Python Crawler

`Python` `Crawler`

Author: Yunqiu Xu

- 我以前学习爬虫的一些笔记, 主要基于*Python核心编程*和静觅的爬虫教程

Chapter 1 使用`urllib/urllib2`

1.1 构造一个最基本的爬取网页的代码

```
1.  import urllib,urllib2
2.  response = urllib2.urlopen("http://www.baidu.com")
3.  print response.read()
```

- `urlopen(url, data, timeout)`:
  - `data`: 访问url时要传送的数据,默认为`None`,1.3节则将用户名和密码作为该参数传输
  - `timeout`:设定访问超时时间，默认`socket._GLOBAL_DEFAULT_TIMEOUT`

1.2 构造Request:

- 爬取过程: 构造request -- 服务器响应request -- 得到response

```
1.  request=urllib2.Request("http://www.baidu.com")
2.  response=urllib2.urlopen(request)
3.  print response.read()
```

1.3 数据传输方式: POST&GET

- 动态网页需要动态传送参数给它（如需要输入用户名密码），该网页做出对应的响应
- GET 数据传输：直接以链接形式访问，连接中包含所有参数(不大安全，但自己可以看到提交了什么内容)

```
#Get 数据传输
values={}
values['username']="username"
values['password']="password"
data=urllib.urlencode(values) #为dict编码，将编码后的dict作为data参数传输
url="url"
geturl=url+"?"+data #url添加用户名密码等参数
request=urllib2.Request(geturl)
response=urllib2.urlopen(request)
print response.read()
```

- POST数据传输：不会在网址上显示所有的参数，查看提交的内容不大方便

```
#Post 数据传输
values={}
values['username']="username"
values['password']="password"
data=urllib.urlencode(values)
url="url"
request=urllib2.Request(url,data) #只有这里和Get不同
response=urllib2.urlopen(request)
print response.read()
```

Chapter 2 `urllib/urllib2`的更多用法

2.1 设置headers: 模拟浏览器工作

▼ **Request Headers** view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
Cookie: laravel_session=eyJpdiI6Imw2c29PclB6cWlYK3NUTG1vbXdNS3c9PSIsInZhbHVlIjoiSWNTZEtYTEZtVW84bHZNUUFCb3F5RCtuaDhHSklzXC9WSG82ZDRoWHdMTGZuWk5ZcTJobmhkR21WTXhHMjFkOXRicTBKVlh5OFJ4WjREblNFN2NFV0NRPT0iLCJtYWMiOiJhN2FhMTcyYzA3MzVjYjRkZWM2YzM0OTkyNDE0NzkwNWY0MDAzMDA2ODllOGQyYTRkYWZhNzRkYTQ2MjgzZTUzIn0%3D; stat_fromWebUrl=; stat_ssid=1474412240793; stat_uuid=1474290791895550452272; stat_isNew=1; _ga=GA1.2.1053141496.1474290342; gr_user_id=fcb6fe2f-9296-4ea1-b934-0fb098fb1e10; Hm_lvt_1320421bf6332884ce38374c23776351=1474290795; Hm_lpvt_1320421bf6332884ce38374c23776351=1474290795; QINGCLOUDELB=b2396366bd639e01fbd88e4a829ce868d562b81daa2a899c46d3a1e304c7eb2b|V9/ke|V9/kY
Host: wiki.jikexueyuan.com
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.84 Safari/537.36

- Resquest Headers 中包含许多信息，譬如agent代表请求身份，没有写入的话服务器可能会不响应
- 在原有的抓取程序中添加headers, 并将user_agent加入这个dict用来让这个程序模拟浏览器，从而让服务器响应

```
...
user_agent = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.84 Safari/537.36"
headers={'User-Agent':user_agent}
data=urllib.urlencode(data)
request=urllib2.Request(url,data,headers=headers)
response=urllib2.urlopen(request)
page=response.read()
```

- 反盗链机制: 服务器识别header中的referer/host是不是其本身
  - 解决措施: 在header中添加referer

```
headers['referer']="http://www.zhihu.com/"
```

