph, crawled
    return graph, crawled


if __name__ == '__main__':

    seed = sys.argv[1]
    method = sys.argv[2]
```

```
    max_page = sys.argv[3]

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

运行结果如下：

```
liliu@bitch code % python3 crawler.py http://www.sjtu.edu.cn dfs 10
http://www.sjtu.edu.cn
https://vs.sjtu.edu.cn/
https://vs.sjtu.edu.cn/jtdx/index/imagesList
https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=51
https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=41
https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=49
https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=45
https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=54
https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=40
https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=42
https://vs.sjtu.edu.cn/jtdx/index/imagesList?themeId=37
liliu@bitch code % python3 crawler.py http://www.sjtu.edu.cn bfs 10
http://www.sjtu.edu.cn
https://mp.weixin.qq.com/s/BrP2KVzU1b6e5AMwAMuHeQ
https://mp.weixin.qq.com/s/LHPVr3QLowSysAlMVNE3cw
https://news.sjtu.edu.cn/jdyw/20220917/174379.html
https://news.sjtu.edu.cn/jdyw/20220916/174357.html
https://news.sjtu.edu.cn/jdzh/20220918/174383.html
https://mp.weixin.qq.com/s/O2kD_8dP7ktreGQ1WoBiSQ
https://news.sjtu.edu.cn/jdyw/20220919/174405.html
https://news.sjtu.edu.cn/jdyw/20220919/174428.html
https://news.sjtu.edu.cn/jdyw/20220920/174459.html
https://news.sjtu.edu.cn/jdyw/20220920/174453.html
```

带有中文输出的结果：

```
liliu@bitch code % python3 crawler.py http://www.baidu.com bfs 40
http://www.baidu.com
http://www.baidu.com/
https://passport.baidu.com/v2/?login&tpl=mn&u=http://www.baidu.com/&sms=5
http://news.baidu.com
https://www.hao123.com?src=from_pc
http://map.baidu.com
http://tieba.baidu.com/
https://haokan.baidu.com/?sfrom=baidu-top
http://image.baidu.com/
https://pan.baidu.com?from=1026962h
http://www.baidu.com/more/
https://top.baidu.com/board?platform=pc&sa=pcindex_entry
https://www.baidu.com/s?wd=丝路古道焕新机&sa=fyb_n_homepage&rsv_dl=fyb_n_homepage&from=super&cl=3&tn=baidutop10&fr=top10
00&rsv_idx=2&hisfilter=1
https://www.baidu.com/s?wd=每个画面都是丰收的中国&sa=fyb_n_homepage&rsv_dl=fyb_n_homepage&from=super&cl=3&tn=baidutop10&
fr=top1000&rsv_idx=2&hisfilter=1
https://www.baidu.com/s?wd=司法部原部长傅政华被判死缓&sa=fyb_n_homepage&rsv_dl=fyb_n_homepage&from=super&cl=3&tn=baiduto
p10&fr=top1000&rsv_idx=2&hisfilter=1
https://www.baidu.com/s?wd=网友吐槽海底捞小料涨到11元&sa=fyb_n_homepage&rsv_dl=fyb_n_homepage&from=super&cl=3&tn=baiduto
p10&fr=top1000&rsv_idx=2&hisfilter=1
https://www.baidu.com/s?wd=志愿者讨论性骚扰隔离女生?贵阳通报&sa=fyb_n_homepage&rsv_dl=fyb_n_homepage&from=super&cl=3&tn=
baidutop10&fr=top1000&rsv_idx=2&hisfilter=1
https://www.baidu.com/s?wd=新疆多地拍到巨大发光不明飞行物&sa=fyb_n_homepage&rsv_dl=fyb_n_homepage&from=super&cl=3&tn=bai
dutop10&fr=top1000&rsv_idx=2&hisfilter=1
http://home.baidu.com
http://ir.baidu.com
TimeOut Error
timed out
http://www.baidu.com/duty
http://help.baidu.com
https://e.baidu.com/?refer=1271
HTTPError       400     Bad Request
http://www.beian.gov.cn/portal/registerSystemInfo?recordcode=11000002000001
https://beian.miit.gov.cn
HTTPError       521
http://www.baidu.com/licence/
https://www.baidu.com/s?rtt=1&bsst=1&cl=2&tn=news
```

