把执行的结果重定向到baidu.html这个文件中。python3中会出错.

python 7.1.4\_urllib2.py > baidu.html

在atom中安装pretty JSON插件, 扩展 > pretty JSON > pretty

pycahrm中 code > Reformat code, 但pycharm中中文使用unicoe编码.. code > reformat code还能格式化代码, 使其符合pep8规范.

# 第一部分 爬虫基础和requests

## 课程概要



## 第一天课程概要

**1、爬虫基础知识**

2、HTTP和HTTPS的复习

3、字符串的复习

**4、Requests的使用**

**5、Fiddle软件的使用**

## 爬虫概要

1、爬虫的应用场景

2、爬虫的概念

3、爬虫的分类

4、爬虫的工作流程

### 爬虫的定义

网络爬虫(又被称为网页蜘蛛，网络机器人) 就是模拟客户端发送网络请求，接收请求响应，一种按照一定的规则，自动地抓取互联网信息的程序。

只要是浏览器能做的事情，原则上，爬虫都能够做

### 爬虫的用途

12306抢票

12306可以用合作网站的内容进行爬取

网站上的投票

短信轰炸

短信轰炸, 很多多网站使用手机注册, 多个爬虫

weixin.sogou.com

可以爬取公众号的内容

### 爬虫的分类

通用爬虫: 通常指搜索引擎的爬虫

聚焦爬虫 : 针对特定网站的爬虫

### 通用搜索引擎工作原理

**想一想:**

**如果自己要实现一个和百度新闻一样的网站需要怎么做**

搜索引擎流程



搜索引擎需要有种子url地址, 种子url地址可以认为是起始url地址, 是一个搜索引擎公司的机密. 通过大数据分析通过什么样的网站才能更快的爬取到整个互联网的信息.

处理和分词

网站排名, 点击次数, 网站之间的引用关系, pagerank算法, 公开的算法. 友情链接, 评论中发表的url地址. 引用本网站的网站的权重.

### 聚焦爬虫流程



### 爬虫的概念

- 爬虫是模拟浏览器发送请求，获取响应

### 爬虫的流程

- url--->发送请求，获取响应--->提取数据---》保存

- 发送请求，获取响应--->提取url

#### 爬虫要根据当前url地址对应的响应为准

- 当前url地址的elements的内容和url的响应response的内容不一样

- 在检查元素中看到的内容是elements的内容

- 在network中点击某页面在右边的response中的内容才是response的内容

### 页面上的数据在哪里

- 当前url地址对应的响应中

- 其他的url地址对应的响应中

- 比如ajax请求中

- js生成的

- 部分数据在响应中

- 全部通过js生成

### 通用搜索引擎的局限性

通用搜索引擎所返回的网页里90%的内容无用。

图片、音频、视频多媒体的内容通用搜索引擎无能为力

不同用户搜索的目的不全相同，但是返回内容相同. 可以用户个性化推荐

现在的搜索引擎还有很多可以优化的地方, 可以自己实现一个搜索引擎,

### ROBOTS协议

Robots协议: 网站通过Robots协议告诉搜索引擎哪些页面可以抓取，哪些页面不能抓取。

例如:

https://www.taobao.com/robots.txt

## HTTP和HTTPS复习内容

1、HTTP和HTTPS

2、HTTP的请求过程

3、HTTP的请求形式

4、HTTP的常见请求头

5、GET和POST

6、响应状态码



### HTTP和HTTPS

HTTP

超文本传输协议

默认端口号:80

HTTPS

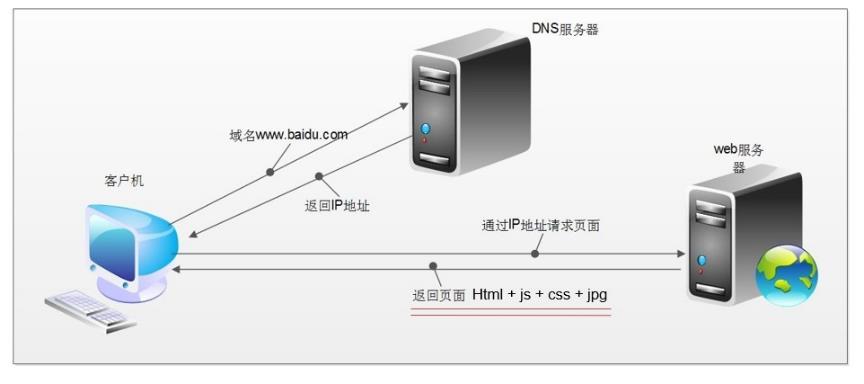
HTTP + SSL(安全套接字层)

默认端口号: 443`

HTTPS比HTTP更安全，但是性能更低

### 浏览器发送HTTP请求的过程

浏览器渲染出来的页面和爬虫请求的页面并不一样, 爬虫只发送一次请求, 页面中的图片, js等都不会请求到, 而浏览器会发送多个请求.



### url的形式

形式 scheme://host[:port#]/path/…/[?query-string][#anchor]

scheme: 协议(例如: http, https, ftp)

host: 服务器的IP地址或者域名

port: 服务器的端口(如果是走协议默认端口，80 or 443)

path: 访问资源的路径

query-string: 参数，发送给http服务器的数据

anchor: 锚(跳转到网页的指定锚点位置)

http://localhost:4000/file/part01/1.2.html

http://item.jd.com/11936238.html#product-detail

有没有锚点的url是同一个响应, 不会重新发送请求.

### HTTP请求的形式



GET / HTTP/1.1 请求行

Host: [www.baidu.com](http://www.baidu.com) 请求头

Connection: keep-alive

Cache-Control: max-age=0

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36

Accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, image/apng, \*/\*;q=0.8

DNT: 1

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US, en;q=0.9, zh-CN;q=0.8, zh;q=0.7

Cookie: BAIDUID=65A0E33156EFF5425D9EEB3EEE6FDD42:FG=1; BIDUPSID=65A0E33156EFF5425D9EEB3EEE6FDD42; H\_PS\_PSSID=1436\_21089\_18560\_22073; BDORZ=B490B5EBF6F3CD402E515D22BCDA1598; BD\_CK\_SAM=1; BDRCVFR[feWj1Vr5u3D]=I67x6TjHwwYf0; BD\_HOME=1; PSINO=7

### HTTP常见请求头

1. Host (主机和端口号)

2. Connection (链接类型)

3. Upgrade-Insecure-Requests (升级为HTTPS请求)

4. User-Agent (浏览器名称)

5. Accept (传输文件类型)

6. Referer (页面跳转处)

7. Accept-Encoding(文件编解码格式)

8. Cookie (Cookie)

9. x-requested-with :XMLHttpRequest (是Ajax 异步请求)

### 常见的请求方法

GET

POST

问题: GET方法和POST方法有什么区别呢？

### 响应状态码(status code)

200: 成功

302: 临时转移至新的url

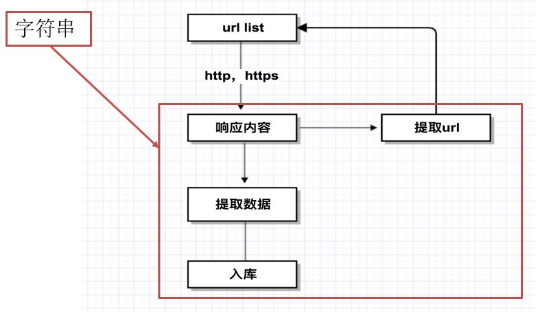
307: 临时转移至新的url

404: not found

500: 服务器内部错误

### 字符串类型的区别和转化

问题: 为什么要掌握python3字符串的相关知识



### str类型和bytes类型

bytes: 二进制

互联网上数据的都是以二进制的方式传输的

str: unicode的呈现形式

### Unicode UTF8 ASCII的补充

字符(Character)是各种文字和符号的总称，包括各国家文字、标点符号、图形符号、数字等

字符集(Character set)是多个字符的集合

字符集包括: ASCII字符集、GB2312字符集、GB18030字符集、Unicode字符集等

ASCII编码是1个字节，而Unicode编码通常是2个字节。

UTF-8是Unicode的实现方式之一，UTF-8是它是一种变长的编码方式，可以是1，2，3个字节

### str和bytes之间如何转化

str 使用encode方法转化为 bytes

bytes通过decode转化为str

编码方式解码方式必须一样，否则就会出现乱码

### bytes和str的区别和相互转换

```python

In [1]: a = "传特别是"

In [2]: type(a)

Out[2]: str

In [3]: b = a.encode()

In [4]: b

Out[4]: b'\xe4\xbc\xa0\xe7\x89\xb9\xe5\x88\xab\xe6\x98\xaf'

In [5]: type(b)

Out[5]: bytes

In [6]: b.decode()

Out[6]: '传特别是'

In [7]: b.decode("utf-8")

Out[7]: '传特别是'

```

## Requests 使用入门

问题: 为什么要学习requests，而不是urllib？

requests的底层实现就是urllib

requests在python2 和python3中通用，方法完全一样

requests简单易用

requests能够自动帮助我们解压(gzip压缩的等)网页内容

### requests的作用

作用: 发送网络请求，返回响应数据

中文文档 API:  <http://docs.python-requests.org/zh_CN/latest/index.html>

需要解决的问题: 如何使用requests来发送网络请求？

开源地址：<https://github.com/kennethreitz/requests>

### 发送简单的请求

response = requests.get(url)

response的常用方法:

* response.text
* respones.content
* response.status\_code
* response.request.headers
* response.headers, 一般只关注响应头中的Set-Cookie, 服务器还可以通过js来设置cookie
* response.encoding 获取编码方式
* response.encoding("utf-8) 设定编码方式
* response.request.url 请求的url地址
* response.url 响应的url地址

response.status\_code == 200时只能表示请求成功了, 但不一定表示我们请求的url地址成功了., 有时候会重定向到其它网页中.

assert response.status\_code == 200

假设返回的响应码为200, 如果不是200, 就会报错.

>>> import requests

>>> response = requests.get("https://www.hellobi.com/23234")

>>> response.status\_code

404

>>> assert response.status\_code == 200

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AssertionError

try:

assert response.status\_code == 200

except AssertionError as e:

print("111")

print(e)

print("222")

111

222

实例: 通过requests向百度首页发送请求，获取百度首页的数据

In [10]: import requests

In [11]: response = requests.get("http://www.baidu.com/")

In [12]: response

Out[12]: <Response [200]>

In [13]: response.text

In [14]: response.encoding

Out[14]: 'ISO-8859-1'

In [15]: response.encoding = "utf-8"

In [16]: response.

import requests

r = requests.get('https://api.github.com/events')

*# 这里是r的text属性,* *在定义类的方法和属性时, 尽量分别以动作的关键词和静态的关键词来定义*

r.text

*# u'[{"repository":{"open\_issues":0, "url":"https://github.com/...}]*

*# Requests 会自动解码来自服务器的内容。大多数 unicode 字符集都能被无缝地解码。*

*# 请求发出后，Requests 会基于 HTTP 头部对响应的编码作出有根据的推测。当你访问 r.text 之时，Requests 会使用其推测的文本编码。你可以找出 Requests 使用了什么编码，并且能够使用 r.encoding 属性来改变它:*

*# r.encoding获得requests模块的解码方式*

r.encoding

*# 'utf-8'*

r.encoding = 'ISO-8859-1'

### response.text 和response.content的区别

* response.text
  1. 类型：str
  2. 解码类型： 根据HTTP 头部对响应的编码作出有根据的推测，推测的文本编码
  3. 如何修改编码方式：response.encoding=”gbk”
* response.content
  1. 类型：bytes
  2. 解码类型： 没有指定
  3. 如何修改编码方式：response.content.deocde(“utf8”)

使用response.text 时，Requests 会基于 HTTP 响应的文本编码自动解码响应内容，大多数 Unicode 字符集都能被无缝地解码。

使用response.content 时，返回的是服务器响应数据的原始二进制字节流，可以用来保存图片等二进制文件。

### requests中解决编解码的方法

- response.content.decode()

- 默认使用utf-8的格式进行解码

- response.content.decode("gbk")

- response.text

案例

### 案例: 请求并保存一张图片

```python

# coding=utf-8

**import** **requests**

#发送请求

response = requests.get("https://www.baidu.com/img/bd\_logo1.png")

# 写入response.text的内容

with open("baidu.html", "w", encoding="utf-8") as f:

f.write(response.text)

# 写入response.content的内容

**with** open("a.png", "wb") **as** f:

f.write(response.content)

```

### 发送带header的请求

为什么请求需要带上header？

模拟浏览器，欺骗服务器，获取和浏览器一致的内容

* header的形式: 字典
* headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36"}
* 用法: requests.get(url, headers=headers)

### 发送带参数的请求

什么叫做请求参数:

列1: http://www.webkaka.com/tutorial/server/2015/021013/☓

例2: https://www.baidu.com/s?wd=python&c=b

* 参数的形式: 字典
* kw = {'wd':'长城'}
* 用法: requests.get(url, params=kw)

### 字符串格式化的另一种方式

```python

>>> "传{}智播客".format(1)

'传1智播客'

>>> "传{}智播客".format("a")

'传a智播客'

>>> "传{}智播客".format([1, 2, 3])

'传[1, 2, 3]智播客'

>>> "传{}智播客".format({"a":2})

"传{'a': 2}智播客"

>>> "传{}智{}播{}客".format({"a":2}, 2, 3)

"传{'a': 2}智2播3客"```

简单百度搜索

# coding=utf-8

**import** **requests**

headers = {"User-Agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

p = {"wd":"传智播客"}

# url\_temp = "https://www.baidu.com/s?"

# ?是可有可无的, 在使用requests发送请求时会自动加上.

url\_temp = "https://www.baidu.com/s"

*# params 接收一个字典或者字符串的查询参数，字典类型自动转换为url编码，不需要urlencode()*

r = requests.get(url\_temp, headers=headers, params=p)

*# 查看响应的状态码*

**print**(r.status\_code)

*# 查看请求的url地址*

**print**(r.request.url)

*# 查看响应的url地址*

print(r.url)

*# # 查看响应头部字符编码*

print(response.encoding)

*# 查看响应内容，response.text 返回的是Unicode格式的数据*

print(response.text)

*# 查看响应内容，response.content返回的字节流数据*

*# print(response.content)*

# 搜索"传智播客"

url = "https://www.baidu.com/s?wd={}".format("传智播客")

r = requests.get(url, headers=headers)

**print**(r.status\_code)

**print**(r.request.url)

### url编码

- `https://www.baidu.com/s?wd=%E4%BC%A0%E6%99%BA%E6%92%AD%E5%AE%A2`

- `https://www.baidu.com/s?wd=传智播客`

百度翻译接口

# coding=utf-8

**import** **requests**

headers = {

# "Accept":"\*/\*",

# "Accept-Encoding":"gzip, deflate",

# "Accept-Language":"zh-CN, zh;q=0.9, en;q=0.8, ru;q=0.7, zh-TW;q=0.6",

# "Connection":"keep-alive",

# "Content-Type":"application/x-www-form-urlencoded",

# "Host":"fanyi.baidu.com",

# "Origin":"http://fanyi.baidu.com",

# "Referer":"http://fanyi.baidu.com/",

"User-Agent":"Mozilla/5.0 (Linux; Android 5.1.1; Nexus 6 Build/LYZ28E) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Mobile Safari/537.36",

# "X-Requested-With":"XMLHttpRequest",

# "Cookie":"BAIDUID=..."

}

data = {"query":"你好啊",

"from":"zh",

"to":"en"

}

r = requests.post("http://fanyi.baidu.com/basetrans", headers=headers, data=data)

**print**(r.content.decode())

### 动手尝试

1、获取新浪首页，查看response.text 和response.content.decode()的区别

2、实现任意贴吧的爬虫，保存网页到本地

#### 字典推导式

### 字典推导式，列表推导式

```python

cookies="anonymid=j3jxk555-nrn0wh; \_r01\_=1; \_ga=GA1.2.1274811859.1497951251; \_de=BF09EE3A28DED52E6B65F6A4705D973F1383380866D39FF5; ln\_uact=mr\_mao\_hacker@163.com; depovince=BJ; jebecookies=54f5d0fd-9299-4bb4-801c-eefa4fd3012b|||||; JSESSIONID=abcI6TfWH4N4t\_aWJnvdw; ick\_login=4be198ce-1f9c-4eab-971d-48abfda70a50; p=0cbee3304bce1ede82a56e901916d0949; first\_login\_flag=1; ln\_hurl=http://hdn.xnimg.cn/photos/hdn421/20171230/1635/main\_JQzq\_ae7b0000a8791986.jpg; t=79bdd322e760beae79c0b511b8c92a6b9; societyguester=79bdd322e760beae79c0b511b8c92a6b9; id=327550029; xnsid=2ac9a5d8; loginfrom=syshome; ch\_id=10016; wp\_fold=0"

cookies = {i.split("=")[0]:i.split("=")[1] for i in cookies.split("; ")}

{'anonymid': 'j3jxk555-nrn0wh',

'\_r01\_': '1',

'\_ga': 'GA1.2.1274811859.1497951251',

'\_de': 'BF09EE3A28DED52E6B65F6A4705D973F1383380866D39FF5',

'ln\_uact': 'mr\_mao\_hacker@163.com',

'depovince': 'BJ',

'jebecookies': '54f5d0fd-9299-4bb4-801c-eefa4fd3012b|||||',

'JSESSIONID': 'abcI6TfWH4N4t\_aWJnvdw',

'ick\_login': '4be198ce-1f9c-4eab-971d-48abfda70a50',

'p': '0cbee3304bce1ede82a56e901916d0949',

'first\_login\_flag': '1',

'ln\_hurl': 'http://hdn.xnimg.cn/photos/hdn421/20171230/1635/main\_JQzq\_ae7b0000a8791986.jpg',