2.2 设置代理

- 网站会检测某段时间某ip的访问次数，若访问次数过多会禁止访问
  - 解决措施: 每隔一段时间更换代理

```
import urllib2
enable_proxy = True
proxy_handler = urllib2.ProxyHandler({"http" : 'http://some-proxy.com:8080'})
null_proxy_handler = urllib2.ProxyHandler({})
if enable_proxy:
    opener = urllib2.build_opener(proxy_handler)
else:
    opener = urllib2.build_opener(null_proxy_handler)
urllib2.install_opener(opener)
```

2.3 设置超时

- 停止访问一些反应过慢的页面
- 注意如果已经写明了data，直接传入超时时间即可,如10；反之，若data参数为空，需要特

别注明timeout=10

```
import urllib2
response = urllib2.urlopen('http://www.baidu.com',data, 10)
```

Chapter 3 异常处理

- URLError
  - 产生原因: 网络未连接；连接不到特定服务器；服务器不存在
  - 捕获异常

```
import urllib2

request = urllib2.Request('http://www.xxxxx.com')
try:
    urllib2.urlopen(request)
except urllib2.URLError, e:
    print e.reason
```

- HTTPError:
  - urlopen发送请求时, 服务器响应请求并返回response, 如果存在不能处理的内容则会产生HTTPError
  - 注意HTTPError是URLError的一个子类, 子类异常优先被捕获,捕获不到子类异常才会开始捕获父类

```
import urllib2

req = urllib2.Request('http://blog.csdn.net/cqcre')
try:
    urllib2.urlopen(req)
except urllib2.HTTPError, e:
    print e.code
except urllib2.URLError, e:
    print e.reason
else:
    print "OK"
```

Chapter 4 设置Cookie

- Cookie：某些网站为了辨别用户身份，进行session跟踪而储存在用户本地终端上的数据
- Opener：例如urlopen是一个特殊的opener,但我们使用Cookie时需要创建更一般的Opener
- Cookielib: 提供可存储cookie的对象以便与urllib2模块配合使用访问网络资源
  - 本模块内的CookieJar对象：捕获cookie并在后续连接请求时重新发送（如模拟登陆）
- 获取cookie并保存到变量:

```
import urllib2
import cookielib
#声明一个CookieJar对象实例来保存cookie
my_cookie = cookielib.CookieJar()

#利用urllib2库的HTTPCookieProcessor对象来创建cookie处理器
handler=urllib2.HTTPCookieProcessor(my_cookie)

#通过handler来构建opener
opener = urllib2.build_opener(handler)

#此处的open方法同urllib2的urlopen方法，也可以传入request
response = opener.open('http://www.baidu.com')
for item in my_cookie:
    print 'Name = '+item.name
    print 'Value = '+item.value

>>>
Name: BAIDUID
Value: 413C05FF289E8797034968BFADA0B1B8:FG=1
Name: BIDUPSID
Value: 413C05FF289E8797034968BFADA0B1B8
Name: H_PS_PSSID
Value: 1461_18240_21123_21193_21161_20928
Name: PSTM
Value: 1474296124
Name: BDSVRTM
Value: 0
Name: BD_HOME
Value: 0
```

- 获取cookie并保存到文件

```
import cookielib
import urllib2

#设置保存cookie的文件，同级目录下的cookie.txt
filename = 'cookie.txt'
#声明一个MozillaCookieJar对象实例来保存cookie，之后写入文件
cookie = cookielib.MozillaCookieJar(filename)
#利用urllib2库的HTTPCookieProcessor对象来创建cookie处理器
handler = urllib2.HTTPCookieProcessor(cookie)
#通过handler来构建opener
opener = urllib2.build_opener(handler)
#创建一个请求，原理同urllib2的urlopen
response = opener.open("http://www.baidu.com")

#保存cookie到文件
cookie.save(ignore_discard=True, ignore_expires=True)
# ignore_discard: 即使cookies将被丢弃也将它保存下来
# ignore_expire: 如果在该文件中cookies已经存在，则覆盖原文件写入
```

- 从文件中获取被保存的cookie进行访问

```
import cookielib
import urllib2

#创建MozillaCookieJar实例对象
cookie = cookielib.MozillaCookieJar()
#从文件中读取cookie内容到变量
cookie.load('cookie.txt', ignore_discard=True, ignore_expires=True)
#创建请求的request
req = urllib2.Request("http://www.baidu.com")
#利用urllib2的build_opener方法创建一个opener
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
response = opener.open(req)
print response.read()
```