分析与反思：

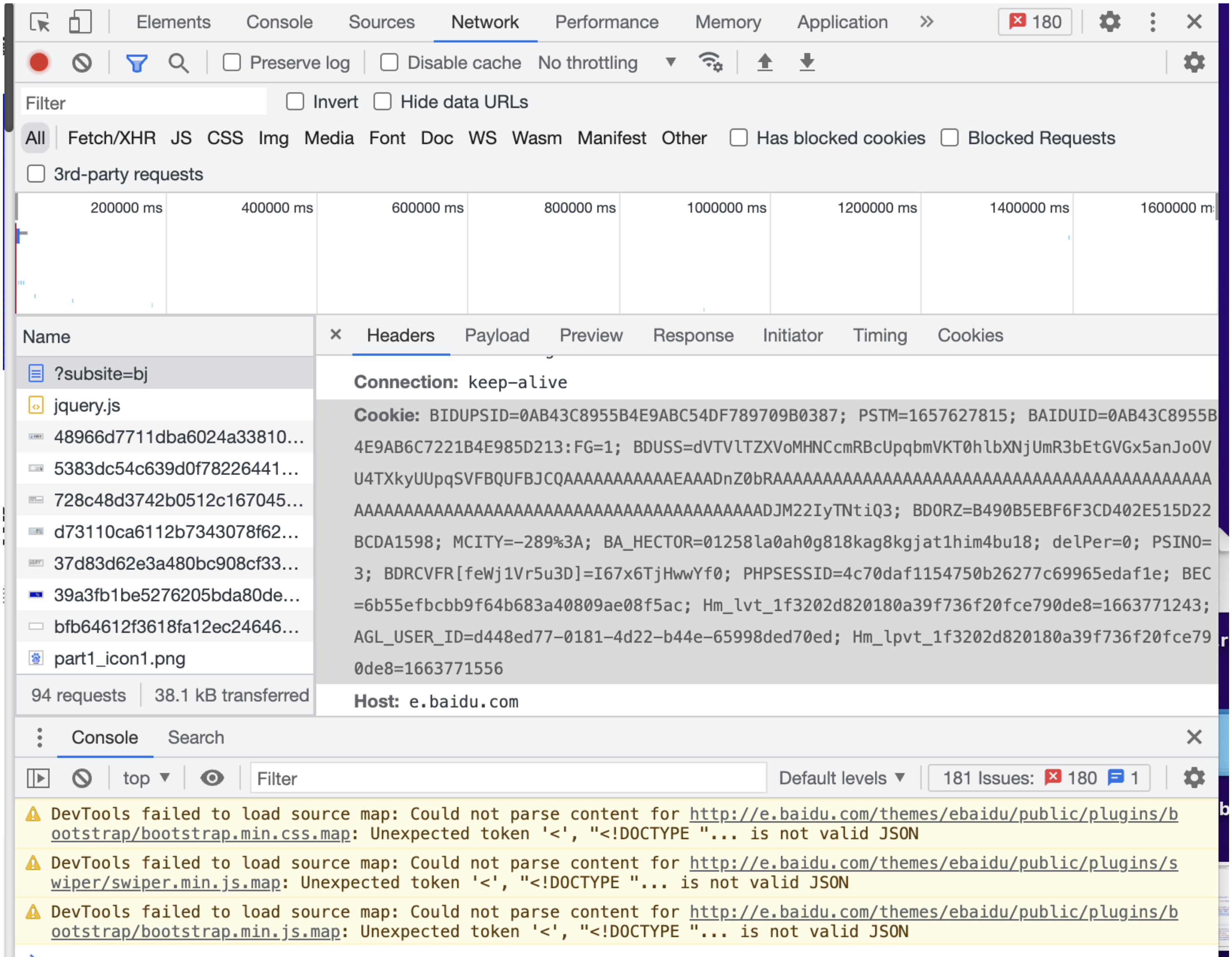
1.

```
socket.gaierror: [Errno 8] nodename nor servname provided, or not known
```

经查询，解决方法[2]与原因如下：
Mac OS系统上使用的一个问题，是Application无法定位到自己的ip地址而报错。部分代码需要通过获取主机名称来查询IP地址，而
当 `/private/etc/hosts` 文件中没有加入 `127.0.0.1 hostname` 时，就会有报错产生。

2.

在爬取
`urllib.error.HTTPError: HTTP Error 400: Bad Request`
可能是网站的反爬机制在起作用，400状态码代表 *Bad Request*, 可能是网站运行其 *JavaScript* 文件核实我们的请求头时发现错误，尝试手动添加cookie:




```
url = 'http://e.baidu.com/?subsite=bj'

cookie = 'BIDUPSID=0AB43C8955B4E9ABC54DF789709B0387; PSTM=1657627815; BAIDUID=0AB43C8955B4E9AB6C7221B4E985D213:FG=1; BDUSS=dVTVlTZXVoMHNCcmRBcUpqbmVKT0hlbXN

data = {"Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
"Accept-Encoding": "gzip, deflate",
"Accept-Language": "en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7",
"Cache-Control": "max-age=0",
"Host": "e.baidu.com",
"Upgrade-Insecure-Requests":"1",
"Connection": "keep-alive"}
data['User Agent'] = header
data["Cookie"] = cookie
data = bytes(parse.urlencode(cookie).encode())
opener = request.build_opener()
open_Request = request.Request(url, headers = data)
```

已经将所有请求头中的数据改为人为浏览时请求头的数据，但是仍得 *HTTP 400 Bad Request* 错误,原因是?

