Python爬虫基础知识

2017年2月15日　　14:15

一、爬取数据步骤

二、数据抓取、下载

抓取使用pyton库

- 自带库：urllib、urllib2
- 第三方： requests、httplib2 …

1. 最基本的抓取

- Requests：

```
import requests
response = requests.get(url)
content = requests.get(url).content
print "response headers:", response.headers
print "content:", content
```

- Urllib2：

```
import urllib2
response = urllib2.urlopen(url)
content = urllib2.urlopen(url).read()
print "response headers:", response.headers
print "content:", content
```

- Httplib2：

```
import httplib2
http = httplib2.Http()
response_headers, content = http.request(url, 'GET')
print "response headers:", response_headers
print "content:", content
```

2、GET请求与POST请求

GET一般用于获取/查询资源信息；而POST一般用于更新资源信息，通常用于我们需要向服务器提交表单的情况。

1) GET请求一般用于我们向服务器查询的过程
对于带有查询字段的url，get请求一般会将来请求的数据附在url之后，以?分割url和传输数据，多个参数用&连接。

- Requests：data为dict，json

```
import requests
data = {'data1':'XXXXX', 'data2':'XXXXX'}
response = requests.get(url=url, params=data)
```

- Urllib2：data为string

```
import urllib, urllib2
data = urllib.urlencode(data)
full_url = url+'?'+data
response = urllib2.urlopen(full_url)
```

2) POST请求

使用表单登陆,这种情况属于post请求，即先向服务器发送表单数据，服务器再将返回的cookie存入本地。

- Requests：data为dict，json

```
import requests
data = {'data1':'XXXXX', 'data2':'XXXXX'}
response = requests.post(url=url, data=data)
```

- Urllib2：data为string

```
import urllib, urllib2
data = urllib.urlencode(data)
req = urllib2.Request(url=url, data=data)
response = urllib2.urlopen(req)
```

3、使用Cookie登陆

使用cookie登陆，服务器会认为你是一个已登陆的用户，所以就会返回给你一个已登陆的内容。因此，需要验证码的情况可以使用带验证码登陆的cookie解决。

```
import requests
requests_session = requests.session()
response = requests_session.post(url=url_login, data=data)
```

若存在验证码，此时采用response = requests_session.post(url=url_login, data=data)是不行的，做法应该如下：

```
response_captcha = requests_session.get(url=url_login, cookies=cookies)
response1 = requests.get(url_login) # 未登陆
response2 = requests_session.get(url_login) # 已登陆，因为之前拿到了Response Cookie!
response3 = requests_session.get(url_results) # 已登陆，因为之前拿到了Response Cookie!
```

4、对于反爬虫机制的处理

3.1 使用代理

适用情况：限制IP地址情况，也可解决由于“频繁点击”而需要输入验证码登陆的情况。

这种情况最好的办法就是维护一个代理IP池，网上有很多免费的代理IP，良莠不齐，可以通过筛选找到能用的。对于“频繁点击”的情况，我们还可以通过限制爬虫访问网站的频率来避免被网站禁掉。

```
proxies = {'http':'http://XX.XX.XX.XX:XXXX'}
Requests：
    import requests
    response = requests.get(url=url, proxies=proxies)
Urllib2：
    import urllib2
    proxy_support = urllib2.ProxyHandler(proxies)
    opener = urllib2.build_opener(proxy_support, urllib2.HTTPHandler)
    urllib2.install_opener(opener) # 安装opener，此后调用urlopen()时都会使用安装过的opener对象
    response = urllib2.urlopen(url)
```

3.2 时间设置

适用情况：限制频率情况。

Requests，Urllib2都可以使用time库的sleep()函数：

```
import time
time.sleep(1)
```

3.3 伪装成浏览器，或者反“反盗链”