- 使用cookie进行模拟登录

```
import urllib
import urllib2
import cookielib

#声明一个MozillaCookieJar对象实例来保存cookie，之后写入文件
filename = 'cookie.txt'
```

```
cookie = cookielib.MozillaCookieJar(filename)
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
userdata = urllib.urlencode({
            'username':'Yunqiu Xu',
            'pwd':'12345678'
        })

#登录教务系统的URL
urlname = 'urlname'
#模拟登录，并把cookie保存到变量
result = opener.open(loginUrl,userdata)
#保存cookie到cookie.txt中
cookie.save(ignore_discard=True, ignore_expires=True)

#通过得到的cookie，访问新网址
urlname2 = 'urlname2'
result = opener.open(urlname2)
print result.read()
```

Chapter 5 正则表达式

- 见*Python Notes Chapter 13*

Chapter 6 `Re` module的使用

- 见*Python Notes Chapter 13*
- 爬虫正则匹配基本框架

```
pattern = re.compile("your regular expression here",re.S)
items = re.findall(pattern,content)
for item in items:
    print item[0],item[1],item[2],item[3],item[4]
```

Chapter 7 一个简单的爬虫: 糗事百科段子抓取

- 抓取指定页数的段子(作者, 内容, 评分),排除其中有图片/视频的段子
- 源代码可见我的github

```python
#!/usr/bin/env python
#coding: utf-8

import urllib
import urllib2
import re

#获取指定页码的内容
def getPageFrom(url,current_page):
    referer = "http://www.qiushibaike.com/"
    user_agent = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)    Chrome/51.0.2704.84 Safari/537.36"
    headers = {'User-Agent': user_agent, 'referer': referer}
    new_url=url+'/page/'+str(current_page)
    request = urllib2.Request(new_url, headers=headers)
    response = urllib2.urlopen(request)
    content = response.read()
    return content

#正则匹配：得到某一页的段子，返回段子的内容，作者和评分数，排除图片及视频
#得到作者
def get_author(content):
    pattern_author = re.compile('<h2>(.*?)</h2>',re.S)
    items_author = re.findall(pattern_author,content)
    for item in items_author:
        print item
#得到段子
def get_content(content):
    pattern_content = re.compile('<div class="content">[.\n]+<span>(.*?)</span>',re.S)
    items_content = re.findall(pattern_content,content)
    for item in items_content:
        print item
#得到评分数
def get_vote(content):
    pattern_vote = re.compile('<i class="number">(.*?)</i>',re.S)
    items_vote=re.findall(pattern_vote,content)
    for item in items_vote:
        print item
#综合以上结果，并排除有图片/视频的内容
def get_text_items(content):
    pattern_total = re.compile(
        '<h2>(.*?)</h2>.*?<div class="content">[.\n]+<span>(.*?)</span>(.*?)<i class="number">(.*?)</i>', re.S)
    items_total = re.findall(pattern_total, content)
```

```
        for item in items_total:
            if 'img' in item[2]:
                print "********************"
                print "带图片的段子, 跳过!"
                continue
            else:
                print "********************"
                print "作者: " + item[0]
                print "内容: " + item[1]
                print "评分: " + item[3]

    #整合之前的函数
    #输入起始页码和结束页码,按页输出段子
    def start(start_page,end_page):
        url = "http://www.qiushibaike.com/8hr"
        current_page=start_page
        while current_page<=end_page:
            print "现在开始打印第{}页的内容".format(current_page)
            content=getPageFrom(url,current_page)
            get_text_items(content)
            current_page+=1
            con=raw_input("继续吗?(Y/N)")
            if con!='Y':
                print "停止打印"
                exit()
        print "完成全部输出!"

    if __name__=="__main__":
        print "糗事百科爬虫"
        start_page=int(raw_input("输入起始页码: "))
        end_page=int(raw_input("输入结束页码: "))
        start(start_page, end_page)
```

Chapter 8 使用`BeautifulSoup`

- Beautiful Soup提供一些简单的、python式的函数用来处理导航、搜索、修改分析树等功能
- Beautiful Soup自动将输入文档转换为Unicode编码，输出文档转换为utf-8编码, 不需要考虑编码方式，除非文档没有指定一个编码方式