't': '79bdd322e760beae79c0b511b8c92a6b9',

'societyguester': '79bdd322e760beae79c0b511b8c92a6b9',

'id': '327550029',

'xnsid': '2ac9a5d8',

'loginfrom': 'syshome',

'ch\_id': '10016',

'wp\_fold': '0'}

```

In [39]: {i:i+10 for i in range(10)}

Out[39]: {0: 10, 1: 11, 2: 12, 3: 13, 4: 14, 5: 15, 6: 16, 7: 17, 8: 18, 9: 19}

In [40]: {i:"v" for i in range(10)}

Out[40]:

{0: 'v',

1: 'v',

2: 'v',

3: 'v',

4: 'v',

5: 'v',

6: 'v',

7: 'v',

8: 'v',

9: 'v'}

#### 列表推导式

```python

url\_temp = "https://tieba.baidu.com/f?kw=&ie=utf-8&pn={}"

[url\_temp.format(i \* 50) for i in range(1000)]

```

## Requests深入

发送POST请求

使用代理

处理cookies session

### 发送POST请求

**GET和POST请求的区别**

GET：请求的url会附带查询参数，查询参数在QueryString里保存

POST：请求的url不带参数，查询参数在Form表单里保存

<http://fanyi.youdao.com>使用的是post的请求

需要找到POST请求的地址, POST请求的headers信息, POST请求的data信息.

**在Fiddler或Chrome中查看post请求的数据和post请求的地址**

Fiddler > Raw >

WebForms就显示出查询的Name和Value值。

在Fiddler > WebForm中的Body部分显示出来，也会在Raw中的最后一行显示出来。

**chrome查看header信息**

POST <http://fanyi.youdao.com/translate_o?smartresult=dict&smartresult=rule&sessionFrom=null> HTTP/1.1

Host: fanyi.youdao.com

Connection: keep-alive

Content-Length: 231

# 表示返回的数据格式为json, 如果是以ajax方式加载的数据, 绝大部分都是json的格式

Accept: application/json, text/javascript, \*/\*; q=0.01

Origin: <http://fanyi.youdao.com>

# 表示使用ajax的方式进行加载的

X-Requested-With: XMLHttpRequest

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.81 Safari/537.36

Content-Type: application/x-www-form-urlencoded; charset=UTF-8

DNT: 1

Referer: <http://fanyi.youdao.com/>

Accept-Encoding: gzip, deflate

Accept-Language: en-US, en;q=0.8, zh-CN;q=0.6, zh;q=0.4

**post请求的地址**

POST <http://fanyi.youdao.com/translate_o?smartresult=dict&smartresult=rule&sessionFrom=null> HTTP/1.1

**post请求的headers信息. 使用sublime的正则匹配来去掉冒号后面的空格, 并加上""**

find: ^(.\*):\s(.\*)$

replace: "\1":"\2",

"Host":"fanyi.youdao.com",

"Connection":"keep-alive",

"Content-Length":"231",

"Accept":"application/json, text/javascript, \*/\*; q=0.01",

"Origin":"http://fanyi.youdao.com",

"X-Requested-With":"XMLHttpRequest",

"User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.81 Safari/537.36",

"Content-Type":"application/x-www-form-urlencoded; charset=UTF-8",

"DNT":"1",

"Referer":"http://fanyi.youdao.com/",

"Accept-Encoding":"gzip, deflate",

"Accept-Language":"en-US, en;q=0.8, zh-CN;q=0.6, zh;q=0.4",

**POST请求的data信息, 在chrome中查看到的form data信息., 即向服务器提交的data信息, 使用urllib.request.Request构建的request对象中如果有data信息, 就是post的请求.**

i=%E4%BD%A0%E5%A5%BD%EF%BC%8Cpython&from=AUTO&to=AUTO&smartresult=dict&client=fanyideskweb&salt=1493768163104&sign=4d6f9cedeff67bea71cbcbef20459aff&doctype=json&version=2.1&keyfrom=fanyi.web&action=FY\_BY\_CLICKBUTTON&typoResult=true

**使用sublime正则替换转换格式把request的内容转化为字典格式**

在sublimetext的查找替换中ctrl+Enter输入回车进行替换

^(.\*):\s(.\*)$

\s表示匹配空格

"i":"%E4%BD%A0%E5%A5%BD%EF%BC%8Cpython",

"from":"AUTO",

"to":"AUTO",

"smartresult":"dict",

"client":"fanyideskweb",

"salt":"1493768163104",

"sign":"4d6f9cedeff67bea71cbcbef20459aff",

"doctype":"json",

"version":"2.1",

"keyfrom":"fanyi.web",

"action":"FY\_BY\_CLICKBUTTON",

"typoResult":"true",

在Raw的第一行中也会看到POST请求的网址

在response下面的JSON中也会看到返回的json格式的内容。

在Response内容中的RAW中也能看到返回的内容。

HTTP/1.1 200 OK

Server: nginx

Date: Wed, 26 Apr 2017 23:25:24 GMT

Content-Type: application/json; charset=utf-8

Connection: keep-alive

Vary: Accept-Encoding

Content-Length: 190

{"translateResult":[[{"tgt":"python", "src":"python"}]], "errorCode":0, "type":"en2zh-CHS", "smartResult":{"entries":["", "n. 巨蟒；大蟒\r\n", "n. (法) 皮东(人名) \r\n"], "type":1}}

使用www.json.cn来转换json格式

{

"translateResult":[

[

{

"tgt":"python",

"src":"python"

}

]

],

"errorCode":0,

"type":"en2zh-CHS",

"smartResult":{

"entries":[

"",

"n. 巨蟒；大蟒 ",

"n. (法) 皮东(人名) "

],

"type":1

}

}

**模拟网页的POST请求实现有道字典接口**

哪些地方我们会用到POST请求:

* 登录注册( POST 比 GET 更安全)
* 需要传输大文本内容的时候( POST 请求对数据长度没有要求)

所以同样的，我们的爬虫也需要在这两个地方回去模拟浏览器发送post请求

用法:

response = requests.post("http://www.baidu.com/", data = data, headers=headers)

data 的形式: 字典

下面我们通过百度翻译的例子看看post请求如何使用

使用检查元素查看百度翻译的请求和响应.

langdetect 检测翻译语言

v2transapi 翻译的内容, 可以使用过滤, 搜索v2, 只显示v2的内容

因为无法获取发送的data中的sign和token, 所以无法使用pc端的百度翻译

# coding=utf-8

**import** **requests**

headers = {

"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

data = {

"from":"en",

"to":"zh",

"query":"hola",

"transtype":"translang",

"simple\_means\_flag":"3",

# 在网页中没有找到sign的关键词, 所以应该是js动态生成的, 因为有这两个验证, 所以无法使用这个headers从百度翻译获取内容

"sign":"372549.85108",

"token":"e89a8f037aac1b51a86cbc82356949d"

}

post\_url = "http://fanyi.baidu.com/v2transapi"

r = requests.post(post\_url, data=data, headers=headers)

**print**(r.content.decode())

使用百度翻译的手机页面来完成翻译的请求.

# test.py

import sys

print(sys.argv)

使用命令行执行

python test.py

# ['test.py']

python test.py 1 2 3

# ['test.py', '1', '2', '3']

# coding=utf-8

**import** **requests**

**import** **json**

**import** **sys**

# 使用命令行参数实现输入

query\_string = input("请输入查询关键词:")

# 使用input实现用户输入

query\_string = sys.argv[1]

headers = {"User-Agent":"Mozilla/5.0 (Linux; Android 5.1.1; Nexus 6 Build/LYZ28E) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Mobile Safari/537.36"}

post\_data = {

"query":query\_string,

"from":"zh",

"to":"en",

}

# 使用手机翻译页面来完成翻译的接口

post\_url = "http://fanyi.baidu.com/basetrans"

r = requests.post(post\_url, data=post\_data, headers=headers)

# print(r.content.decode())

# 把json转换为字典

dict\_ret = json.loads(r.content.decode())

ret = dict\_ret["trans"][0]["dst"]

**print**("result is :", ret)

python 04\_fanyi.py 人生

python 04\_fanyi.py 你好你好

想要实现输入以下内容实现翻译

fanyi 人生

使用alias来实现

linux中emacs ~/.bashrc

mac中 emacs ~/.zbashrc

在最后添加上一行

alias fanyi="python /User/zdw/python19/day02/code/04\_baidu\_fanyi\_01.py"

source ~/.bashrc

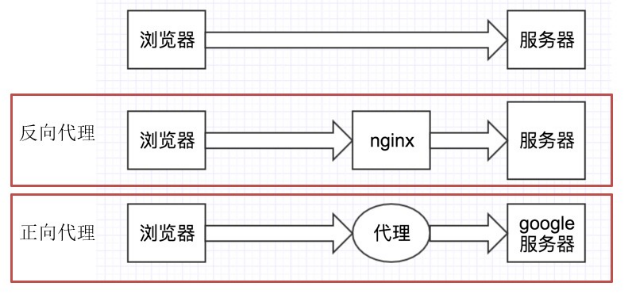
fanyi 天气真好

想要实现翻译语言的自动化检测, 可以参考浏览器的实现方式, 通过发送请求的方式来完成. landetect

### 使用代理

问题: 为什么爬虫需要使用代理？

* 让服务器以为不是同一个客户端在请求
* 防止我们的真实地址被泄露，防止被追究



用法: requests.get("http://www.baidu.com", proxies = proxies)

proxies的形式: 字典

proxies = {

"http": "http://12.34.56.79:9527",

"https": "https://12.34.56.79:9527",

}

### 使用代理ip

- 准备一堆的ip地址，组成ip池，随机选择一个ip来时用

- 如何随机选择代理ip，让使用次数较少的ip地址有更大的可能性被用到

- {"ip":ip, "times":0}

- [{}, {}, {}, {}, {}], 对这个ip的列表进行排序，按照使用次数进行排序

- 选择使用次数较少的10个ip，从中随机选择一个

- 检查ip的可用性

- 可以使用requests添加超时参数，判断ip地址的质量

- 在线代理ip质量检测的网站

简单的代理

# coding=utf-8

**import** **requests**

proxies = {"http":"http://163.177.151.23:80"}

headers = {

"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

r = requests.get("http://www.baidu.com", proxies=proxies, headers=headers)

**print**(r.status\_code)

### 使用ProxyHandler向opener添加代理

学习方法定义代理开关

**import** urllib.request, urllib.parse

*# 构建了两个代理Handler，一个有代理IP，一个没有代理IP*

httpproxy\_handler = urllib.request.ProxyHandler({**"http"** : **"61.191.173.31:808"**})

nullproxy\_handler = urllib.request.ProxyHandler({})

*#定义一个代理开关*

proxySwitch = **True**

*# 通过 urllib.request.build\_opener()方法使用这些代理Handler对象，创建自定义opener对象*

*# 根据代理开关是否打开，使用不同的代理模式*

**if** proxySwitch:

opener = urllib.request.build\_opener(httpproxy\_handler)

**else**:

opener = urllib.request.build\_opener(nullproxy\_handler)

request = urllib.request.Request(**"http://www.baidu.com/"**)

*# 1. 如果这么写，只有使用opener.open()方法发送请求才使用自定义的代理，而urlopen()则不使用自定义代理。*

response = opener.open(request)

*# 2. 如果这么写，就是将opener应用到全局，之后所有的，不管是opener.open()还是urlopen() 发送请求，都将使用自定义代理。*

*# urllib.request.install\_opener(opener)*

*# response = urllib.parse.urlopen(request)*

print(response.read().decode(**"utf-8"**))

<html>

<head><meta http-equiv="Content-Type" content="text/html; charset=utf-8" /></head>

<body>

<h1>Unauthorized ...</h1>

<h2>

IP Address: 180.168.188.185<br>

MAC Address: <br>

Server Time: 2017-05-03 09:36:25<br>

Auth Result: 无效用户.

</h2>

</body>

</html>

**from** **requests** **import** Request, Session

proxies = {"http":"222.94.151.100:808"}

headers = {"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

url = "http://www.baidu.com/"

s = Session()

req = Request('GET', url, headers=headers)

prepped = s.prepare(req)

prepped = s.prepare\_request(req)

proxy\_switch = True

**if** proxy\_switch:

resp = s.send(prepped,

proxies=proxies,

)

**else**:

resp = s.send(prepped,

proxies=None,

)

**print**(resp.status\_code)

### 随机选择代理地址

**import** urllib.request, random

proxy\_list = [

{**"http"** : **"124.88.67.81:80"**},

{**"http"** : **"124.88.67.81:80"**},

{**"http"** : **"124.88.67.81:80"**},

{**"http"** : **"124.88.67.81:80"**},

{**"http"** : **"124.88.67.81:80"**}

]

*# 随机选择一个代理*

proxy = random.choice(proxy\_list)

*# 使用选择的代理构建代理处理器对象*

httpproxy\_handler = urllib.request.ProxyHandler(proxy)

opener = urllib.request.build\_opener(httpproxy\_handler)

request = urllib.request.Request(**"http://www.baidu.com/"**)

response = opener.open(request)

print(response.read())

import requests

import random

proxy\_list = [

{"http" : "124.88.67.81:80"},

{"http" : "124.88.67.81:80"},

{"http" : "124.88.67.81:80"},

{"http" : "124.88.67.81:80"},

{"http" : "124.88.67.81:80"}

]

url = "http://www.baidu.com/"

*# 随机选择一个代理*

proxy = random.choice(proxy\_list)

response = requests.get(url, proxies = proxy)

print(response.status\_code)

一般要写一个文件，把代理放进去，然后引入这个文件就行了。

### requests处理cookies session

### cookie和session区别:

* cookie数据存放在客户的浏览器上，session数据放在服务器上。
* cookie不是很安全，别人可以分析存放在本地的cookie并进行cookie欺骗。
* session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能。
* 单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

在 requests 里，session对象是一个非常常用的对象，这个对象代表一次用户会话：从客户端浏览器连接服务器开始，到客户端浏览器与服务器断开。

会话能让我们在跨请求时候保持某些参数，比如在同一个 Session 实例发出的所有请求之间保持 cookie 。

### 爬虫处理cookie和session

带上cookie、session的好处:

能够请求到登录之后的页面

带上cookie、session的弊端:

一套cookie和session往往和一个用户对应

请求太快，请求次数太多，容易被服务器识别为爬虫

不需要cookie的时候尽量不去使用cookie

但是为了获取登录之后的页面，我们必须发送带有cookies的请求

cookeis详解

**import** **requests**

url = 'http://www.hellobi.com/'

headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36"}

r = requests.get(url, headers=headers)

r.cookies

*# <RequestsCookieJar[Cookie(version=0, name='XSRF-TOKEN', value='eyJpdiI6IkJFN3grcE1PbFwvN0Npa0k2U1lmV3ZnPT0iLCJ2YWx1ZSI6ImRIWU1wWjZJcDBrYzU4OURpZUFzUW9lUFFRYUFkQmJjNWV0RzdYUWtkdzQrc09zS20rclppUmhUQlBWY1FzeWkwWGhVSUtTSitDdkhDXC9cL095c29kS3c9PSIsIm1hYyI6ImIyZjY0MGFkNzAwZTUyZmVlMzFjM2FmMjYyYWM1ZTc2MWYwOTJjYmI3MTQ1YWExMjI5MmU1MTczNTdhOTQ0ODUifQ%3D%3D', port=None, port\_specified=False, domain='.hellobi.com', domain\_specified=True, domain\_initial\_dot=True, path='/', path\_specified=True, secure=False, expires=1528033096, discard=False, comment=None, comment\_url=None, rest={}, rfc2109=False), Cookie(version=0, name='laravel\_session', value='eyJpdiI6IjU3NkhKUHl2NUhKOThIMHFQM2NTVkE9PSIsInZhbHVlIjoiRmtWUkFDVUpBWTNxeVUzQmxVNW0za3VRVXI0ZnpoR3AzcXJPVU5VU0tTV1J4VHZrbHJZXC9VbUpUTG11S0ZzT0Qwd3NnS0pkRU4xWHZVbG9wZVhqS0RBPT0iLCJtYWMiOiI4YzBlMTY5NTUxZjA4MzlmZmRiODhiNDY4ODhkYjMyNzRhNmE4YzJlZmExZmEzZDFlY2QxZWNhZjg1YzgyOGJhIn0%3D', port=None, port\_specified=False, domain='.hellobi.com', domain\_specified=True, domain\_initial\_dot=True, path='/', path\_specified=True, secure=False, expires=1528033096, discard=False, comment=None, comment\_url=None, rest={'HttpOnly': None}, rfc2109=False)]>*

*# r.cookies是一个类字典的对象, 可以使用字典的所有方法*

# 把cookies转换为字典格式

r.cookies.get\_dict()

{'XSRF-TOKEN': 'eyJpdiI6IkJFN3grcE1PbFwvN0Npa0k2U1lmV3ZnPT0iLCJ2YWx1ZSI6ImRIWU1wWjZJcDBrYzU4OURpZUFzUW9lUFFRYUFkQmJjNWV0RzdYUWtkdzQrc09zS20rclppUmhUQlBWY1FzeWkwWGhVSUtTSitDdkhDXC9cL095c29kS3c9PSIsIm1hYyI6ImIyZjY0MGFkNzAwZTUyZmVlMzFjM2FmMjYyYWM1ZTc2MWYwOTJjYmI3MTQ1YWExMjI5MmU1MTczNTdhOTQ0ODUifQ%3D%3D',