有些网站会检查你是不是真的浏览器访问，还是机器自动访问的。这种情况，加上User-Agent，表明你是浏览器访问即可。有时还会检查是否带Referer信息还会检查你的Referer是否合法，一般再加上Referer。

```
headers = {'User-Agent':'XXXXX'} # 伪装成浏览器访问，适用于拒绝爬虫的网站
headers = {'Referer':'XXXXX'}
headers = {'User-Agent':'XXXXX', 'Referer':'XXXXX'}
```

```
Requests：
    response = requests.get(url=url, headers=headers)
Urllib2：
    import urllib, urllib2
    req = urllib2.Request(url=url, headers=headers)
    response = urllib2.urlopen(req)
```

4. 对于断线重连

```
def multi_session(session, *arg):
    retryTimes = 20
    while retryTimes>0:
        try:
            return session.post(*arg)
        except:
            print '.',
            retryTimes -= 1
```

或者

```
def multi_open(opener, *arg):
    retryTimes = 20
    while retryTimes>0:
        try:
            return opener.open(*arg)
        except:
            print '.',
            retryTimes -= 1
```

这样我们就可以使用multi_session或multi_open对爬虫抓取的session或opener进行保持。

5. 对于Ajax请求的处理
对于“加载更多”情况，使用Ajax来传输很多数据。

它的工作原理是：从网页的url加载网页的源代码之后，会在浏览器里执行JavaScript程序。这些程序会加载更多的内容，“填充”到网页里。这就是为什么如果你直接去爬网页本身的url，你会找不到页面的实际内容。

这里，若使用Google Chrome分析”请求“对应的链接(方法：右键→审查元素→Network→清空，点击”加载更多“，出现对应的GET链接寻找Type为text/html的，点击，查看get参数或者复制Request URL)，循环过程。

如果“请求”之前有页面，依据上一步的网址进行分析推导第1页。以此类推，抓取抓Ajax地址的数据。
对返回的json格式数据(str)进行正则匹配。json格式数据中，需从’\uxxxx’形式的unicode_escape编码转换成u’\uxxxx’的unicode编码。

三、网页分析

1、正则表达式re 模块

re.match函数

re.match 尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match()就返回none。

函数语法：

```
re.match(pattern, string, flags=0)
```

函数参数说明：

参数	描述
pattern	匹配的正则表达式
string	要匹配的字符串。
flags	标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。

匹配成功re.match方法返回一个匹配的对象，否则返回None。

re.search方法

re.search 扫描整个字符串并返回第一个成功的匹配。

函数语法：

```
re.search(pattern, string, flags=0)
```

函数参数说明：

参数	描述
pattern	匹配的正则表达式
string	要匹配的字符串。
flags	标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。

匹配成功re.search方法返回一个匹配的对象，否则返回None。

实例 1：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import re
print(re.search('www', 'www.runoob.com').span())  # 在起始位置匹配
print(re.search('com', 'www.runoob.com').span())         # 不在起始位置匹配
```

以上实例运行输出结果为：

```
(0, 3)
(11, 14)
```

我们可以使用group(num) 或 groups() 匹配对象函数来获取匹配表达式。

匹配对象方法	描述
group(num=0)	匹配的整个表达式的字符串，group() 可以一次输入多个组号，在这种情况下它将返回一个包含那些组所对应值的元组。
groups()	返回一个包含所有小组字符串的元组，从 1 到 所含的小组号。

实例 2：

```
#!/usr/bin/python
import re

line = "Cats are smarter than dogs";

searchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)

if searchObj:
   print "searchObj.group() : ", searchObj.group()
   print "searchObj.group(1) : ", searchObj.group(1)
   print "searchObj.group(2) : ", searchObj.group(2)
else:
   print "Nothing found!!"
```

以上实例执行结果如下：

```
searchObj.group() :  Cats are smarter than dogs
searchObj.group(1) :  Cats
searchObj.group(2) :  smarter
```

re.match与re.search的区别

re.match只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回None；而re.search匹配整个字符串，直到找到一个匹配。

检索和替换

Python 的 re 模块提供了re.sub用于替换字符串中的匹配项。

语法：