8.1 创建BS对象

```
import urllib
import urllib2
from bs4 import BeautifulSoup

#获取response的过程跟前面一样
request = urllib2.Request(new_url, headers=headers)
my_html=urllib2.urlopen(request)
#将response转化为BS对象
my_soup=BeautifulSoup(my_html)
print my_soup.prettify() #格式化输出
```

8.2 对象种类

8.2.1 Tag: HTML中的标签

```
print my_soup.title
>>> <title>糗事百科 - 超搞笑的原创糗事笑话分享社区</title>
print my_soup.a
>>> <a href="/"><h1>糗事百科</h1></a>
```

- Tag的两个重要属性: names, attrs
  - names为对象名, BS对象比较特殊输出[document],对于其他内部标签输出标签名
  - attrs为属性,得到字典
  - 属性可以进行赋值替换或删除

```
print my_soup.name
>>> [document]

print my_soup.head.name
>>> head

print my_soup.title.name
>>> title

print my_soup.a.attrs
>>> {'href': '/'}

print my_soup.div.attrs
>>> {'id': 'header', 'class': ['head']}
```

```
15.
16.  del my_soup.div['class']
```

8.2.2 NavigableString: 可以遍历的字符串

- 得到标签内的内容: 省略了正则表达式

```
1.  print my_soup.title.string
2.  >>> 糗事百科 - 超搞笑的原创糗事笑话分享社区
3.
4.  #若使用.strings则产生生成器对象
```

8.2.3 BeautifulSoup: 文档全部内容, 可以看做特殊的tag对象

```
1.  print my_soup.attrs
2.  >>> {}
3.
4.  print my_soup.string
5.  >>> None
6.
7.  print type(my_soup)
8.  >>> <class 'bs4.BeautifulSoup'>
```

8.2.4 Comment: 特殊的NavigableString

- 建议在使用前先判断类型是否为Comment

```
1.  if type(soup.a.string)==bs4.element.Comment:
2.      print soup.a.string
```

8.3 遍历文档树

- 子结点, 结点内容, 父结点, 兄弟结点, 前后结点

8.3.1 子结点:

- 直接子结点: `.contents, .children`
  - `.contents`输出方式为列表
  - `.children`输出方式为生成器
  - 使用for循环的话这两个的结果是一样的, 但`.children`不能直接输出列表

```
1.  for item in my_soup.head.contents:
```

```
    print item

for item in my_soup.head.children:
    print item
```

- 子孙结点: `.decendants`, 对所有tag的子孙结点进行递归循环

```
for item in my_soup.descendants:
    print item
```

8.3.2 结点内容:

- `.string`
  - 如果一个标签里不存在标签,则返回当前标签中内容
  - 如果一个标签里只存在一个标签,则返回最里面标签的内容
  - 如果一个标签里存在多个标签,返回None

```
print my_soup.head.string
```

- 多个内容: `.strings / .stripped_strings`
  - `.strings`遍历获取多个内容
  - `.stripped_strings`去除空行

```
for item in my_soup.strings:
    print item

for item in my_soup.stripped.strings:
    print item
```

8.3.3 父结点:

- `.parent`

```
print my_soup.div.parent.name
>>> body
```

- 递归返回全部父结点: `.parents`

8.3.4 兄弟结点:

- `.next_sibling / .previous_sibling`

- 全部兄弟结点: `.next_siblings / .previous_siblings`

8.3.4 前后结点:

- 与兄弟结点不同: 前后结点针对所有结点, 不分层次
- `.next_element / .previous_element`
- `.next_elements / .previous_elements`

8.4 搜索文档树

8.4.1 `find_all( name , attrs , recursive , text , **kwargs )`

- 遍历当前tag的所有子结点
- `name`: 查找所有名字为name的tag

```
#传入字符串
my_soup.find_all('a')
my_soup.find_all('div')

#传入正则表达式
pattern=re.compile('regular expression')
my_soup.find_all(pattern)

#传入列表：只要匹配列表中某一个函数就会返回
my_soup.find_all(['a','div'])

#传入True：返回所有标签，但不会返回字符串结点
my_soup.find_all(True)
my_soup.find_all() #等价

#传入方法：定义一个方法接受当前元素，返回True则表示匹配成功
#有些类似filter函数
def has_div(item):
    return 'div' in item
my_soup.find_all(has_div)
```