1.

遇到了出现十六进制乱码，中文无法显示的问题，经过查询：
得到的是 *UTF-8* 编码形式，但是数据类型是 *str* ,但并不是 *bytes* 类型的 *UTF-8* 编码，没法通过 `.decode(encoding = 'utf-8')(bytes->str)` 方法转换为中文。
第一种方法是利用 *b*'' 来转换为 *bytes* 对象,想到可以使用 `bytes()` 方法,但是 `bytes()` 方法只能一次只能处理单个字节， `.encode()` 方法也会产生如下错误，如下：

```
s = b'\xe9\x83\xad'
print(s)
ss = '\xe9\x83\xad'
print(bytes(ss, encoding='utf-8'))
print(ss.encode())
```

结果为：

```
b'\xe9\x83\xad'
b'\xc3\xa9\xc2\x83\xc2\xad'
b'\xc3\xa9\xc2\x83\xc2\xad'
```

```
s = '\xe9\x83\xad'
ss = bytes(s, encoding='utf-8').decode(encoding='utf-8')
print(ss)
```

```
s = b'\xe9\x83\xad'
print(s.decode(encoding='utf-8'))
print(str(s, 'utf-8'))
```

结果如下：

```
éƒ
郭
郭
```

解决方法：
利用 `.encode("raw_unicode_escape")` 以 *Unicode* 方法通过 '\x' 来进行分割，encode转换成 `<class bytes>` 再decode.

```
str.encode('raw_unicode_escape').decode()
```

兼容性更强的解决方法[3]：

但在正则表达式匹配上又遇到了问题：
结论是：以下方法是合法的 *regex* 匹配模式：

```
re.compile(r'(.+(\\x)+.+)+', re.DOTALL)
```

而以下这两种方法是不合法的 *regex* 匹配模式：

```
re.compile('(.+(\\x)+.+)+', re.DOTALL)
re.compile(r'(.+([\\x])+.+)+', re.DOTALL)
```

会报错如下：

```
re.error: incomplete escape \x at position 5
```

因为 `\x` 是一个转义字符，代表十六进制, 所以要在 *regex* 文本中特殊处理。如果 `re.compile(pat, s)` 的 pat 是以 raw literal string形式传入时， `\` 应该不会解释为转义字符而作为纯文本处理。但是另一个问题在调用 `re.match(pat, s)` 时产生，
当运行代码：

```
s = 'http://\x2d\xc3\xd3\xa2\xc6/'
pat = re.compile(r'(.+(\\x)+.+)+', re.DOTALL)
print(re.match(pat, s))
```

时结果如下：

```
None
```

这与我们的预期不符，原因其实是字符串s中仍含有转义字符 `\x` ,所以应该将这个字符串转换成 *raw string literal*, 所以可以调用 `.__repr()__` 方法或者用 `r''` 形式来将一个含转义字符的字符串当作 *raw string literal* 来处理。

与这个疑惑最相近的网上的例子在[4]，理解思路受其启发

```
/Users/liliu/Desktop/lab/test2.py:196: FutureWarning: Possible nested set at position 5
pat = re.compile(r'(.+([[\\x][\\X]%])+.+)+', re.DOTALL)
```

1. *User-Agent* 池和 *IP* 代理池
   运用 *User-Agent* 池和 *IP* 代理池可以防止服务器以访问频率和访问时间的分布规律来判断人与机，以更随机的方式来模拟不同用户来克服反爬机制。

User-Agent池参考[5]