'laravel\_session': 'eyJpdiI6IjU3NkhKUHl2NUhKOThIMHFQM2NTVkE9PSIsInZhbHVlIjoiRmtWUkFDVUpBWTNxeVUzQmxVNW0za3VRVXI0ZnpoR3AzcXJPVU5VU0tTV1J4VHZrbHJZXC9VbUpUTG11S0ZzT0Qwd3NnS0pkRU4xWHZVbG9wZVhqS0RBPT0iLCJtYWMiOiI4YzBlMTY5NTUxZjA4MzlmZmRiODhiNDY4ODhkYjMyNzRhNmE4YzJlZmExZmEzZDFlY2QxZWNhZjg1YzgyOGJhIn0%3D'}

# 以元组形式输出, 一个元组中保存一对键值, 每个元组中的键值分别为元组的2个元素r.cookies.items()

[('XSRF-TOKEN',

'eyJpdiI6IkJFN3grcE1PbFwvN0Npa0k2U1lmV3ZnPT0iLCJ2YWx1ZSI6ImRIWU1wWjZJcDBrYzU4OURpZUFzUW9lUFFRYUFkQmJjNWV0RzdYUWtkdzQrc09zS20rclppUmhUQlBWY1FzeWkwWGhVSUtTSitDdkhDXC9cL095c29kS3c9PSIsIm1hYyI6ImIyZjY0MGFkNzAwZTUyZmVlMzFjM2FmMjYyYWM1ZTc2MWYwOTJjYmI3MTQ1YWExMjI5MmU1MTczNTdhOTQ0ODUifQ%3D%3D'),

('laravel\_session',

'eyJpdiI6IjU3NkhKUHl2NUhKOThIMHFQM2NTVkE9PSIsInZhbHVlIjoiRmtWUkFDVUpBWTNxeVUzQmxVNW0za3VRVXI0ZnpoR3AzcXJPVU5VU0tTV1J4VHZrbHJZXC9VbUpUTG11S0ZzT0Qwd3NnS0pkRU4xWHZVbG9wZVhqS0RBPT0iLCJtYWMiOiI4YzBlMTY5NTUxZjA4MzlmZmRiODhiNDY4ODhkYjMyNzRhNmE4YzJlZmExZmEzZDFlY2QxZWNhZjg1YzgyOGJhIn0%3D')]

# 返回cookies的键

r.cookies.keys()

['XSRF-TOKEN', 'laravel\_session']

# 返回cookies的值

r.cookies.values()

['eyJpdiI6IkJFN3grcE1PbFwvN0Npa0k2U1lmV3ZnPT0iLCJ2YWx1ZSI6ImRIWU1wWjZJcDBrYzU4OURpZUFzUW9lUFFRYUFkQmJjNWV0RzdYUWtkdzQrc09zS20rclppUmhUQlBWY1FzeWkwWGhVSUtTSitDdkhDXC9cL095c29kS3c9PSIsIm1hYyI6ImIyZjY0MGFkNzAwZTUyZmVlMzFjM2FmMjYyYWM1ZTc2MWYwOTJjYmI3MTQ1YWExMjI5MmU1MTczNTdhOTQ0ODUifQ%3D%3D',

'eyJpdiI6IjU3NkhKUHl2NUhKOThIMHFQM2NTVkE9PSIsInZhbHVlIjoiRmtWUkFDVUpBWTNxeVUzQmxVNW0za3VRVXI0ZnpoR3AzcXJPVU5VU0tTV1J4VHZrbHJZXC9VbUpUTG11S0ZzT0Qwd3NnS0pkRU4xWHZVbG9wZVhqS0RBPT0iLCJtYWMiOiI4YzBlMTY5NTUxZjA4MzlmZmRiODhiNDY4ODhkYjMyNzRhNmE4YzJlZmExZmEzZDFlY2QxZWNhZjg1YzgyOGJhIn0%3D']

# 把cookies转换为字典

requests.utils.dict\_from\_cookiejar(r.cookies)

{'XSRF-TOKEN': 'eyJpdiI6IkJFN3grcE1PbFwvN0Npa0k2U1lmV3ZnPT0iLCJ2YWx1ZSI6ImRIWU1wWjZJcDBrYzU4OURpZUFzUW9lUFFRYUFkQmJjNWV0RzdYUWtkdzQrc09zS20rclppUmhUQlBWY1FzeWkwWGhVSUtTSitDdkhDXC9cL095c29kS3c9PSIsIm1hYyI6ImIyZjY0MGFkNzAwZTUyZmVlMzFjM2FmMjYyYWM1ZTc2MWYwOTJjYmI3MTQ1YWExMjI5MmU1MTczNTdhOTQ0ODUifQ%3D%3D',

'laravel\_session': 'eyJpdiI6IjU3NkhKUHl2NUhKOThIMHFQM2NTVkE9PSIsInZhbHVlIjoiRmtWUkFDVUpBWTNxeVUzQmxVNW0za3VRVXI0ZnpoR3AzcXJPVU5VU0tTV1J4VHZrbHJZXC9VbUpUTG11S0ZzT0Qwd3NnS0pkRU4xWHZVbG9wZVhqS0RBPT0iLCJtYWMiOiI4YzBlMTY5NTUxZjA4MzlmZmRiODhiNDY4ODhkYjMyNzRhNmE4YzJlZmExZmEzZDFlY2QxZWNhZjg1YzgyOGJhIn0%3D'}

# 以元组形式打印

**for** item **in** r.cookies.items():

**print**(item)

('XSRF-TOKEN', 'eyJpdiI6IkJFN3grcE1PbFwvN0Npa0k2U1lmV3ZnPT0iLCJ2YWx1ZSI6ImRIWU1wWjZJcDBrYzU4OURpZUFzUW9lUFFRYUFkQmJjNWV0RzdYUWtkdzQrc09zS20rclppUmhUQlBWY1FzeWkwWGhVSUtTSitDdkhDXC9cL095c29kS3c9PSIsIm1hYyI6ImIyZjY0MGFkNzAwZTUyZmVlMzFjM2FmMjYyYWM1ZTc2MWYwOTJjYmI3MTQ1YWExMjI5MmU1MTczNTdhOTQ0ODUifQ%3D%3D')

('laravel\_session', 'eyJpdiI6IjU3NkhKUHl2NUhKOThIMHFQM2NTVkE9PSIsInZhbHVlIjoiRmtWUkFDVUpBWTNxeVUzQmxVNW0za3VRVXI0ZnpoR3AzcXJPVU5VU0tTV1J4VHZrbHJZXC9VbUpUTG11S0ZzT0Qwd3NnS0pkRU4xWHZVbG9wZVhqS0RBPT0iLCJtYWMiOiI4YzBlMTY5NTUxZjA4MzlmZmRiODhiNDY4ODhkYjMyNzRhNmE4YzJlZmExZmEzZDFlY2QxZWNhZjg1YzgyOGJhIn0%3D')

chars = [11,22,33,44]

[i **for** i **in** enumerate(chars)]

Out[75]: [(0, 11), (1, 22), (2, 33), (3, 44)]

**for** i,num **in** enumerate(chars):

**print**(i,num)

0 11

1 22

2 33

3 44

**for** i **in** enumerate(r.cookies):

**print**(i)

(0, Cookie(version=0, name='XSRF-TOKEN', value='eyJpdiI6Im5naFh5bUlqQm1UcFNKV3VzbG92QVE9PSIsInZhbHVlIjoiTG5wclpvTWNpdjMrR1hUelBiRU5XWEFKb2VvWGhrOXBjbVMzWHF3U3htVTc1ZnZsSFplSEN2eHlhZFwvVTk5N3V4TlBWYUVSVXI4TUd6ZmdodmVzWWZBPT0iLCJtYWMiOiJmMTBiNzAxNWE5NzNiMjdlYTU4MTk2YmJjMDU0YzBiMzRjM2JmMjQ0N2Q3NDNlZTQxNzM2YjU3NzFmODdlNjcxIn0%3D', port=None, port\_specified=False, domain='.hellobi.com', domain\_specified=True, domain\_initial\_dot=True, path='/', path\_specified=True, secure=False, expires=1528031098, discard=False, comment=None, comment\_url=None, rest={}, rfc2109=False))

(1, Cookie(version=0, name='laravel\_session', value='eyJpdiI6IjNDM0RqVmpPbGdNeldGeFphbWZ4REE9PSIsInZhbHVlIjoiT1M1XC9XUElKUnp4ZlVZWjltRTNDc0lTVERIWlQwQ0UyZEc5TlpqcGdxZk1UY04xT2ZqRWY1TWNlYjF2Ukh2d3FyRzBkeDVlNUhnSUlTOVE5V2pWd3hnPT0iLCJtYWMiOiIxOGU5YWNhMWJmMDJhZDZjZTcwNGEwMGQ3ODYzNGUxMWQyOGMyNjk1YjI1NDNmYTdjMjI0YTRlY2U5MjU0ODhkIn0%3D', port=None, port\_specified=False, domain='.hellobi.com', domain\_specified=True, domain\_initial\_dot=True, path='/', path\_specified=True, secure=False, expires=1528031098, discard=False, comment=None, comment\_url=None, rest={'HttpOnly': None}, rfc2109=False))

**for** i,j **in** enumerate(r.cookies):

**print**(i,j)

0 <Cookie XSRF-TOKEN=eyJpdiI6Im5naFh5bUlqQm1UcFNKV3VzbG92QVE9PSIsInZhbHVlIjoiTG5wclpvTWNpdjMrR1hUelBiRU5XWEFKb2VvWGhrOXBjbVMzWHF3U3htVTc1ZnZsSFplSEN2eHlhZFwvVTk5N3V4TlBWYUVSVXI4TUd6ZmdodmVzWWZBPT0iLCJtYWMiOiJmMTBiNzAxNWE5NzNiMjdlYTU4MTk2YmJjMDU0YzBiMzRjM2JmMjQ0N2Q3NDNlZTQxNzM2YjU3NzFmODdlNjcxIn0%3D **for** .hellobi.com/>

1 <Cookie laravel\_session=eyJpdiI6IjNDM0RqVmpPbGdNeldGeFphbWZ4REE9PSIsInZhbHVlIjoiT1M1XC9XUElKUnp4ZlVZWjltRTNDc0lTVERIWlQwQ0UyZEc5TlpqcGdxZk1UY04xT2ZqRWY1TWNlYjF2Ukh2d3FyRzBkeDVlNUhnSUlTOVE5V2pWd3hnPT0iLCJtYWMiOiIxOGU5YWNhMWJmMDJhZDZjZTcwNGEwMGQ3ODYzNGUxMWQyOGMyNjk1YjI1NDNmYTdjMjI0YTRlY2U5MjU0ODhkIn0%3D **for** .hellobi.com/>

**for** i,j **in** enumerate(r.cookies.items()):

**print**(i,j)

0 ('XSRF-TOKEN', 'eyJpdiI6Im5naFh5bUlqQm1UcFNKV3VzbG92QVE9PSIsInZhbHVlIjoiTG5wclpvTWNpdjMrR1hUelBiRU5XWEFKb2VvWGhrOXBjbVMzWHF3U3htVTc1ZnZsSFplSEN2eHlhZFwvVTk5N3V4TlBWYUVSVXI4TUd6ZmdodmVzWWZBPT0iLCJtYWMiOiJmMTBiNzAxNWE5NzNiMjdlYTU4MTk2YmJjMDU0YzBiMzRjM2JmMjQ0N2Q3NDNlZTQxNzM2YjU3NzFmODdlNjcxIn0%3D')

1 ('laravel\_session', 'eyJpdiI6IjNDM0RqVmpPbGdNeldGeFphbWZ4REE9PSIsInZhbHVlIjoiT1M1XC9XUElKUnp4ZlVZWjltRTNDc0lTVERIWlQwQ0UyZEc5TlpqcGdxZk1UY04xT2ZqRWY1TWNlYjF2Ukh2d3FyRzBkeDVlNUhnSUlTOVE5V2pWd3hnPT0iLCJtYWMiOiIxOGU5YWNhMWJmMDJhZDZjZTcwNGEwMGQ3ODYzNGUxMWQyOGMyNjk1YjI1NDNmYTdjMjI0YTRlY2U5MjU0ODhkIn0%3D')

In requests, the cookie jar **is** a very special object. You might notice that **if** you do:

r.cookies['XSRF-TOKEN']

You''ll receive the value of that cookie **as** a string (**in** your example f).

To get the actual cookie object that holds that information, you will have to iterate over the cookie jar like so:

**for** cookie **in** r.cookies:

**if** cookie.name == 'XSRF-TOKEN':

domain = cookie.domain

**print**(domain)

*# 查看cookie所有的方法*

**for** cookie **in** r.cookies:

**print**(dir(cookie))

[dir(cookie) **for** cookie **in** r.cookies][0]

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', '\_rest', 'comment', 'comment\_url', 'discard', 'domain', 'domain\_initial\_dot', 'domain\_specified', 'expires', 'get\_nonstandard\_attr', 'has\_nonstandard\_attr', 'is\_expired', 'name', 'path', 'path\_specified', 'port', 'port\_specified', 'rfc2109', 'secure', 'set\_nonstandard\_attr', 'value', 'version']

### 处理cookies、session请求

requests 提供了一个叫做session类，来实现客户端和服务端的会话保持

使用方法:

1. 实例化一个session对象
2. 让session发送get或者post请求

session = requests.session()

response = session.get(url, headers)

Session模拟登陆豆瓣

import requests

# 1. 创建session对象，可以保存Cookie值

ssion = requests.session()

url = "https://accounts.douban.com/login"

# 2. 处理 headers

headers={ "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko"}

# 3. 需要登录的用户名和密码, 在request中的form\_data中查看向服务器发送的表单信息

data = {"form\_email": "superwuzhiwei@163.com", "form\_password":"w6u9y5a3n8zi"}

# 4. 发送附带用户名和密码的请求，并获取登录后的Cookie值，保存在ssion里

ssion.post(url, data=data, headers=headers)

# 个人的订单页面, 如果没有登录, 会跳转到登录页面，必须要在登陆的状态下才能查看

order\_url = "https://www.douban.com/mine/orders/"

# 5. ssion包含用户登录后的Cookie值，可以直接访问那些登录后才可以访问的页面

response = ssion.get(order\_url)

# 6. 打印响应内容

# print(response.text)

with open("douban.html", "w", encoding="utf-8") as f:

f.write(response.text)

### 携带cookie请求

- 通过带和不带cookie对网站进行测试, 看不带能不能获取数据

- 携带一堆cookie进行请求，把cookie组成cookie池

### 使用requests提供的session类来请求登陆之后的网站的思路

- 实例化session

- 先使用session发送请求，登录对网站，把cookie保存在session中

- 再使用session请求登陆之后才能访问的网站，session能够自动的携带登录成功时保存在其中的cookie，进行请求

### 不发送post请求，使用cookie获取登录后的页面

- cookie过期时间很长的网站

- 在cookie过期之前能够拿到所有的数据，但需要手动更新cookie, 比较麻烦

- 配合其他程序一起使用，其他程序专门获取cookie，当前程序专门请求页面

### 获取登录后的页面的三种方式

- 实例化session，使用session发送post请求，在使用他获取登陆后的页面

- headers中添加cookie键，值为cookie字符串

- 在请求方法中添加cookies参数，接收字典形式的cookie。字典形式的cookie中的键是cookie的name对应的值，值是cookie的value对应的值

动手尝试使用session来登录人人网:

<http://www.renren.com/PLogin.do>

### 三种方式模拟登陆人人网

方法1, 使用session登录

# coding=utf-8

**import** **requests**

session = requests.session()

post\_url = "http://www.renren.com/PLogin.do"

post\_data = {"email":"mr\_mao\_hacker@163.com", "password":"alarmchime"}

headers = {

"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"

}

#使用session发送post请求，cookie保存在其中

session.post(post\_url, data=post\_data, headers=headers)

#在使用session进行请求登陆之后才能访问的地址

r = session.get("http://www.renren.com/327550029/profile", headers=headers)

#保存页面

**with** open("renren1.html", "w", encoding="utf-8") **as** f:

f.write(r.content.decode())

方法二: 使用保存的cookie值登陆, 把cookie作为header的一部分

# coding=utf-8

**import** **requests**

headers = {

"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36",

"Cookie":"anonymid=j3jxk555-nrn0wh; \_r01\_=1; \_ga=GA1.2.1274811859.1497951251; \_de=BF09EE3A28DED52E6B65F6A4705D973F1383380866D39FF5; ln\_uact=mr\_mao\_hacker@163.com; depovince=BJ; jebecookies=54f5d0fd-9299-4bb4-801c-eefa4fd3012b|||||; JSESSIONID=abcI6TfWH4N4t\_aWJnvdw; ick\_login=4be198ce-1f9c-4eab-971d-48abfda70a50; p=0cbee3304bce1ede82a56e901916d0949; first\_login\_flag=1; ln\_hurl=http://hdn.xnimg.cn/photos/hdn421/20171230/1635/main\_JQzq\_ae7b0000a8791986.jpg; t=79bdd322e760beae79c0b511b8c92a6b9; societyguester=79bdd322e760beae79c0b511b8c92a6b9; id=327550029; xnsid=2ac9a5d8; loginfrom=syshome; ch\_id=10016; wp\_fold=0"

}

r = requests.get("http://www.renren.com/327550029/profile", headers=headers)

#保存页面

**with** open("renren2.html", "w", encoding="utf-8") **as** f:

f.write(r.content.decode())

方法二: 使用保存的cookie值登陆, 把cookie作为requests.get的一个参数

# coding=utf-8

**import** **requests**

headers = {

"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36",

}

cookies="anonymid=j3jxk555-nrn0wh; \_r01\_=1; \_ga=GA1.2.1274811859.1497951251; \_de=BF09EE3A28DED52E6B65F6A4705D973F1383380866D39FF5; ln\_uact=mr\_mao\_hacker@163.com; depovince=BJ; jebecookies=54f5d0fd-9299-4bb4-801c-eefa4fd3012b|||||; JSESSIONID=abcI6TfWH4N4t\_aWJnvdw; ick\_login=4be198ce-1f9c-4eab-971d-48abfda70a50; p=0cbee3304bce1ede82a56e901916d0949; first\_login\_flag=1; ln\_hurl=http://hdn.xnimg.cn/photos/hdn421/20171230/1635/main\_JQzq\_ae7b0000a8791986.jpg; t=79bdd322e760beae79c0b511b8c92a6b9; societyguester=79bdd322e760beae79c0b511b8c92a6b9; id=327550029; xnsid=2ac9a5d8; loginfrom=syshome; ch\_id=10016; wp\_fold=0"

cookies = {i.split("=")[0]:i.split("=")[1] **for** i **in** cookies.split("; ")}

**print**(cookies)

r = requests.get("http://www.renren.com/327550029/profile", headers=headers, cookies=cookies)

#保存页面

**with** open("renren3.html", "w", encoding="utf-8") **as** f:

f.write(r.content.decode())

### 人人网的登录逻辑解析

**人人网的登陆地址从哪里来**

打开人人网主页, 检查元素

<form method="post" id="loginForm" class="login-form" action="http://www.renren.com/PLogin.do">

...

</from>

form表单中action的地址就是模拟登陆人人网的地址

**而post的键值对的键从哪里来**

input标签的name属性值.

在表单进行提交时, 把name中的值作为键进行提交.

email: superwuzhiwei@163.com

icode:

origURL: http://www.renren.com/home

domain: renren.com

key\_id: 1

captcha\_type: web\_login

password: 750f315dc62c79009d483a691d7703145d0fe48bf044e4352f8cc9275064b35f

rkey: 3102325bf2950a0c02b66649b7a956a3

### 寻找登录的post地址

- 在form表单中寻找action对应的url地址