- 关键字参数

```
my_soup.find_all("span",class_="sister")
my_soup.find_all("span",id="Yunqiu Xu")
```

- attrs参数: 一些包含特殊属性的tag无法搜索, 可通过attrs参数转换为字典参数

```
#HTML5中的data-*属性不能直接搜索
data_soup = BeautifulSoup('<div data-foo="value">foo!</div>')
data_soup.find_all(data-foo="value")
>>> SyntaxError: keyword can't be an expression

#使用attrs转换
data_soup.find_all(attrs={"data-foo": "value"})
>>> [<div data-foo="value">foo!</div>]
```

- text参数: 搜索文档中的字符串, 参数可选值同name

```
my_soup.find_all(text=True)
my_soup.find_all(text='sb')
```

- limit参数: 限制返回结果数量

```
my_soup.findall(limit=10)
```

- recursive参数:
  - 默认为True: 遍历全部子孙结点
  - False: 仅仅返回直接子结点

```
len(my_soup.find_all('div'))
>>> 215
len(my_soup.find_all('div',recursive=False))
>>> 0
```

8.4.2 其他搜索函数

```
find() #参数同find_all(), 但是仅仅返回第一个匹配的结果

#返回当前匹配结点的父辈结点
find_parent()
find_parents()

#返回当前匹配结点的兄弟结点
find_next_sibling()
find_next_siblings()
find_previous_sibling()
find_previous_siblings()
```

```
12.
13.  #返回当前匹配结点的前后结点
14.  find_next()
15.  find_all_next()
16.  find_previous()
17.  find_all_previous()
```

8.5 CSS选择器 `soup.select()`

- 标签名查找

```
1.  my_soup.select('title')
```

- 同理, 还有类名查找/id名查找, 或者多种查找方式的组合查找(使用空格分割)

```
1.  my_soup.select('head title'
```

- 属性查找: 属性与标签属于同一结点, 中间不可加空格

```
1.  soup.select('a[href="http://example.com/elsie"]')
```

Chapter 9 `requests` module

- urllib2提供了爬虫所需的大部分功能, 但其过于复杂, requests库用于简化这些功能

9.1 发出请求

```
1.  import requests
2.
3.  url='http://www.qiushibaike.com/'
4.  r1=requests.get(url)
5.  r2=requests.post(url,data={"username":"password"})
6.  r3=requests.put(url,data={"username":"password"})
7.  r4=requests.head(url)
8.
9.  #传递参数
10. data={"username":"password"}
```

```
r5=requests.get(url,params=data)

#设置超时
r6=requests.get(url,timeout=0.001)
```

9.2 响应内容

```
r.text #相当于之前的response.read()
r.encoding #获取网页编码
>>> 'UTF-8'
r.content #二进制响应
r.json() #json响应

#获取响应状态
r.status_code
>>> 405

#响应headers
r.headers
```

9.3 模拟登陆

```
# 设置headers
referer = "http://www.qiushibaike.com/"
user_agent = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)    Chrome/51.0.2704.84 Safari/537.36"
headers = {'User-Agent': user_agent, 'referer': referer}
r=requests.get(url,headers=headers)

# 传递一个dict到data参数处
my_data={'key1':'val1','key2':'val2'}
r=requests.post(url,data=my_data)
# 也可以传递到json参数处
r=requests.post(url,json=my_data)
# 从文件中提取数据传递到file参数
my_file={'file':open(path,'rb')}
r=requests.post(url,file=my_file)
```

9.4 获取并传递cookies

- 获取cookies

```
r=requests.get(url,headers=headers)
my_cookies=r.cookies
```

- 将自己的cookies发送到服务器

```
my_cookies=={'cookies':'xxx'}
r=requests.get(url,cookies=my_cookies)
```

9.5 查询历史及重定向

```
r.history #查询历史
r=requests.get(url,allow_redirects=False) #禁止重定向
```

9.6 高级用法

- 可参考官方文档
- To be continued