4.拓展思考

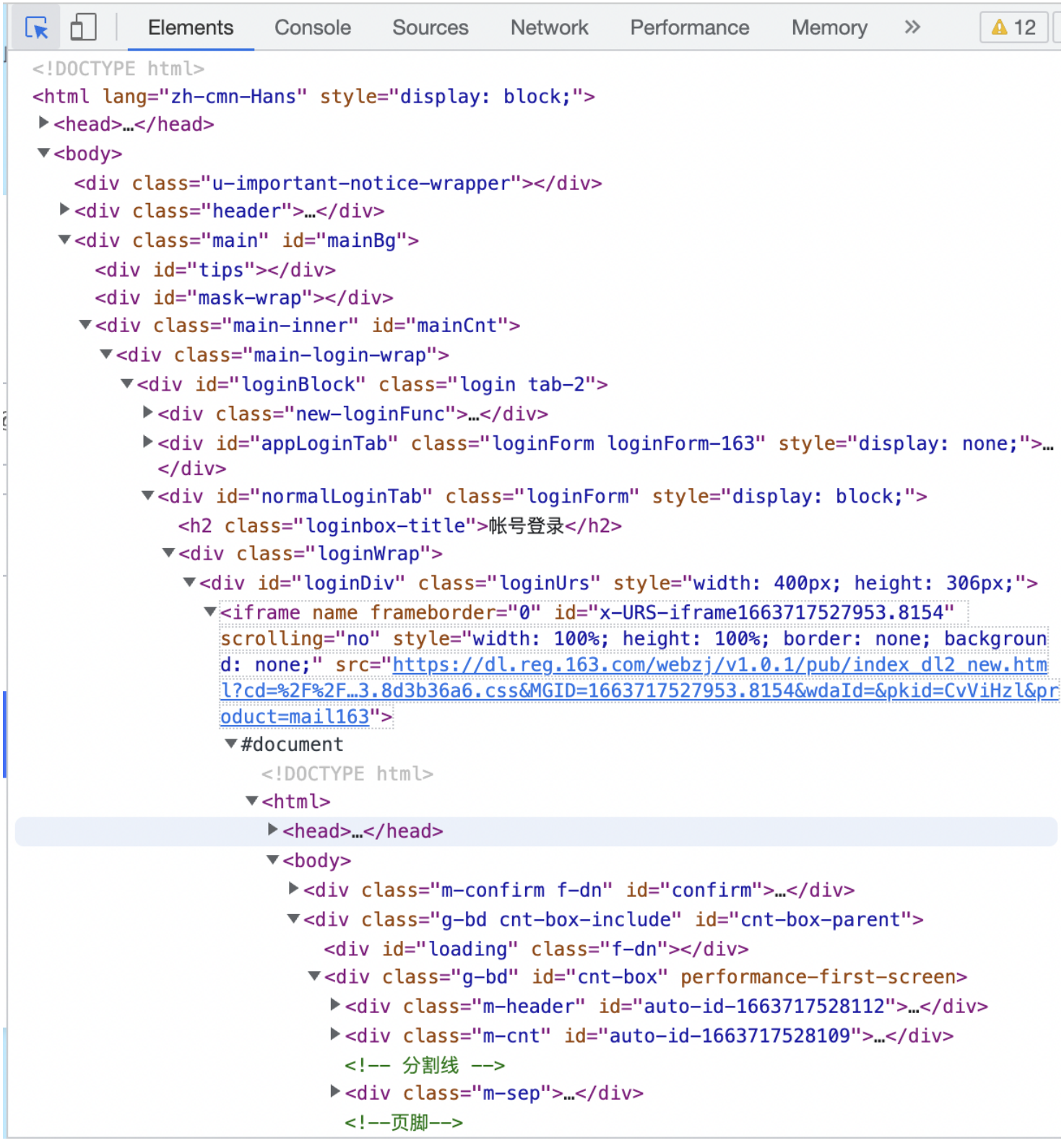
- 报错如
  下： `ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed certificate in certificat`
  是因为 *urllib3* 库中强制要求检查 SSL证书，SSL证书的检查可以保证目标网站的数据真实性，而所爬取URL是自签证书，所以报错。
  解决方法[6]:

```
import ssl

#  全局取消证书验证
<p class="mume-header " id="全局取消证书验证"></p>

ssl._create_default_https_context = ssl._create_unverified_context
```

- 在运行
  `login_url = 'https://mail.163.com/'` 时， 会报错如下：

```
HTTPError 302 The HTTP server returned a redirect error that would lead to an infinite loop.
The last 30x error message was:
Moved Temporarily Server: nginx
Date: Wed, 21 Sep 2022 00:20:17 GMT
Content-Type: text/html
Content-Length: 138
Connection: close
Location: https://mail.163.com/404_error.html
```

第一步推测：可能有 `<iframe></iframe>` 动态网页框架



1. https://www.runoob.com/http/http-status-codes.html ↩
2. https://blog.csdn.net/Zero_Muzi/article/details/119845560 ↩
3. https://blog.csdn.net/YungGuo/article/details/110197818 ↩
4. https://stackoverflow.com/questions/51970640/error-escaping-char-in-regex ↩
5. https://blog.csdn.net/weixin_48917089/article/details/125256415 ↩
6. https://www.cnblogs.com/lxmtx/p/12929905.html ↩