- post的数据是input标签中name的值作为键，真正的用户名密码作为值的字典，post的url地址就是action对应的url地址

- 抓包，寻找登录的url地址

- 在chrome的Network中勾选perserve log按钮，防止页面跳转找不到url

- 寻找post数据，确定参数

- 参数不会变，直接用，比如密码不是动态加密的时候

- 参数会变, 多次输入错误的密码找规律, 看post data的内容会不会变化

- 参数在当前的响应中

- 通过js生成

在登陆页面的请求的response中找不到rkey的参数, 所以这个参数应该是在其它请求中, 点击检查元素中右边的菜单, 选择search all files: rkey:, 找到生成rkey值的js代码

function s() {

var e, i = $("errorMessage"), s = $("codeTip"), r = $("code"), a = $("codeimg"), l = ($("forgetPwd"),

$("getpassword"),

$("account\_stop"),

$("account\_lock"),

$("password")), d = $("email"), g = $("sendemail"), f = $("yellow\_error"), v = $("gotoEmail");

e = "" != u ? {

password: c,

rkey: u

} : {},

n.getKeys = function() {

var e = new \_.Request({

url: "http://login.renren.com/ajax/getEncryptKey",

onSuccess: function(e) {

var n = JSON.parse(e.responseText);

1 == n.isEncrypt && (l(2 \* n.maxdigits),

T = new t(n.e, "null", n.n),

XN.isFunction(i) && i.call(this, T, n.rkey))

},

onError: function() {

XN.DO.showError("网络通信失败，请稍后再试！")

}

});

n是responseText中的内容, rkey存在于变量n中, 也就是说rkey是请求之后得到的一个值,

刷新人人登陆页面, 就会看到getEncryptKey的地址,

在etEncryptKey的请求中, 查看preview中的内容, 就可以看到rkey的内容

...

{isEncrypt: true, e: "10001", n: "dfe070a55ca4fe581f4e1c60a41cef1279730507325a33d9b932a222f50fc493", …}

e:"10001"

isEncrypt:true

maxdigits:"19"

n:"dfe070a55ca4fe581f4e1c60a41cef1279730507325a33d9b932a222f50fc493"

rkey:"88f1e28182875cfc25345825522c87a9"

...

此时使用刷新后的页面进行登陆, 发送的form data中的rkey值就是上面的值. 也就是每次刷新登陆页面后都会生成新的rkey值.

由于login发送login请求时的时间戳和密码都发生了变化, 所以比较麻烦, 可以尝试手机端的登陆

使用手机登陆人人网, 每次点击登录按钮都会出现rkey和clog两个请求, 也就是使用手机端点击登录后会发送2次请求, 1次发送get请求获取rkey, 1次发送post请求进行登陆. 这与百度翻译的请求类似, 先发送一个请求确认源语言和目标语言, 再发送请求进行翻译.

但是发送的password是变化的, 是通过js改变的, 如何来获取对应的js代码呢.

右击登陆按钮, 选择检查元素, 选择Event Listeners, 选中Framework listeners, 点击click中的代码, 就可以跳转到点击事件的js中.

可以把js中的代码复制到chrome的console中执行, 查看返回的信息. 如:

$(".phonenum").value,

$(".password").value,

可以在js代码中添加断点, 对js代码的功能进行解析. 再次刷新页面, 就会暂停在断点的位置, 放在某个函数中, 就会显示函数的位置, 点击进入该函数. 也可以点右边的工具栏, step into next function call

对于没有事件触发的js代码, 如手机端的rkey请求, 就需要使用search all file中进行搜索.

### 定位想要的js

- 选择会触发js时间的按钮，点击event listener，找到js的位置

- 通过chrome中的search all file来搜索url中关键字

- 添加断点的方式来查看js的操作，通过python来进行同样的操作

In [47]: a = 3 if 3>2 else 4

In [48]: a

Out[48]: 3

In [49]: a = 3 if 3>5 else 4

In [50]: a

Out[50]: 4

In [51]:

### Requests小技巧

1、requests.utils.dict\_from\_cookiejar 把cookie对象转化为字典

1.1. requests.get(url, cookies={})

2、**请求 SSL证书验证**

response = requests.get("https://www.12306.cn/mormhweb/ ", verify=False)

3、设置超时

response = requests.get(url, 1)

4、配合状态码判断是否请求成功

assert response.status\_code == 200

In [59]: import requests

In [60]: response = requests.get("http://www.baidu.com/")

In [61]: response.cookies

Out[61]: <RequestsCookieJar[Cookie(version=0, name='BDORZ', value='27315', port=None, port\_specified=False, domain='.baidu.com', domain\_specified=True, domain\_initial\_dot=True, path='/', path\_specified=True, secure=False, expires=1523240635, discard=False, comment=None, comment\_url=None, rest={}, rfc2109=False)]>

In [62]: requests.utils.dict\_from\_cookiejar(response.cookies)

Out[62]: {'BDORZ': '27315'}

In [63]: requests.utils.cookiejar\_from\_dict({'BDORZ': '27315'})

Out[63]: <RequestsCookieJar[Cookie(version=0, name='BDORZ', value='27315', port=None, port\_specified=False, domain='', domain\_specified=False, domain\_initial\_dot=False, path='/', path\_specified=True, secure=False, expires=None, discard=True, comment=None, comment\_url=None, rest={'HttpOnly': None}, rfc2109=False)]>

In [64]:

requests.utils.unquote("https://www.baidu.com/s?wd=%E4%BC%A0%E6%99%BA%E6%92%AD%E5%AE%A2")

Out[64]: 'https://www.baidu.com/s?wd=传智播客'

In [65]: requests.utils.quote("https://www.baidu.com/s?wd=传智播客")

Out[65]: 'https%3A//www.baidu.com/s%3Fwd%3D%E4%BC%A0%E6%99%BA%E6%92%AD%E5%AE%A2'

下面我们通过一个例子整体来看一下以上4点的用法

# coding=utf-8

# 注意本文件不能以数字开头, 才能被其它文件引用

import requests

headers={"User-Agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

def \_parse\_url(url):

response = requests.get(url, headers=headers, timeout=3)

assert response.status\_code == 200

return response.content.decode()

def parse\_url(url):

# 重新定义一个函数, 把try和except放在这个函数中

try:

html\_str = \_parse\_url(url, method, data, proxies)

except:

html\_str = None

return html\_str

if \_\_name\_\_ == '\_\_main\_\_':

url = "www.baidu.com"

print(parse\_url(url))

有时候遇到网络问题了, 就需要再次发送一下requests请求.

### 安装第三方模块

- pip install retrying

- 下载源码解码，进入解压后的目录，```python setup.py install```

- `\*\*\*.whl` 安装方法 `pip install \*\*\*.whl`

使用retrying模块

pip install retrying

# coding=utf-8

# 注意本文件不能以数字开头, 才能被其它文件引用

import requests

from retrying import retry

headers={"User-Agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

@retry(stop\_max\_attempt\_number=3)

def \_parse\_url(url):

# 测试进行了多少次尝试

print("\*"\*20)

response = requests.get(url, headers=headers, timeout=3)

assert response.status\_code == 200

return response.content.decode()

def parse\_url(url):

# 重新定义一个函数, 把try和except放在这个函数中

try:

html\_str = \_parse\_url(url, method, data, proxies)

except:

html\_str = None

return html\_str

if \_\_name\_\_ == '\_\_main\_\_':

url = "www.baidu.com"

print(parse\_url(url))

可以发送POST请求和使用代理地址

# coding=utf-8

# 注意本文件不能以数字开头, 才能被其它文件引用

import requests

from retrying import retry

headers={"User-Agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

# 尝试3次发送请求, 如果3次请求还没有成功, 整个程序才会报错. 再在下面定义一个try来捕获异常

@retry(stop\_max\_attempt\_number=3)

def \_parse\_url(url, method, data, proxies):

print("\*"\*20)

if method=="POST":

response = requests.post(url, data=data, headers=headers, proxies=proxies)

else:

response = requests.get(url, headers=headers, timeout=3, proxies=proxies)

assert response.status\_code == 200

return response.content.decode()

# 默认的请求方式是GET, 如果用户传过来了post, 就可以发送post请求.

def parse\_url(url, method="GET", data=None, proxies={}):

# 重新定义一个函数, 把try和except放在这个函数中

try:

html\_str = \_parse\_url(url, method, data, proxies)

except:

html\_str = None

return html\_str

if \_\_name\_\_ == '\_\_main\_\_':

url = "www.baidu.com"

print(parse\_url(url))

定义一个专门解析url请求的文件, 以后都可以再调用这个文件

parse\_url.py

# coding=utf-8

# 注意本文件不能以数字开头, 才能被其它文件引用

**import** **requests**

**from** **retrying** **import** retry

headers={"User-Agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

# 尝试3次发送请求, 如果3次请求还没有成功, 整个程序才会报错. 再在下面定义一个try来捕获异常

@retry(stop\_max\_attempt\_number=3)

**def** \_parse\_url(url, method, data, proxies):

**print**("\*"\*20)

**if** method=="POST":

response = requests.post(url, data=data, headers=headers, proxies=proxies)

**else**:

response = requests.get(url, headers=headers, timeout=3, proxies=proxies)

**assert** response.status\_code == 200

**return** response.content.decode()

# 默认的请求方式是GET, 如果用户传过来了post, 就可以发送post请求.

**def** parse\_url(url, method="GET", data=None, proxies={}):

# 重新定义一个函数, 把try和except放在这个函数中

**try**:

html\_str = \_parse\_url(url, method, data, proxies)

**except**:

html\_str = None

**return** html\_str

**if** \_\_name\_\_ == '\_\_main\_\_':

url = "www.baidu.com"

**print**(parse\_url(url))

### 综合练习: 使用json爬取豆瓣电视剧

douban\_tv\_spider\_cate.py

# coding=utf-8

**import** **requests**

**import** **json**

**class** **DoubanSpider**:

**def** \_\_init\_\_(self):

self.url\_temp\_list = [

{

"url\_temp": "https://m.douban.com/rexxar/api/v2/subject\_collection/filter\_tv\_american\_hot/items?start={}&count=18&loc\_id=108288",

# 也可以使用正则匹配来获取国家的信息

"country": "US"

},

{

"url\_temp": "https://m.douban.com/rexxar/api/v2/subject\_collection/filter\_tv\_english\_hot/items?start={}&count=18&loc\_id=108288",

"country": "UK"

},

{

"url\_temp": "https://m.douban.com/rexxar/api/v2/subject\_collection/filter\_tv\_domestic\_hot/items?start={}&count=18&loc\_id=108288",

"country": "CN"

}

]

self.headers = {

"Referer": "https://m.douban.com/tv/american",

"User-Agent": "Mozilla/5.0 (Linux; Android 5.1.1; Nexus 6 Build/LYZ28E) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Mobile Safari/537.36"}

**def** parse\_url(self, url): # 发送请求，获取响应

**print**(url)

response = requests.get(url, headers=self.headers)

**return** response.content.decode()

**def** get\_content\_list(self, json\_str): # 提取是数据

dict\_ret = json.loads(json\_str)

content\_list = dict\_ret["subject\_collection\_items"]

total = dict\_ret["total"]

**return** content\_list, total

**def** save\_content\_list(self, content\_list, country): # 保存

# 把文件循环写入放在打开文件中, 打开一次文件就能进行多次的写入. 与mysql的操作过程类似

**with** open("douban.txt", "a", encoding="utf-8") **as** f:

**for** content **in** content\_list:

content["country"] = country

f.write(json.dumps(content, ensure\_ascii=False))

f.write("**\n**") # 写入换行符，进行换行, 每一行是一条新闻的数据

**print**("保存成功")

**def** run(self): # 实现主要逻辑

**for** url\_temp **in** self.url\_temp\_list:

num = 0

total = 100 # 假设有第一页. 给出一个total值, 由于0<100, 所以此时表明有第一页, 至少会执行一次循环

#如果最后一页刚好有15条数据, 在倒数第二页取完数据时, 倒数第二页的total+18就大于total的值了, 就不会再进行爬取了, 所以这里要设置num<total+18, 这里也不能使用等号. 如果是18的倍数, 使用等号时在最后一页爬取完成后会再次爬取一个空页. 使用下面的判断方式更好一些, 因为页面的内容数可能并没有total中的那么多.

**while** num < total + 18:

# 1.start\_url

url = url\_temp["url\_temp"].format(num)

# 2.发送请求，获取响应

json\_str = self.parse\_url(url)

# 3.提取是数据

content\_list, total = self.get\_content\_list(json\_str)

# 4.保存

self.save\_content\_list(content\_list, url\_temp["country"])

# if len(content\_list)<18:

# break

# 5.构造下一页的url地址, 进入循环. 因为不确定一共有多少数据, 就没办法构造很确定的终止页

num += 18

**if** \_\_name\_\_ == '\_\_main\_\_':

douban\_spider = DoubanSpider()

douban\_spider.run()

# 第二部分 数据提取方法

## 第二部分课程概要

1. **基础知识**
2. **Json知识点复习**
3. **正则表达式的复习**
4. **xpath和lxml**

## 数据提取

什么是数据提取？

简单的来说，数据提取就是从响应中获取我们想要的数据的过程

## 数据分类

**非结构化的数据: html等**

**处理方法: 正则表达式、xpath**

**结构化数据: json，xml等**

**处理方法: 转化为python数据类型**

json

访问[www.toutiao.com](http://www.toutiao.com)

在network中查看focus响应, preview

## 数据提取之JSON

由于把json数据转化为python内建数据类型很简单，所以爬虫中，如果我们能够找到返回json数据的URL，就会尽量使用这种URL

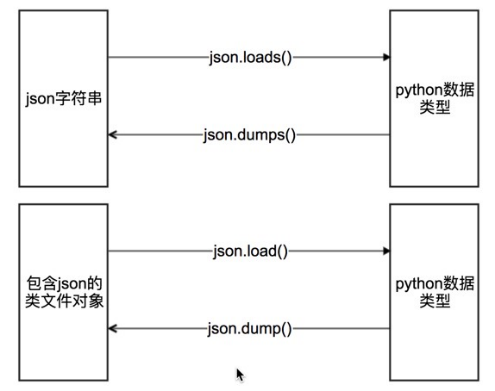
JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，它使得人们很容易的进行阅读和编写。同时也方便了机器进行解析和生成。适用于进行数据交互的场景，比如网站前台与后台之间的数据交互。

那么问题来了: 哪里能找到返回json的url呢？

1、使用chrome切换到手机页面

2、抓包手机app的软件

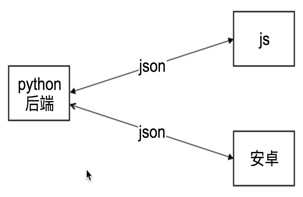
## 数据提取之JSON



具有read()或者write()方法的对象就是类文件对象

f = open("a.txt", "r") f就是类文件对象

## 数据提取之JSON





Json在数据交换中起到了一个载体的作用，承载着相互传递的数据

### json使用注意点

- json中的字符串都是双引号引起来的

- 如果不是双引号

- eval：能实现简单的字符串和python类型的转化, 但嵌套太复杂时就不能转换了.

- replace：把单引号替换为双引号, 再使用json.dumps进行处理

- 往一个文件中写入多个json串，不再是一个json串，不能直接读取

- 一行写一个json串，按照行来读取

### 豆瓣电影数据提取分析

访问手机端豆瓣电影, https://m.douban.com/movie/

看电影信息有没有显示在响应的Response中, 如果没有, 可以在preview中搜索., 因为中文可能会进行编码. 如果preview中也没有搜索到页面上显示的电影, 不说明当前的响应中没有我们想要的数据. 因为电影的信息非常多, 不可能全部是使用js来生成的. 排除掉js, css, 图片, 再查找. 可以使用多选模式过滤掉不需要的内容, 但不一定准确. 可以使用全局搜索功能. 中文可能被编码, 所以搜索页面上的数字或英文效果更好.

https://m.douban.com/rexxar/api/v2/subject\_collection/movie\_showing/items?os=ios&callback=jsonp1&start=0&count=8&loc\_id=108288&\_=1523159626778

在network中找到subject\_collection

Request URL: https://m.douban.com/rexxar/api/v2/subject\_collection/movie\_showing/items?os=android&callback=jsonp1&start=0&count=8&loc\_id=108288&\_=1523160047154

访问该, 网页, 得到的不是标准的json格式, 要把其中的callback=jsonp1去掉, 返回的就是json数据了.

访问此页面

### 使用json爬取豆瓣电影信息

豆瓣抓取手机页面时发送的headers中必须要有来自手机页面的"Referer": "https://m.douban.com/tv/american"的这个参数. 只有来自m.douban的请求才会被解析, 否则就会出错.

修改parse\_url.py, 给headers中添加referer信息, 并使用移动端的User-Agent

parse\_url.py

# coding=utf-8

# 注意本文件不能以数字开头, 才能被其它文件引用

**import** **requests**

**from** **retrying** **import** retry

# headers={"User-Agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

headers={

"User-Agent":"Mozilla/5.0 (Linux; Android 5.0; SM-G900P Build/LRX21T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Mobile Safari/537.36",

"Referer": "https://m.douban.com/tv/",

}

# 尝试3次发送请求, 如果3次请求还没有成功, 整个程序才会报错. 再在下面定义一个try来捕获异常

@retry(stop\_max\_attempt\_number=3)

**def** \_parse\_url(url, method, data, proxies):

**print**("\*"\*20)

**if** method=="POST":

response = requests.post(url, data=data, headers=headers, proxies=proxies)

**else**:

response = requests.get(url, headers=headers, timeout=3, proxies=proxies)

**assert** response.status\_code == 200

**return** response.content.decode()

# 默认的请求方式是GET, 如果用户传过来了post, 就可以发送post请求.

**def** parse\_url(url, method="GET", data=None, proxies={}):

# 重新定义一个函数, 把try和except放在这个函数中

**try**:

html\_str = \_parse\_url(url, method, data, proxies)

**except**:

html\_str = None

**return** html\_str

**if** \_\_name\_\_ == '\_\_main\_\_':

url = "www.baidu.com"

**print**(parse\_url(url))

douban\_mobile\_movie\_spider.py

# coding=utf-8

**import** **json**

**import** **requests**

**from** **parse\_url** **import** parse\_url

# 美化的输出

**from** **pprint** **import** pprint

# 豆瓣抓取手机页面时发送的headers中必须要有来自手机页面的"Referer": "https://m.douban.com/tv/american"的这个参数. 只有来自m.douban的请求才会被解析, 否则就会出错.

url = "https://m.douban.com/rexxar/api/v2/subject\_collection/movie\_showing/items?start=0&count=18&loc\_id=108288"

html\_str = parse\_url(url)

# json.loads把json字符串转化为python类型

ret1 = json.loads(html\_str)

# pprint(ret1)

# print(type(ret1))

# json.dumps能够把python类型转化为json字符串, 在把数据写入文本时使用, 写的时候必须是文本, 不能是字典

**with** open("douban.json", "w", encoding="utf-8") **as** f:

# ensure\_ascii=False, 如果是中文, 就不转换为ascii, 而是显示为中文. indent在上一级的基础上都缩进4个空格

f.write(json.dumps(ret1, ensure\_ascii=False, indent=4))

# 也可以直接转换为字符串写入, 但没办法进行格式化的操作了.

# f.write(str(ret1))

# 读取保存的内容

# with open("douban.json", "r", encoding="utf-8") as f:

# ret2 = f.read()

# ret3 = json.loads(ret2)

# print(ret3)

# print(type(ret3))

# 使用json.load提取类文件对象中的数据

**with** open("douban.json", "r", encoding="utf-8") **as** f:

ret4 = json.load(f)

**print**(ret4)

**print**(type(ret4))

#json.dump能够把python类型放入类文件对象中

**with** open("douban1.json", "w", encoding="utf-8") **as** f:

json.dump(ret1, f, ensure\_ascii=False, indent=2)

### 使用类来实现百度翻译

# coding=utf-8

**import** **requests**

**import** **json**

**import** **sys**

**class** **BaiduFanyi**:

**def** \_\_init\_\_(self, trans\_str):

self.trans\_str = trans\_str

self.lang\_detect\_url = "http://fanyi.baidu.com/langdetect"

self.trans\_url = "http://fanyi.baidu.com/basetrans"

self.headers = {"User-Agent":"Mozilla/5.0 (Linux; Android 5.1.1; Nexus 6 Build/LYZ28E) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Mobile Safari/537.36"}

# 获取语言类型和获取翻译结果都是发送post请求, 所以可以写为一个方法

**def** parse\_url(self, url, data): #发送post请求，获取响应

response = requests.post(url, data=data, headers=self.headers)

# 获取语言类型和获取翻译结果都是返回json, 可以直接使用json.loads转换为字典格式

**return** json.loads(response.content.decode())

**def** get\_ret(self, dict\_response):#提取翻译的结果

ret = dict\_response["trans"][0]["dst"]

**print**("result is :", ret)

**def** run(self):#实现主要逻辑

#1.获取语言类型

#1.1 准备post的url地址，post\_data

lang\_detect\_data = {"query":self.trans\_str}

#1.2 发送post请求，获取响应, 如果实现起来比较复杂, 就可以把功能定义到方法中

lang = self.parse\_url(self.lang\_detect\_url, lang\_detect\_data)["lan"]

#1.3 提取语言类型

#2.准备post的数据, 如果语句不复杂时可以不用定义方法来实现

trans\_data = {"query":self.trans\_str, "from":"zh", "to":"en"} **if** lang== "zh" **else** \

{"query":self.trans\_str, "from":"en", "to":"zh"}

#3.发送请求，获取响应

dict\_response = self.parse\_url(self.trans\_url, trans\_data)

#4.提取翻译的结果

self.get\_ret(dict\_response)

**if** \_\_name\_\_ == '\_\_main\_\_':

trans\_str = sys.argv[1]

baidu\_fanyi = BaiduFanyi(trans\_str)

baidu\_fanyi.run()

### 实现百度翻译的语言自动识别

baidu\_translate\_automatic.py

import requests, sys, json

class Automatic():

def \_\_init\_\_(self, translade\_word):

self.translade\_word = translade\_word

self.langdetect\_headers ={"User-Agent": "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Mobile Safari/537.36"}

self.langdetect\_parpams = {"query": translade\_word}

self.langdetect\_url ="http://fanyi.baidu.com/langdetect"

self.translated\_url = "http://fanyi.baidu.com/basetrans"

def langdetect(self):

response = requests.post(self.langdetect\_url, headers = self.langdetect\_headers, data=self.langdetect\_parpams)

return json.loads(response.content.decode())['lan']

def get\_data\_language(self, language\_word):

*#*

*# if "zh" ==language\_word:*

*# translade\_data ={"query":self.translade\_word,*

*# "from":"zh",*

*# "to":"en"}*

*# else:*

*# translade\_data = {"query": self.translade\_word,*

*# "from": language\_word,*

*# "to": "zh"}*

return {"query":self.translade\_word, "from":"zh", "to":"en"} if "zh" ==language\_word \

else {"query": self.translade\_word, "from": language\_word, "to": "zh"}

def translade(self, translade\_data):

response = requests.post(self.translated\_url, data=translade\_data, headers = self.langdetect\_headers)

response\_data = json.loads(response.text)

*# print("1111111111", response\_data)*

return response\_data

def get\_ret(self, response\_data):

data = response\_data["trans"][0]["dst"]

print("{} 翻译后的结果：{}".format(self.translade\_word, data))

def run(self):

language\_word = self.langdetect()

translade\_data= self.get\_data\_language(language\_word)

response\_data = self.translade(translade\_data)

self.get\_ret(response\_data)

if \_\_name\_\_ == '\_\_main\_\_':

translade\_word = sys.argv[1]

automatic = Automatic(translade\_word)

automatic.run()

## 正则表达式复习

<https://docs.python.org/3.5/library/re.html>

<http://usyiyi.cn/translate/python_352/library/re.html>

有一点需要特别注意的是，正则表达式使用对特殊字符进行转义，所以如果我们要使用原始字符串，只需加一个r前缀，示例：为了使字符不受转义字符的影响，在字符前面加一个r, 所以正则表达式一般都需要在前面加一个r

r'chuanzhiboke\t\.\tpython'

str="nihao\tpython\naab"

print(str)

nihao python

aab

str2=r"nihao\tpython\naab"

print(str2)

加一个u表示是unicode字符串。

### 正则表达式的定义

**就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个"规则字符串"，这个"规则字符串"用来表达对字符串的一种过滤逻辑。**

### 常用正则表达式的方法

**pattern =** re.compile(编译)

pattern.match(从头找一个) 从第1个字符找, 一次匹配

pattern.search(找一个) 从任何一个位置找, 一次匹配

pattern.findall(找所有) , 返回列表

pattern.finditer 全部匹配，返回迭代器

pattern.split 分割字符串，返回列表

pattern.sub(替换)

### 正则表达式复习

In [1]: import re

In [2]: re.findall(".", "\n")

Out[2]: []

In [3]: re.findall(".", "\n", re.DOTALL)

Out[3]: ['\n']

In [4]: re.findall(".", "\n", re.S)

Out[4]: ['\n']

In [5]: re.findall("\.", ".")

Out[5]: ['.']

In [6]: re.findall("a[bcd]e", "abce")

Out[6]: []

In [7]: re.findall("a[bcd]+e", "abce")

Out[7]: ['abce']

In [8]: re.findall("abce|aede|afce", "abce")

Out[8]: ['abce']

| 竖线在爬虫中用的比较多, 把爬取到的内容中的所有不规则的字符都替换为空, 就可以把多个匹配规则写在|中.

In [9]: a = """

...: <div>

...: <div data-reactroot="" style="display: none;">想来知乎工作？请发送邮件到 jobs@zhihu.com

...: </div>

...: </div>

...: """

这里为什么可以匹配到3个元素, 因为复制的字符串中有换行符. 使用a来查看.

In [10]: re.findall("<.+>", a)

Out[10]: ['<div>', '<div data-reactroot="" style="display: none;">', '</div>', '</div>']

In [11]: a

Out[11]: '\n<div>\n <div data-reactroot="" style="display: none;">想来知乎工作？请发送邮件到 jobs@zhihu.com

\n </div>\n</div>\n'

使用re.S来使得点号.可以匹配到换行符, 就只匹配到1个元素

In [12]: re.findall("<.+>", a, re.S)

Out[12]: ['<div>\n <div data-reactroot="" style="display: none;">想来知乎工作？请发送邮件到 jobs@zhihu.com\

n </div>\n</div>']

打开非贪婪匹配模式, 可以匹配到3个元素

In [13]: re.findall("<.+?>", a, re.S)

Out[13]: ['<div>', '<div data-reactroot="" style="display: none;">', '</div>', '</div>']

In [14]: b = "chuanzhi2"

In [15]: re.sub("\d", "\_", b)

Out[15]: 'chuanzhi\_'

In [16]: re.sub("\d", "", b)

Out[16]: 'chuanzhi'

使用re.sub("\d+", "", b) 进行匹配和替换时, 每次执行这条语句时python都会把\d+所能匹配的结果全部读取到内存中, 然后逐个与b进行匹配, 如果页面内容非常长, 并且我们需要匹配的内容非常多, 反复进行匹配就会占用较多的内存, 并耗费较长的时间, 可以使用compile方法把\d+所能匹配到的结果预先全部读取到内存中去, 使用编译后的结果进行匹配和替换, 在反复执行时就可以节省内存和时间.

In [14]: b = "chuanzhi2"

In [15]: re.sub("\d", "\_", b)

Out[15]: 'chuanzhi\_'

In [16]: re.sub("\d", "", b)

Out[16]: 'chuanzhi'

In [17]: p = re.compile("\d")

In [18]: p.findall(b)

Out[18]: ['2']

In [19]: p.sub("\_", b)

Out[19]: 'chuanzhi\_'

In [20]: q = re.compile(".")

In [21]: q.findall("\n")

Out[21]: []

In [22]: q.findall("\n", re.S)

Out[22]: []

要把匹配模式和匹配的参数都放在re.compile中.

In [24]: q = re.compile(".", re.S)

In [25]: q.findall("\n")

Out[25]: ['\n']

**python中原始字符串r的用法**

让字符串表示它字面的意思.

在python正则表达中尽可能的使用原始字符串，待匹配的字符串中看到什么就在正则表达式写什么，就不会出现问题

In [26]: a = "a\nb"

In [27]: a[1]

Out[27]: '\n'

In [28]: len(a)

Out[28]: 3

In [29]: b = r"a\nb"

In [30]: len(b)

Out[30]: 4

In [31]: b[1]

Out[31]: '\\'

In [32]: b[2]

Out[32]: 'n'

In [33]: r"a\nb" == "a\nb"

Out[33]: False

In [34]: re.findall("a\nb", "a\nb")

Out[34]: ['a\nb']

使用原始字符串r进行匹配, 就能忽略目标字符串中转义字符的影响.

In [35]: r"a\nb" == "a\\nb"

Out[35]: True

In [36]: re.findall(r"a\nb", "a\nb")

Out[36]: ['a\nb']

In [37]: re.findall(r"a\nb", "a\\nb")

Out[37]: []

In [38]: re.findall(r"a\nb", "a\nb")

Out[38]: ['a\nb']

In [39]: re.findall(r"a\\nb", "a\\nb")

Out[39]: ['a\\nb']

In [40]: re.findall(r"a\\\nb", "a\\\nb")

Out[40]: ['a\\\nb']

### python中原始字符串r的用法

原始字符串定义(raw string): 所有的字符串都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符，原始字符串往往针对特殊字符而言。

'\n'长度为1，r'\n'长度为2

windows下不使用原始字符串会出现的问题:

>>> with open("D:\David\Desktop\IT学习文件夹\00 爬虫专题\00\_spider\_projects\mobile\_user\_agent.json", "r") as f:

... f.read()

...

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: embedded null character

>>> with open(r"D:\David\Desktop\IT学习文件夹\00 爬虫专题\00\_spider\_projects\mobile\_user\_agent.json", "r") as f:

... f.read()

...

'{"mobile":{"android":["Mozilla/5.0 (Linux; Android 7.0;…)]}'

正则表达式复习

问题来了:

1. re.compile该如何使用？
2. 如何非贪婪的去匹配内容？
3. re.findall(r"a.\*bc", "a\nbc", re.DOTALL) 和re.findall(r"a(.\*)bc", "a\nbc", re.DOTALL)的区别？

不分组时匹配的是全部，分组后匹配的是组内的内容

In [42]: re.findall(r"a.\*bc", "a\nbc", re.DOTALL)

Out[42]: ['a\nbc']

In [43]: re.findall(r"a(.\*)bc", "a\nbc", re.DOTALL)

Out[43]: ['\n']

在获取页面的内容时, 只有很少的一部分不规则的内容需要使用正则进行匹配,

### 正则表达式爬虫练习

1. 通过正则表达式匹配内涵段子的段子

<http://neihanshequ.com/>

### 内涵社区段子爬虫

# coding=utf-8

**import** **requests**

**import** **re**

**import** **json**

**class** **Neihan**:

**def** \_\_init\_\_(self):

self.start\_url = "http://neihanshequ.com/"

self.next\_url\_temp = "http://neihanshequ.com/joke/?is\_json=1&app\_name=neihanshequ\_web&max\_time={}"

self.headers = {"User-Agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

**def** parse\_url(self, url):#发送请求

**print**(url)

response = requests.get(url, headers=self.headers)

**return** response.content.decode()

**def** get\_first\_page\_content\_list(self, html\_str): #提取第一页的数据

# 只写p标签的匹配规则时会匹配到一些无用的内容, 所以要把匹配规则写的更加具体一些

# content\_list = re.findall(r"<p>(.\*?)</p>", html\_str, re.S)

content\_list = re.findall(r"<h1 class=**\"**title**\"**>.\*?<p>(.\*?)</p>", html\_str, re.S)

# 分析页面, 点击加载更多时, 会发送ajax请求, 发送请求的地址变化的部分就是max\_time的值, 上一次响应中的max\_time作为下一次请求的地址中的特征值. 需要提取出max\_time的值, 构造下一页的url地址, 第1页中的max\_time值的获取方式与第2页开始都不同

max\_time = re.findall("max\_time: '(.\*?)', ", html\_str)[0]

**return** content\_list, max\_time

**def** save\_content\_list(self, content\_list): #保存

**with** open("neihan.txt", "a", encoding="utf-8") **as** f:

**for** content **in** content\_list:

f.write(json.dumps(content, ensure\_ascii=False))

f.write("**\n\n**")

**print**("保存成功")

# 提取从第二页开始的json中的数据, 与第一页中提取数据的方法不同, 第1页是从html中提取, 从第2页开始都是从json中提取

**def** get\_content\_list(self, json\_str):

dict\_ret = json.loads(json\_str)

data = dict\_ret["data"]["data"]

content\_list = [i["group"]["content"] **for** i **in** data]

max\_time = dict\_ret["data"]["max\_time"]

has\_more = dict\_ret["data"]["has\_more"]

**return** content\_list, max\_time, has\_more

**def** run(self):#实现主要逻辑

#1.start\_url, 在\_\_init\_\_中定义

#2.发送请求，获取响应

html\_str = self.parse\_url(self.start\_url)

#3.提取数据

content\_list, max\_time = self.get\_first\_page\_content\_list(html\_str)

#4.保存

self.save\_content\_list(content\_list)

# 在ajax返回的数据中如果has\_more = True, 就表示有下一页

has\_more = True #有第二页

# 因为第一页的max\_time的值的传递方式与从第二页开始时都不同, 所以不能把第1页一起放到循环中.

**while** has\_more:

#5.构造下一页的url地址

next\_url = self.next\_url\_temp.format(max\_time)

#6.发送请求，获取响应

json\_str = self.parse\_url(next\_url)

#7.提取数据，提取max\_time

content\_list, max\_time, has\_more = self.get\_content\_list(json\_str)

#8.保存

self.save\_content\_list(content\_list)

#9.循环5-8步

**if** \_\_name\_\_ == '\_\_main\_\_':

neihan = Neihan()

neihan.run()

1. 获取http://36kr.com/网站首页的所有新闻

### 获取http://36kr.com/网站首页的所有新闻

# coding=utf-8

**import** **re**

**from** **parse\_url** **import** parse\_url

**import** **json**

url = "http://36kr.com/"

html\_str = parse\_url(url)

# 会出现错误. json.decoder.JSONDecodeError: Extra data: lin 1 column 136996

ret = re.findall("<script>var props=(.\*?)</script>", html\_str)[0]

ret = re.findall("<script>var props=(.\*?), locationnal=", html\_str)[0]

**with** open("36kr.json", "w", encoding="utf-8") **as** f:

f.write(ret)

ret = json.loads(ret)

**print**(ret)

## XPATH和LXML类库

为什么要学习XPATH和LXML类库**:**

lxml是一款高性能的 Python HTML/XML 解析器，我们可以利用XPath，来快速的定位特定元素以及获取节点信息

使用XPath解析时如果取不到内容，就把user-agent改为IE的，这样一般就能取出来了。

### 什么是XPATH:

XPath (XML Path Language) 是一门在 HTML\XML 文档中查找信息的语言，

可用来在 HTML\XML 文档中对元素和属性进行遍历。

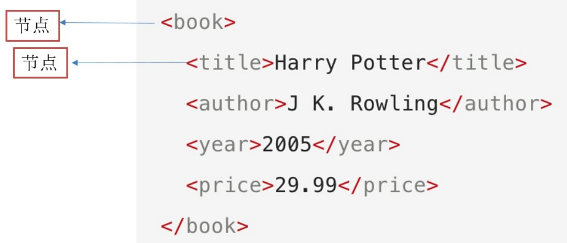
W3School官方文档: <http://www.w3school.com.cn/xpath/index.asp>

### 认识XML

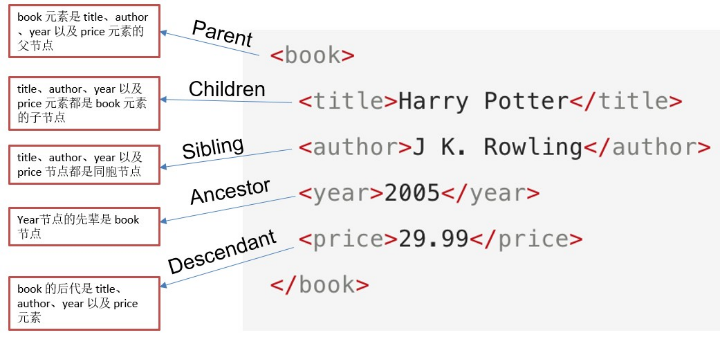


### XML的节点关系

节点的概念: 每个XML的标签我们都称之为节点



### XML的节点关系



### XPATH节点选择

常用节点选择工具:

* Chrome插件 XPath Helper
* 开源的XPath表达式编辑工具:XMLQuire(XML格式文件可用)
* Firefox插件 XPath Checker

### 节点选择语法

XPath 使用路径表达式来选取 XML 文档中的节点或者节点集。这些路径表达式和我们在常规的电脑文件系统中看到的表达式非常相似。

|  |  |  |  |
| --- | --- | --- | --- |
| **表达式** | **描述** | **示例** | **结果** |
| nodename | 选取此节点的所有子节点 | bookstore | 选取bookstore下所有的子节点 |
| / | 如果是在最前面，代表从根节点选取。否则选择某节点下的某个节点 | /bookstore | 选取根元素下所有的bookstore节点 |
| // | 从全局节点中选择节点，随便在哪个位置 | //book | 从全局节点中找到所有的book节点 |
| @ | 选取某个节点的属性 | //book[@price] | 选择所有拥有price属性的book节点 |
| . | 选取当前节点 | ./a | 选取当前节点下的a标签 |
| .. | 选取当前节点的父节点 |  |  |



//title/text() 获取title的文本内容

使用chrome插件选择标签时候，选中时，插件会给选中的标签添加属性class="xh-highlight", 不要认为这是html网页原本的样式.

#### 查找某个特定的节点或者包含某个指定的值的节点



注意: 谓词中的下标是从1开始的，不是从0开始的。

#### 选择未知节点



//\*[@id="page"] 选择id="page"的元素

//node()[@id="page"] 与上面的功能相同

#### 选取若干路径

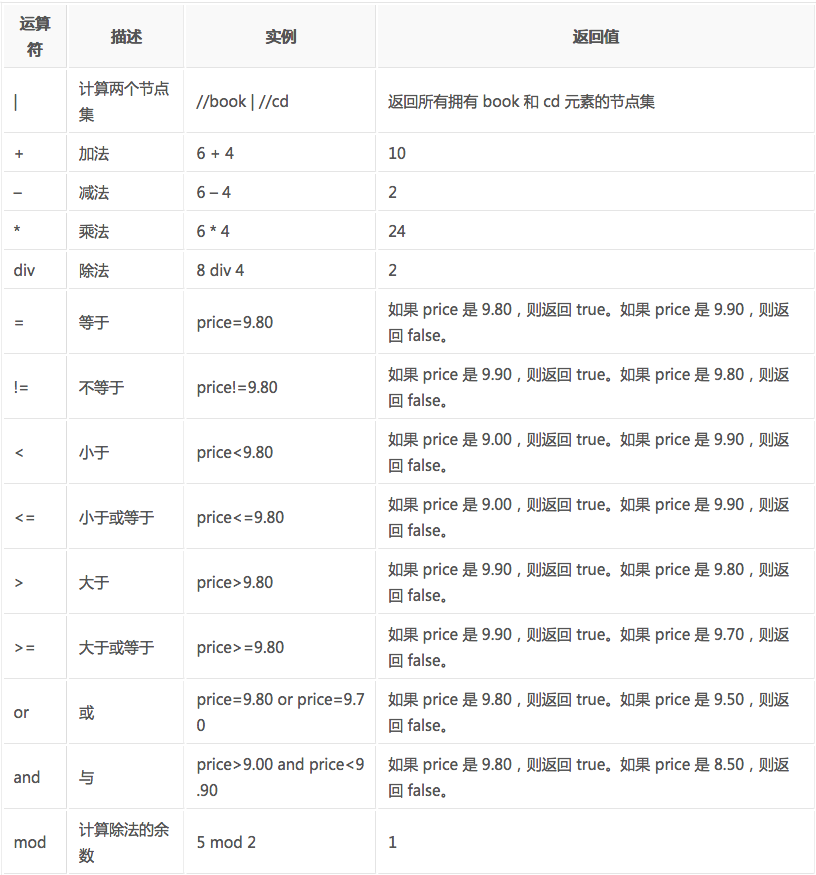


//div[@class="job"]//div[@class="title"] | //div[@class="beijing"]

表示选择class="job"的div标签下的class="title"的div标签, 或者class="beijing"的div标签, 注意后一个选择器也是从根节点下选择的, 不是从class="job"的div标签下开始选择的.

| 两边表示的是并列的关系, 再边是独立的, 是两个单独的选择语句.

### 运算符：



其中的and比较常用,

//dl[@class="job\_detail" and @id="job\_detail"]

表示选择同时具有class="job\_detail" 和 id="job\_detail" 属性的dl标签

#### 模糊匹配

有时候某个属性中包含了多个值，那么可以使用`contains`函数。

### xpath的包含

- `//div[contains(@class, 'i')]`

//div[contains(@class, 'job\_detail')]

xpath的更多语法:

<https://msdn.microsoft.com/zh-cn/library/ms256039(v=vs.80).aspx>

### lxml库

* 使用入门:
  1. 导入lxml 的 etree 库

from lxml import etree

* + 利用etree.HTML，将html字符串转化为Element对象
  + Element对象具有xpath的方法
  + 可以接收str或bytes类型的字符串, 即可以传入response.content及response.content.decode()

html = etree.HTML(text)

html.xpath("//div[@class='container']")

* lxml 可以自动修正 html 代码

### lxml使用方法

# coding=utf-8

**from** **lxml** **import** etree

text = ''' <div> <ul>

<li class="item-1"><a>first item</a></li>

<li class="item-1"><a href="link2.html">second item</a></li>

<li class="item-inactive"><a href="link3.html">third item</a></li>

<li class="item-1"><a href="link4.html">fourth item</a></li>

<li class="item-0"><a href="link5.html">fifth item</a>

</ul> </div> '''

html = etree.HTML(text)

**print**(html)

#查看element对象中包含的字符串

# print(etree.tostring(html).decode())

#获取class为item-1 li下的a的herf

ret1 = html.xpath("//li[@class='item-1']/a/@href")

**print**(ret1)

#获取class为item-1 li下的a的文本

ret2 = html.xpath("//li[@class='item-1']/a/text()")

**print**(ret2)

#每个li是一条新闻，把url和文本组成字典

# 直接从页面中分别提取出内容, 如果html的结构发生变化, 就可能不能得到正确的内容, 不同规则得到的内容之间不匹配

**for** href **in** ret1:

item = {}

item["href"] = href

item["title"] = ret2[ret1.index(href)]

**print**(item)

**print**("\*"\*100)

#分组，根据li标签进行分组，对每一组继续写xpath, 先获取某一节点, 对此节点的内容进行提取, 再处理下一个节点, 就能解决不同的匹配规则得到的数据不匹配的问题

ret3 = html.xpath("//li[@class='item-1']")

**print**(ret3)

**for** i **in** ret3:

item= {}

item["title"] = i.xpath("a/text()")[0] **if** len(i.xpath("./a/text()"))>0 **else** None

item["href"] = i.xpath("./a/@href")[0] **if** len( i.xpath("./a/@href"))>0 **else** None

**print**(item)

### 动手

用XPath来做一个简单的爬虫，爬取某个贴吧里的所有帖子，获取每个帖子的标题，连接和帖子中图片

### 使用lxml的百度贴吧爬虫

# coding=utf-8

**import** **requests**

**from** **lxml** **import** etree

**import** **json**

**class** **TiebaSpider**:

**def** \_\_init\_\_(self, tieba\_name):

self.tieba\_name = tieba\_name

# 度贴吧上pc端很多内容都被注释掉了, 使用js来取消注释, 所以优先考虑爬取移动端的页面. 使用3g版的页面来进行爬取

self.start\_url = "http://tieba.baidu.com/mo/q----, sz@320\_240-1-3---2/m?kw="+tieba\_name+"&pn=0"

self.part\_url = "http://tieba.baidu.com/mo/q----, sz@320\_240-1-3---2/"

self.headers= {"User-Agent":"Mozilla/5.0 (Linux; Android 5.1.1; Nexus 6 Build/LYZ28E) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Mobile Safari/537.36"}

#发送请求，获取响应

**def** parse\_url(self, url):

**print**(url)

response = requests.get(url, headers=self.headers)

# 注意这里不能使用utf-8进行解码, 否则会在etree.HTML(html\_str)时出错, Unicode strings with encoding declaration are not supported, please use bytes input or XML fragments without declaration

**return** response.content

**def** get\_content\_list(self, html\_str):#提取某一个贴吧列表页的数据

html = etree.HTML(html\_str)

div\_list = html.xpath("//div[contains(@class, 'i')]") #根据div分组

# 保存一个帖吧的数据

content\_list = []

**for** div **in** div\_list:

item = {}

# 帖子标题

item["title"] = div.xpath("./a/text()")[0] **if** len(div.xpath("./a/text()"))>0 **else** None

# 提取的url地址是不完整的, 要添加完整.

# 如果if条件不成立, 就把None值赋值给前面+连接的整体, 即如果没有提取到内容, item["href"] = None, 不是self.part\_url + None

item["href"] = self.part\_url+div.xpath("./a/@href")[0] **if** len(div.xpath("./a/@href"))>0 **else** None

# 进入到每一个帖子中, 获取帖子中的图片地址, 通过列表返回来所有图片的url地址

item["img\_list"] = self.get\_img\_list(item["href"], [])

# 上面取到的图片是缩略图, 要对其进行解码, 然后获取大图

item["img\_list"] = [requests.utils.unquote(i).split("src=")[-1] **for** i **in** item["img\_list"]]

# 把一个贴吧下每个帖子的内容都保存到列表中

content\_list.append(item)

#提取贴吧下一页的url地址, 最后一页没有下一页的标签

next\_url = self.part\_url+html.xpath("//a[text()='下一页']/@href")[0] **if** len(html.xpath("//a[text()='下一页']/@href"))>0 **else** None

**return** content\_list, next\_url

# 进入到每一个帖子中, 获取帖子中的图片地址

**def** get\_img\_list(self, detail\_url, total\_img\_list): #获取帖子中的所有的图片

#3.2请求列表页的url地址，获取详情页的第一页

detail\_html\_str = self.parse\_url(detail\_url)

detail\_html = etree.HTML(detail\_html\_str)

#3.3提取详情页第一页的图片，提取下一页的地址

img\_list = detail\_html.xpath("//img[@class='BDE\_Image']/@src")

# 使用递归调用自己时, 下一页的数据会把上一页的数据覆盖掉, 所以要把上一页的数据保存起来. 使用一个列表来保存.

total\_img\_list.extend(img\_list)

#3.4请求详情页下一页的地址，进入循环3.2-3.4

detail\_next\_url = detail\_html.xpath("//a[text()='下一页']/@href")

**if** len(detail\_next\_url)>0:

detail\_next\_url = self.part\_url + detail\_next\_url[0]

# 递归调用自己. 直到获取到最后一页的数据为止. 递归要写return, 不然只是调用了, 不会返回数据, 不写return时就会返回None.

# 每次递归调用时, 都把total\_img\_list传递过去, 这样每次的total\_img\_list就保存了本页之前所有页面的url地址, 这样就解决了下一次递归调用覆盖上一次的问题.

**return** self.get\_img\_list(detail\_next\_url, total\_img\_list)

# else:

# return total\_img\_list

**return** total\_img\_list

#保存数据

**def** save\_content\_list(self, content\_list):

# 把每个帖吧中的帖子保存到不同的文件中

file\_path = self.tieba\_name+".txt"

**with** open(file\_path, "a", encoding="utf-8") **as** f:

**for** content **in** content\_list:

f.write(json.dumps(content, ensure\_ascii=False, indent=2))

f.write("**\n**")

**print**("保存成功")

**def** run(self):#实现主要逻辑

# 刚开始执行程序时使用第一页, 然后从第一页中提取下一页的url, 循环请求下一页并从中提取数据

next\_url = self.start\_url

# 循环执行下一页的提取, 直到最后一页

**while** next\_url **is** **not** None:

#1.start\_url

#2.发送请求，获取响应

html\_str = self.parse\_url(next\_url)

#3.提取数据，提取下一页的url地址

#3.1提取列表页的url地址和标题

#3.2请求列表页的url地址，获取详情页的第一页

#3.3提取详情页第一页的图片，提取下一页的地址

#3.4请求详情页下一页的地址，进入循环3.2-3.4

content\_list, next\_url = self.get\_content\_list(html\_str)

#4.保存数据

self.save\_content\_list(content\_list)

#5.请求下一页的url地址，进入循环2-5不

**if** \_\_name\_\_ == '\_\_main\_\_':

tieba\_spider = TiebaSpider("做头发")

tieba\_spider.run()

百度贴吧多

1、爬取糗事百科段子，页面的URL是

 http://www.qiushibaike.com/8hr/page/1

### 使用lxml的糗事百科段子爬虫

# coding=utf-8

**import** **requests**

**from** **lxml** **import** etree

**class** **QiubaiSpdier**:

**def** \_\_init\_\_(self):

self.url\_temp = "https://www.qiushibaike.com/8hr/page/{}/"

self.headers = {"User-Agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

# 构造url列表, 因为清楚的知道一共有多少页, 每一页的url地址也是已知的. 构造完列表就可以方便的发送请求了.

**def** get\_url\_list(self):

**return** [self.url\_temp.format(i) **for** i **in** range(1, 14)]

**def** parse\_url(self, url):

**print**(url)

response = requests.get(url, headers=self.headers)

**return** response.content.decode()

**def** get\_content\_list(self, html\_str): #提取数据

html = etree.HTML(html\_str)

div\_list = html.xpath("//div[@id='content-left']/div") #分组

content\_list = []

**for** div **in** div\_list:

item= {}

item["content"] = div.xpath(".//div[@class='content']/span/text()")

# 获取的content是一个列表, 其中有多个元素, 把其中的\n去掉

item["content"] = [i.replace("**\n**", "") **for** i **in** item["content"]]

item["author\_gender"] = div.xpath(".//div[contains(@class, 'articleGender')]/@class")

item["author\_gender"] = item["author\_gender"][0].split(" ")[-1].replace("Icon", "") **if** len(item["author\_gender"])>0 **else** None

item["auhtor\_age"] = div.xpath(".//div[contains(@class, 'articleGender')]/text()")

item["auhtor\_age"] = item["auhtor\_age"][0] **if** len(item["auhtor\_age"])>0 **else** None

item["content\_img"] = div.xpath(".//div[@class='thumb']/a/img/@src")

item["content\_img"] = "https:"+item["content\_img"][0] **if** len(item["content\_img"])>0 **else** None

item["author\_img"] = div.xpath(".//div[@class='author clearfix']//img/@src")

item["author\_img"] = "https:"+item["author\_img"][0] **if** len(item["author\_img"])>0 **else** None

item["stats\_vote"] = div.xpath(".//span[@class='stats-vote']/i/text()")

item["stats\_vote"] = item["stats\_vote"][0] **if** len(item["stats\_vote"])>0 **else** None

content\_list.append(item)

**return** content\_list

**def** save\_content\_list(self, content\_list): #保存

**for** i **in** content\_list:

**print**(i)

**def** run(self): #实现主要逻辑

#1.url\_list, 构造url地址列表

url\_list = self.get\_url\_list()

#2.遍历，发送请求，获取响应

**for** url **in** url\_list:

html\_str = self.parse\_url(url)

#3.提取数据

content\_list = self.get\_content\_list(html\_str)

#4.保存

self.save\_content\_list(content\_list)

**if** \_\_name\_\_ == '\_\_main\_\_':

qiubai = QiubaiSpdier()

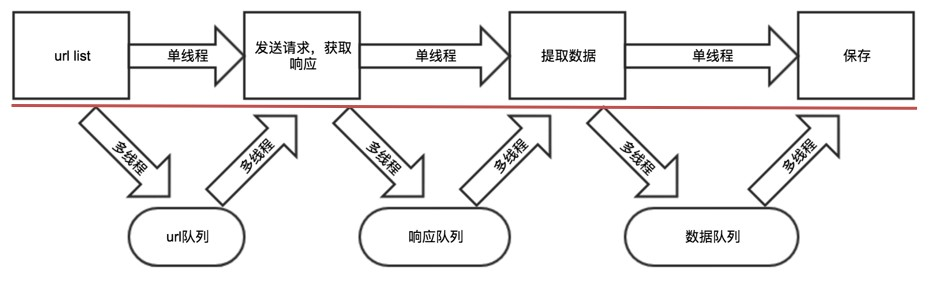
qiubai.run()

2、动手把上述的爬虫改为多线程爬虫

对于贴吧, 每个贴吧的页数不一样, 不能通过构造的方法来获取url地址, 而是应该从页面中提取url的地址. 在每一页中都提取下一页的url地址

pc端的页面中很多内容都是被注释掉的, 可以使用正则匹配把注释去掉, 提取其中的内容, 也可以优先考虑爬取移动端, 并且获取最原始的3g版的页面. 其中没有js, 所有需要的内容都在html中.

对于手机版网页, 使用view source查看时, 因为没有切换到手机版的选项, 是使用view source再次发送的请求, 获取的可能是pc端的代码, 所以手机端的要在检查元素中和network的response中来查看



多线程爬虫, 在处理的比较慢的地方使用多线程来完成

放在队列中, 而不是放在列表中, 是考虑到数据的安全, 不同的线程同时读取某一个url地址, 同时爬取. 放在队列中时, 一个线程在取数时另一个线程是无法读取的.

通过队列联系各个模块, 模块之间不再有直接的联系, 如果某一个模块出现问题, 其它的模块功能也不受影响.

### 使用lxml的糗事百科多线程爬虫

# coding=utf-8

**import** **requests**

**from** **lxml** **import** etree

**import** **threading**

# 使用队列, 就不用在多个线程之间传递参数

**from** **queue** **import** Queue

**class** **QiubaiSpdier**:

**def** \_\_init\_\_(self):

self.url\_temp = "https://www.qiushibaike.com/8hr/page/{}/"

self.headers = {"User-Agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_13\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}

# 初始化3个队列

self.url\_queue = Queue()

self.html\_queue = Queue()

self.content\_queue = Queue()

**def** get\_url\_list(self):

# return [self.url\_temp.format(i) for i in range(1, 14)]

**for** i **in** range(1, 4):

# 把url地址放到队列中

self.url\_queue.put(self.url\_temp.format(i))

**def** parse\_url(self):

# url的队列中有很多地址, 所以要使用循环从中逐个取出

**while** True:

# 从url的队列中取出来url地址

url = self.url\_queue.get()

**print**(url)

response = requests.get(url, headers=self.headers)

# 把响应也放到响应队列中去, 然后在get\_content\_list中从响应队列中取出来进行处理

self.html\_queue.put(response.content.decode())

# 必须与task\_done()配合使用url\_queue队列的长度才会减1

self.url\_queue.task\_done()

**def** get\_content\_list(self): #提取数据

**while** True:

html\_str = self.html\_queue.get()

html = etree.HTML(html\_str)

div\_list = html.xpath("//div[@id='content-left']/div") #分组

content\_list = []

**for** div **in** div\_list:

item= {}

item["content"] = div.xpath(".//div[@class='content']/span/text()")

item["content"] = [i.replace("**\n**", "") **for** i **in** item["content"]]

item["author\_gender"] = div.xpath(".//div[contains(@class, 'articleGender')]/@class")

item["author\_gender"] = item["author\_gender"][0].split(" ")[-1].replace("Icon", "") **if** len(item["author\_gender"])>0 **else** None

item["auhtor\_age"] = div.xpath(".//div[contains(@class, 'articleGender')]/text()")

item["auhtor\_age"] = item["auhtor\_age"][0] **if** len(item["auhtor\_age"])>0 **else** None

item["content\_img"] = div.xpath(".//div[@class='thumb']/a/img/@src")

item["content\_img"] = "https:"+item["content\_img"][0] **if** len(item["content\_img"])>0 **else** None

item["author\_img"] = div.xpath(".//div[@class='author clearfix']//img/@src")

item["author\_img"] = "https:"+item["author\_img"][0] **if** len(item["author\_img"])>0 **else** None

item["stats\_vote"] = div.xpath(".//span[@class='stats-vote']/i/text()")

item["stats\_vote"] = item["stats\_vote"][0] **if** len(item["stats\_vote"])>0 **else** None

content\_list.append(item)

self.content\_queue.put(content\_list)

# 为什么要把task\_done放在最后面. 为什么不在get取出来一个数据之后直接task\_done呢, 假设html\_queue中有一条数据, 而其它的queue中都没有数据, 那么只要get出来最后一条数据, 因为所有队列为空, 不等这里的其它操作执行结束, 主程序就会结束退出了. 所以要把取出的数据的操作执行完后再task\_done

self.html\_queue.task\_done()

**def** save\_content\_list(self): #保存

**while** True:

content\_list = self.content\_queue.get()

**for** i **in** content\_list:

# print(i)

**pass**

self.content\_queue.task\_done()

**def** run(self): #实现主要逻辑

# 分别使用不同的线程处理不同的任务, 但要启动每个线程, 可以把线程放在一个列表中, 依次启动即可.

thread\_list = []

#1.url\_list

t\_url = threading.Thread(target=self.get\_url\_list)

thread\_list.append(t\_url)

#2.遍历，发送请求，获取响应

**for** i **in** range(20):

t\_parse = threading.Thread(target=self.parse\_url)

thread\_list.append(t\_parse)

#3.提取数据

**for** i **in** range(2):

t\_html = threading.Thread(target=self.get\_content\_list)

thread\_list.append(t\_html)

#4.保存

t\_save = threading.Thread(target=self.save\_content\_list)

thread\_list.append(t\_save)

# 开启线程列表中的每一个子线程

**for** t **in** thread\_list:

t.setDaemon(True) #把子线程设置为守护线程，该线程不重要, 主线程结束，子线程结束. 这样也就结束了每个方法中的while True死循环

t.start()

**for** q **in** [self.url\_queue, self.html\_queue, self.content\_queue]:

q.join() #让主线程等待阻塞，等待队列的任务完成之后再完成. 队列全部为空时主线程结束, 然后子线程结束.

**print**("主线程结束")

**if** \_\_name\_\_ == '\_\_main\_\_':

qiubai = QiubaiSpdier()

qiubai.run()

使用xpath时必须保证在response中和检查元素的Elements中看到的内容是一样的, 才能使用检查元素中的Elements或者xpath helper来对页面内容进行提取. 可以在response中搜索页面中显示的内容来判断. 如果页面内容是使用js动态生成的, 就只能在response中进行匹配.

### 总结: 实现爬虫的套路

### 实现爬虫的套路

- 准备url

- 准备start\_url

- url地址规律不明显，总数不确定

- 通过代码提取下一页的url

- xpath

- 寻找url地址，部分参数在当前的响应中(比如，当前页码数和总的页码数在当前的响应中)

- 准备url\_list

- 页码总数明确

- url地址规律明显

- 发送请求，获取响应

- 添加随机的User-Agent, 反反爬虫

- 添加随机的代理ip，反反爬虫

- 在对方判断出我们是爬虫之后，应该添加更多的headers字段，包括cookie

- cookie的处理可以使用session来解决. 在init中定义self.session = request.session(), 在parse\_url中使用session发送请求, 就能带上上一次服务器设置的cookie值. response = self.session.get(url, headers=self.headers). 不能在parse\_url中使用self.session = reqeust.session(), 因为这样会每次都使用新的session去发送请求, 没有把上一次服务器的cookie发送给服务器. 所以要在init中定义一个全局的session

- 准备一堆能用的cookie，组成cookie池. 如果所有的请求都使用同一个cookie, 也会被识别为爬虫.

- 如果不登录

- 准备刚开始能够成功请求对方网站的cookie，即接收对方网站设置在response的cookie

- 下一次请求的时候，使用之前的列表中的cookie来请求

- 如果登录

- 准备多个账号

- 使用程序获取每个账号的cookie

- 之后请求登录之后才能访问的网站随机的选择cookie

- 提取数据

- 确定数据的位置

- 如果数据在当前的url地址中

- 提取的是列表页的数据

- 直接请求列表页的url地址，不用进入详情页

- 提取的是详情页的数据

- 1. 确定url

- 2. 发送请求

- 3. 提取数据

- 4. 返回

- 如果数据不在当前的url地址中

- 在其他的响应中，寻找数据的位置

- 1. 从network中从上往下找

- 2. 使用chrome中的过滤条件，选择出了js, css, img之外的按钮, 不准确

- 3. 使用chrome的search all file，搜索数字和英文

- 数据的提取

- xpath, 从html中提取整块的数据，先分组，之后每一组再提取

- re，提取max\_time, price, html中的json字符串

- json

- 保存

- 保存在本地，text, json, csv

- 保存在数据库

### 后续爬虫代码的建议

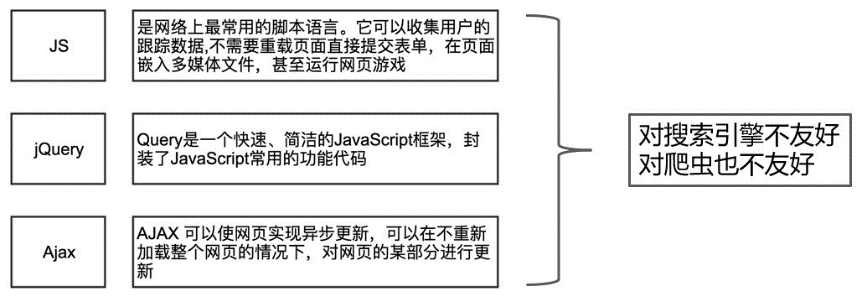


**动态HTML处理**

# 第三部分课程概要

1. **动态html介绍**
2. **selenium和phantomjs**
3. **机器视觉和tesseract介绍**

## 动态HTML技术了解



### Ajax动态加载数据的获取

**做爬虫最需要关注的不是页面信息，而是页面信息的数据来源. 所以有的时候并不一定要获取网页上显示的内容.**

AJAX方式加载的页面，数据来源一定是JSON, 拿到了JSON，就是拿到了网页的数据

访问

<https://movie.douban.com/typerank?type_name=科幻&type=17&interval_id=100:90&action>=

向下拖动加载更多的内容, 此时浏览器中显示的地址始终不发生变化, 这就是使用ajax异步请求的方式加载的数据.

在抓包软件中找到请求的地址：

GET https://movie.douban.com/j/chart/top\_list?type=17&interval\_id=100%3A90&action=&start=20&limit=20 HTTP/1.1

Host: movie.douban.com

Connection: keep-alive

Accept: \*/\*

X-Requested-With: XMLHttpRequest

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.81 Safari/537.36

DNT: 1

Referer: https://movie.douban.com/typerank?type\_name=%E7%A7%91%E5%B9%BB&type=17&interval\_id=100:90&action=

Accept-Encoding: gzip, deflate, sdch, br

Accept-Language: en-US, en;q=0.8, zh-CN;q=0.6, zh;q=0.4

Cookie:

构建post的form

"start":"0",

"limit":"1",

"interval\_id":"100:90",

"type":"17",

"start":"0"

"limit":"20"

**import** urllib.parse, urllib.request

*#https://movie.douban.com/j/chart/top\_list?type=17&interval\_id=100%3A90&action=&start=20&limit=20*

get\_url = **" https://movie.douban.com/j/chart/top\_list?type=17&interval\_id=100%3A90&action"**

headers={**"User-Agent"**: **"Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko"**}

*# 变动的是这两个参数，从start开始往后显示limit个*

formdata = {

**'start'**:**'0'**,

**'limit'**:**'20'**

}

data = urllib.parse.urlencode(formdata).encode(**"utf-8"**)

request = urllib.request.Request(get\_url, data = data, headers = headers)

response = urllib.request.urlopen(request)

html = response.read()

print(response.read().decode(**"utf-8"**))

**with** open(**'douban.html'**, **'wb'**) **as** f:

f.write(html)

如果无法写入文件, 把写入文件的代码改为如下

print(html.decode(**"utf-8"**))

**with** open(**'douban.html'**, **'w'**, encoding=**"utf-8"**) **as** f:

f.write(html.decode(**"utf-8"**))

## Selenium和PhantomJS

* Selenium

Selenium是一个Web的自动化测试工具，最初是为网站自动化测试而开发的，Selenium 可以直接运行在浏览器上，它支持所有主流的浏览器(包括PhantomJS这些无界面的浏览器) ，可以接收指令，让浏览器自动加载页面，获取需要的数据，甚至页面截屏

* PhantomJS

PhantomJS 是一个基于Webkit的“无界面”(headless)浏览器，它会把网站加载到内存并执行页面上的 JavaScript

<http://selenium-python-zh.readthedocs.io/en/latest/waits.html>

## Selenium和PhantomJS入门

1. 加载网页:
   1. from selenium import webdriver
   2. driver = webdriver.PhantomJS(“c:…/pantomjs.exe”)
   3. driver.get("http://www.baidu.com/")
   4. driver.save\_screenshot("长城.png")
2. 定位和操作:
   1. driver.find\_element\_by\_id(“kw”).send\_keys(“长城”)
   2. driver.find\_element\_by\_id("su").click()
3. 查看请求信息:
   1. driver.page\_source
   2. driver.get\_cookies()
   3. driver.current\_url
4. 退出
   1. driver.close() #退出当前页面
   2. driver.quit() #退出浏览器

## cookie

* Cookie相关用法:
  1. {cookie['name']: cookie['value'] for cookie in driver.get\_cookies()}
  2. driver.delete\_cookie("CookieName")
  3. driver.delete\_all\_cookies()

## selenium初级使用方法

# try\_selenium

# coding=utf-8

**from** **selenium** **import** webdriver

**import** **time**

#实例化一个浏览器

driver = webdriver.Chrome()

# driver = webdriver.PhantomJS()

#设置窗口大小

driver.set\_window\_size(1920, 1080)

# 最大化窗口

driver.maximize\_window()

# 发送请求

driver.get("http://www.baidu.com")

# 进行页面截屏

# driver所有的操作都是在页面加载完成后才进行的

driver.save\_screenshot("./baidu.png")

# 元素定位的方法find\_elements\_by\_id, 使用send\_keys向元素输入文本

driver.find\_element\_by\_id("kw").send\_keys("python")

# 点击搜索按钮

driver.find\_element\_by\_id("su").click()

# driver 获取html字符串

# 浏览器中elements的内容, 包含js, css, 想要从其中提取数据, 可以使用xpath的elements方法

**print**(driver.page\_source)

# current\_url是点击搜索之后的地址

**print**(driver.current\_url)

# driver获取cookie

cookies = driver.get\_cookies()

**print**(cookies)

**print**("\*"\*100)

# 字典表达式

cookies = {i["name"]:i["value"] **for** i **in** cookies}

**print**(cookies)

# 退出浏览器

time.sleep(3)

driver.quit()

## 动手练习

1. 模拟登陆豆瓣网

### 使用selenium模拟登陆豆瓣

# coding=utf-8

**from** **selenium** **import** webdriver

**import** **time**

#实例化driver

driver = webdriver.Chrome()

driver.get("https://www.douban.com/")

driver.find\_element\_by\_id("form\_email").send\_keys("784542623@qq.com")

driver.find\_element\_by\_id("form\_password").send\_keys("zhoudawei123")

# 在此期间手动输入验证码

time.sleep(5)

driver.find\_element\_by\_class\_name("bn-submit").click()

#获取cookie

cookies = {i["name"]:i["value"] **for** i **in** driver.get\_cookies()}

**print**(cookies)

time.sleep(3)

driver.quit()

## 页面元素定位

* 用法:
  1. find\_element\_by\_id (返回一个)
  2. find\_elements\_by\_xpath (返回一个列表)
  3. find\_elements\_by\_link\_text
  4. find\_elements\_by\_partial\_link\_text
  5. find\_elements\_by\_tag\_name
  6. find\_elements\_by\_class\_name
  7. find\_elements\_by\_css\_selector
* 注意点:
  1. find\_element 和find\_elements的区别: 返回一个和返回一个列表
  2. by\_link\_text和by\_partial\_link\_text的区别: 全部文本和包含某个文本
  3. by\_css\_selector的用法: #food span.dairy.aged
  4. by\_xpath中获取属性和文本需要使用get\_attribute() 和.text

### 使用selenium的内涵社区段子爬虫

# coding=utf-8

**from** **selenium** **import** webdriver

driver = webdriver.Chrome()

driver.get("http://neihanshequ.com/")

ret1 = driver.find\_element\_by\_xpath("//ul[@id='detail-list']/li")

# 使用find\_element时, 如果找不到, 就会报错.

**print**(ret1)

ret1 = driver.find\_elements\_by\_xpath("//ul[@id='detail-list']/li")

# 使用find\_elements\_by\_xpath时返回列表, 如果找不到, 就返回空列表. 所以在提取下一列的地址时就要使用find\_elements.

**print**(ret1)

# 使用find\_elements\_by\_xpath时得到的是列表, 不能直接对结果使用xpath提取信息, 也不能使用.text或get\_attribute方法来获取文本内容, 必须要遍历出列表中的元素才能使用xpath提取信息

**for** li **in** ret1:

# find\_element\_by\_xpath不能获取html标签中的字符串内容, 只能获取到html标签, 即element, 得到element对象, 想要获取标签的内容或者属性值, 就要使用.text方法和get\_attribute方法

**print**(li.find\_element\_by\_xpath(".//h1/p").text)

driver.quit()

### find\_element\_by\_link\_text

# coding=utf-8

**from** **selenium** **import** webdriver

driver = webdriver.Chrome()

driver.get("http://neihanshequ.com/")

driver.get("https://www.baidu.com/s?ie=utf-8&f=8&rsv\_bp=1&rsv\_idx=1&ch=&tn=baidu&bar=&wd=python&rn=&oq=&rsv\_pq=87739988000939bf&rsv\_t=b194dxdCny6hrJFXQrh4D6bavkKZwfpeT4s7j7V6AvGfiiAvTgxqGAvWbCM&rqlang=cn")

# find\_element\_by\_link\_text通过链接的文字来选择html标签

# 使用get\_attribute方法获取url地址时可以对地址进行自动补全

**print**(driver.find\_element\_by\_link\_text("下一页>").get\_attribute("href"))

# find\_element\_by\_partial\_link\_text文本中包含"下一页"的a标签

**print**(driver.find\_element\_by\_partial\_link\_text("下一页").get\_attribute("href")

driver.quit()

### selenium使用的注意点

### selenium使用的注意点

- 获取文本和获取属性

- 先定位到元素，然后调用`.text`或者`get\_attribute`方法来去

- selenium获取的页面数据是浏览器中elements的内容

- find\_element和find\_elements的区别

- find\_element返回一个element，如果没有会报错

- find\_elements返回一个列表，没有就是空列表. 使用elements时要遍历后才能使用.text或.get\_attribute方法

- 在判断是否有下一页的时候，使用find\_elements来根据结果的列表长度来判断

- 如果页面中含有iframe、frame，需要先调用driver.switch\_to.frame的方法切换到frame中才能定位元素

### 云打码平台

<http://www.yundama.com/>

使用云打码平台识别豆瓣验证吗, 注意要在yundama/dama.py中更改图片验证码的类型

也可以使用PIL模块对验证码区域进行截图

### 使用云打码平台处理豆瓣登陆验证码

# coding=utf-8

**from** **selenium** **import** webdriver

**import** **time**

**import** **requests**

**from** **yundama.dama** **import** indetify

#实例化driver

driver = webdriver.Chrome()

driver.get("https://www.douban.com/")

driver.find\_element\_by\_id("form\_email").send\_keys("784542623@qq.com")

driver.find\_element\_by\_id("form\_password").send\_keys("zhoudawei123")

#识别验证码

captcha\_image\_url = driver.find\_element\_by\_id("captcha\_image").get\_attribute("src")

captcha\_content = requests.get(captcha\_image\_url).content

captcha\_code = indetify(captcha\_content)

**print**("验证码的识别结果为:", captcha\_code)

#输入验证码

driver.find\_element\_by\_id("captcha\_field").send\_keys(captcha\_code)

driver.find\_element\_by\_class\_name("bn-submit").click()

#获取cookie

cookies = {i["name"]:i["value"] **for** i **in** driver.get\_cookies()}

**print**(cookies)

time.sleep(3)

driver.quit()

### 验证码的识别总结

#### 验证码的识别

- url不变，验证码不变

- 请求验证码的地址，获得响应，识别

- url不变，验证码会变

- 思路: 对方服务器返回验证码的时候，会和每个用户的信息和验证码进行一个对应，之后，在用户发送post请求的时候，会对比post请求中发送的验证码和当前用户真正的存储在服务器端的验证码是否相同.

- 1.实例化session

- 2.使用seesion请求登录页面，获取验证码的地址

- 3.使用session请求验证码，识别

- 4.使用session发送post请求. 保证两次发送的请求使用同一个session.

- 5.如果这里使用request来发送请求验证码和发送post请求, 必须要保存上一次请求的cookie, 并在下一次请求时把上一次的cookie传递给服务器, 否则2次请求的cookie就不是服务器返回的cookies, 就会与服务器生成的验证码不一致, 就会出错.

- 使用selenium登录，遇到验证码

- url不变，验证码不变，同上

- url不变，验证码会变

- 1.selenium请求登录页面，同时拿到验证码的地址

- 2.获取登录页面中driver中的cookie，交给requests模块发送验证码的请求，识别

- 3.输入验证码，点击登录

- 登录模块比较复杂, 可以使用selenium来处理, 获取到cookie时再交给requests来处理, 就会很快.

### 使用selenium 的iframe实例,

iframe在网页中嵌套另一个网页, 相当于2个网页

### 模拟登陆QQ邮箱

# coding=utf-8

**from** **selenium** **import** webdriver

**import** **time**

driver = webdriver.Chrome()

driver.get("https://mail.qq.com/")

#切换到iframe

driver.switch\_to.frame("login\_frame")

driver.switch\_to\_frame("login\_frame")

# 页面中有iframe, 直接使用find\_element\_by\_xx会出错, 必须先要切换到iframe再执行操作

driver.find\_element\_by\_id("u").send\_keys("12312312312")

time.sleep(3)

driver.quit()

### 使用selenium提取bilibili中的元素

# coding=utf-8

**from** **selenium** **import** webdriver

**import** **time**

driver = webdriver.Chrome()

driver.get("https://www.bilibili.com/v/kichiku/mad/#/all/stow")

# 对于第一次的请求, 会等待页面加载完成后才进行元素的提取

**print**(driver.find\_element\_by\_xpath("//ul[@class='vd-list mod-2']/li//a[@class='title']").text)

# 点击下一页的按钮

driver.find\_element\_by\_xpath("//button[@class='nav-btn iconfont icon-arrowdown3']").click()

time.sleep(3)

# 翻页后会马上执行下一次操作, 因为元素还未加载出来, 会报错, 点击时会立即进行, 不会等待页面的加载结束. 可以先在执行之前等待一段时间.

**print**(driver.find\_element\_by\_xpath("//ul[@class='vd-list mod-2']/li//a[@class='title']

### selenium使用的注意点

### selenium使用的注意点

- selenium请求第一页的时候会等待页面加载完了之后在获取数据，但是在点击翻页之后，或者通过点击其它元素跳转之后, 会直接获取数据，此时可能会报错，因为数据还没有加载出来，需要time.sleep(3)

- selenium中find\_element\_by\_class\_name只能接收一个class对应的一个值，不能传入多个. 而在xpath中就要把class写全

## 页面等待

* 为什么需要等待
  1. 如果网站采用了动态html技术，那么页面上的部分元素出现时间便不能确定，这个时候就可以设置一个等待时间，强制要求在时间内出现，否则报错
* 强制等待
  1. time.sleep(10)
* 显式等待(了解)
  1. 显式等待指定某个条件，然后设置最长等待时间。如果在这个时间还没有找到元素，那么便会抛出异常了。
  2. WebDriverWait(driver, 10).until(EC.presence\_of\_element\_located((By.ID, "myDynamicElement")))
* 隐式等待(了解)
  1. 就是简单地设置一个最大等待时间，单位为秒。
  2. driver.implicitly\_wait(10)

注意显式等待和隐式等待都不常用, 因为在点击某个元素跳转之后, 会立即获取元素, 不会等到页面的元素加载完成, 此时显式等待和隐式等待都不能用.

### 页面等待案例

# coding=utf-8

**from** **selenium** **import** webdriver

**import** **time**

driver = webdriver.Chrome()

driver.get("https://www.bilibili.com/v/kichiku/mad/#/all/stow")

# 对于第一次的请求, 会等待页面加载完成后才进行元素的提取

**print**(driver.find\_element\_by\_xpath("//ul[@class='vd-list mod-2']/li//a[@class='title']").text)

driver.find\_element\_by\_xpath("//button[@class='nav-btn iconfont icon-arrowdown3']").click()

time.sleep(3)

#翻页后会马上执行下一步元素提取的操作, 因为元素还未加载出来, 会报错, 点击时会立即进行, 不会等待页面的加载结束

**print**(driver.find\_element\_by\_xpath("//ul[@class='vd-list mod-2']/li//a[@class='title']").text)

driver.quit()

### 使用selenium的斗鱼爬虫

# coding=utf-8

**from** **selenium** **import** webdriver

**import** **time**

**class** **DouyuSpider**:

**def** \_\_init\_\_(self):

self.start\_url = "https://www.douyu.com/directory/all"

self.driver = webdriver.Chrome()

# 使用selenium提取信息的速度也非常慢, 可以把driver.page\_source传递给lxml.etree, 使用lxml来提取并处理数据

**def** get\_content\_list(self):

li\_list = self.driver.find\_elements\_by\_xpath("//ul[@id='live-list-contentbox']/li")

# 把所有房间的信息都保存到一个列表中

content\_list = []

**for** li **in** li\_list:

# 把每个房间的信息保存到一个字典中

item = {}

item["room\_img"]=li.find\_element\_by\_xpath(".//span[@class='imgbox']/img").get\_attribute("src")

item["room\_title"] = li.find\_element\_by\_xpath("./a").get\_attribute("title")

item["room\_cate"] = li.find\_element\_by\_xpath(".//span[@class='tag ellipsis']").text

item["anchor\_name"] = li.find\_element\_by\_xpath(".//span[@class='dy-name ellipsis fl']").text

item["watch\_num"] = li.find\_element\_by\_xpath(".//span[@class='dy-num fr']").text

**print**(item)

content\_list.append(item)

#获取下一页的元素. 因为xpath在选择元素时必须要把所有的class写全, 对于最后一页, 因为样式不同, 所以在最后一页中得到的结果为空列表, 就可以以此来判断是否是最后一页, 并结束循环

next\_url = self.driver.find\_elements\_by\_xpath("//a[@class='shark-pager-next']")

next\_url = next\_url[0] **if** len(next\_url)>0 **else** None

**return** content\_list, next\_url

**def** save\_content\_list(self, content\_list):

**pass**

**def** run(self):#实现主要逻辑

#1.start\_url

#2.发送请求，获取响应

self.driver.get(self.start\_url)

#3.提取数据，提取下一页的元素

content\_list, next\_url = self.get\_content\_list()

#4.保存数据

self.save\_content\_list(content\_list)

#5.点击下一页元素，循环

**while** next\_url **is** **not** None:

next\_url.click()

time.sleep(3)

content\_list, next\_url = self.get\_content\_list()

self.save\_content\_list(content\_list)

**if** \_\_name\_\_ == '\_\_main\_\_':

douyu = DouyuSpider()

douyu.run()

练习1, 爬取网易云音乐中的所有大分类和小分类中的歌单

<http://music.163.com/#/discover/playlist>

练习2, 爬取天天基金上的基金排行中的内容

<http://fund.eastmoney.com/data/fundranking.html>

练习3, 爬取bilibi下所有分类下所有视频的弹幕

<https://www.bilibili.com/v/technology/digital/>

### bilibi视频弹幕爬虫, 不完善

# \_\*\_ coding: utf-8 \_\*\_

**import** **requests**, **re**

**from** **lxml** **import** etree

**class** **BlBl**:

**def** \_\_init\_\_(self, url):

self.url = url

# 哔哩哔哩弹幕url

self.danmu\_url= 'https://comment.bilibili.com/{}.xml'

self.headers = {

'User-Agent': 'Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36'

}

**def** get\_html(self, url):

"""发送请求, 返回响应"""

**return** requests.get(url, headers = self.headers).content.decode()

**def** save\_danmu(self, l, num):

"""保存弹幕"""

**print**('保存')

**with** open('./danmu/{}.txt'.format(num), 'a') **as** f:

**for** danmu\_str **in** l:

**print**(danmu\_str)

f.write(danmu\_str)

f.write("**\n**")

**def** get\_xml(self, li):

**print**(li)

**for** num **in** li: # 遍历cid列表, 设置弹幕请求url

# 拼接弹幕url, 调用函数, 发送请求, 获取结果

danmu\_xml = self.get\_html(self.danmu\_url.format(num)).encode() # 解析时说有问题, encode()转为byte型

# 将获取到的xml类型转换为etree对象

xml\_etr\_obj = etree.HTML(danmu\_xml)

# 获取弹幕列表

l = xml\_etr\_obj.xpath('//d/text()') # # 解析时说有问题, 获取结果是用encode()转为byte型

**print**('准备保存')

self.save\_danmu(l, num) # 保存

**def** run(self):

# 发送请求, 获取结果

bl\_html = self.get\_html(self.url)

**print**('正则获取cid')

# 提取此套的所有网页url\_地址和cid

li = re.findall(r"<option value='.\*?' cid='(\d+)'>", bl\_html)

**if** len(li) == 0: # 如果只有一个视频, 上边这个列表为空, 获取单个cid

li = re.findall(r"EmbedPlayer\('player', .\*?cid=(\d+)&aid", bl\_html)

# 请求xml的url并保存弹幕

self.get\_xml(li)

**if** \_\_name\_\_ == '\_\_main\_\_':

url = 'https://www.bilibili.com/video/av18198653/'

bili = BlBl(url)

bili.run()



chromedirver下载地址:<https://npm.taobao.org/mirrors/chromedriver>

phantomjs下载地址:<http://phantomjs.org/download.html>

注意chromedriver和桌面的chrome会有对应关系, 版本差别太大时可能会出错.

## 动手练习

1. 爬取斗鱼直播平台的所有房间信息

<https://www.douyu.com/directory/all>

## 使用starturl

实例化driver > 发送第1页的请求 > 对每一个房间进行分组 > 对每个分组进行信息提取 > 第1页信息提取完后保存到本地 > 点击下一页 > 请求下一页 > 再次分组, 提取, 保存

# coding=utf-8

from selenium import webdriver

import time

class DouyuSpider:

def \_\_init\_\_(self):

self.start\_url = "https://www.douyu.com/directory/all"

self.driver = webdriver.Chrome()

# 使用selenium提取信息的速度也非常慢, 可以把driver.page\_source传递给lxml.etree, 使用lxml来提取并处理数据

def get\_content\_list(self):

li\_list = self.driver.find\_elements\_by\_xpath("//ul[@id='live-list-contentbox']/li")

# 把所有房间的信息都保存到一个列表中

content\_list = []

for li in li\_list:

# 把每个房间的信息保存到一个字典中

item = {}

item["room\_img"]=li.find\_element\_by\_xpath(".//span[@class='imgbox']/img").get\_attribute("src")

item["room\_title"] = li.find\_element\_by\_xpath("./a").get\_attribute("title")

item["room\_cate"] = li.find\_element\_by\_xpath(".//span[@class='tag ellipsis']").text

item["anchor\_name"] = li.find\_element\_by\_xpath(".//span[@class='dy-name ellipsis fl']").text

item["watch\_num"] = li.find\_element\_by\_xpath(".//span[@class='dy-num fr']").text

print(item)

content\_list.append(item)

#获取下一页的元素. 因为xpath在选择元素时必须要把所有的class写全, 对于最后一页, 因为样式不同, 所以在最后一页中得到的结果为空列表, 就可以以此来判断是否是最后一页, 并结束循环

next\_url = self.driver.find\_elements\_by\_xpath("//a[@class='shark-pager-next']")

next\_url = next\_url[0] if len(next\_url)>0 else None

return content\_list, next\_url

def save\_content\_list(self, content\_list):

pass

def run(self):#实现主要逻辑

#1.start\_url

#2.发送请求，获取响应

self.driver.get(self.start\_url)

#3.提取数据，提取下一页的元素

content\_list, next\_url = self.get\_content\_list()

#4.保存数据

self.save\_content\_list(content\_list)

#5.点击下一页元素，循环

while next\_url is not None:

next\_url.click()

time.sleep(3)

content\_list, next\_url = self.get\_content\_list()

self.save\_content\_list(content\_list)

if \_\_name\_\_ == '\_\_main\_\_':

douyu = DouyuSpider()

douyu.run()

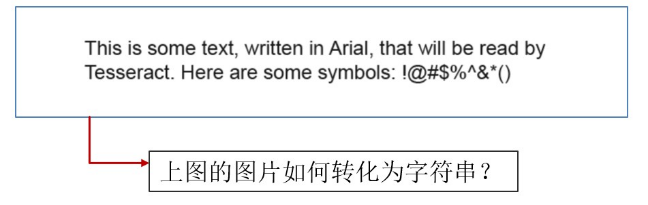
## Selenium总结

1. 应用场景:
   1. 动态html页面请求
   2. 登录获取cookies
2. 如何使用
   1. 导包并且实例化driver
   2. 发送请求
   3. 定位获取数据
   4. 保存
   5. 退出driver
3. Cookies相关方法:
   1. get\_cookies()
4. 页面等待
   1. 强制等待

## Tesseract

1. 定义:
   1. Tesseract是一个将图像翻译成文字的OCR库(光学文字识别，Optical Character Recognition)
   2. 安装:
   3. sudo apt-get install tesseract-ocr
2. 在python中调用Tesseract
   1. pip install pytesseract

Tesseract处理规范的文字



* 在终端中:

tesseract test.jpg test.txt

* 在python代码中

import pytesseract

from PIL import Image

image = Image.open(jpg)

pytesseract.image\_to\_string(image)

# coding=utf-8

**from** **PIL** **import** Image

**import** **pytesseract**

file\_path = "/User/zdw/Documents/work/备课/爬虫/图片/tesseracttest1.jpg"

file\_path = "/User/zdw/Documents/work/备课/爬虫/图片/test.jpg"

img = Image.open(file\_path)

**print**(pytesseract.image\_to\_string(img))