```
re.sub(pattern, repl, string, count=0, flags=0)
```

参数：

- pattern：正则中的模式字符串。
- repl：替换的字符串，也可为一个函数。
- string：要被查找替换的原始字符串。
- count：模式匹配后替换的最大次数，默认 0 表示替换所有的匹配。

正则表达式模式

模式	描述
^	匹配字符串的开头
$	匹配字符串的末尾。
.	匹配任意字符，除了换行符，当re.DOTALL标记被指定时，则可以匹配包括换行符的任意字符。
[...]	用来表示一组字符,单独列出：[amk] 匹配 'a'，'m'或'k'
[^...]	不在[]中的字符：[^abc] 匹配除了a,b,c之外的字符。
re*	匹配0个或多个的表达式。
re+	匹配1个或多个的表达式。

re?	匹配0个或1个由前面的正则表达式定义的片段，非贪婪方式
re{ n}	
re{ n,}	精确匹配n个前面表达式。
re{ n, m}	匹配 n 到 m 次由前面的正则表达式定义的片段，贪婪方式
a| b	匹配a或b
(re)	G匹配括号内的表达式，也表示一个组
(?imx)	正则表达式包含三种可选标志：i, m, 或 x 。只影响括号中的区域。
(?-imx)	正则表达式关闭 i, m, 或 x 可选标志。只影响括号中的区域。
(?: re)	类似 (...), 但是不表示一个组
(?imx: re)	在括号中使用i, m, 或 x 可选标志
(?-imx: re)	在括号中不使用i, m, 或 x 可选标志
(?#...)	注释.
(?= re)	前向肯定界定符。如果所含正则表达式，以 ... 表示，在当前位置成功匹配时成功，否则失败。但一旦所含表达式已经尝试，匹配引擎根本没有提高；模式的剩余部分还要尝试界定符的右边。
(?! re)	前向否定界定符。与肯定界定符相反；当所含表达式不能在字符串当前位置匹配时成功
(?> re)	匹配的独立模式，省去回溯。
\w	匹配字母数字及下划线
\W	匹配非字母数字及下划线
\s	匹配任意空白字符，等价于 [\t\n\r\f].
\S	匹配任意非空字符
\d	匹配任意数字，等价于 [0-9].
\D	匹配任意非数字
\A	匹配字符串开始
\Z	匹配字符串结束，如果是存在换行，只匹配到换行前的结束字符串。c
\z	匹配字符串结束
\G	匹配最后匹配完成的位置。
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。
\B	匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。
\n, \t, 等.	匹配一个换行符。匹配一个制表符。等
\1...\9	匹配第n个分组的子表达式。
\10	匹配第n个分组的子表达式，如果它经匹配。否则指的是八进制字符码的表达式。

2、LXML包XPath

lxml是**Python**的一个html/xml解析并建立dom的库，lxml的特点是功能强大，性能也不错，xml包含了ElementTree，html5lib ， beautfulsoup 等库，但是lxml也有自己相对应的库，所以，导致lxml比较复杂，初次使用者很难了解其关系。

使用lxml前注意事项：先确保html经过了utf-8解码，即code = html.decode('utf-8', 'ignore')，否则会出现解析出错情况。因为中文被编码成utf-8之后变成 '/u2541' 之类的形式，lxml一遇到 "/"就会认为其标签结束。

1）解析url，得到html

```
1 import codecs
2 from lxml import etree
3 f=codecs.open("ceshi.html","r","utf-8")
4 content=f.read()
5 f.close()
6 tree=etree.HTML(content)
```

etree提供了HTML这个解析函数，现在我们可以直接对HTML使用xpath了

2)Xpath 选取节点

实例：

```
>>> from lxml import etree
>>> xml = """<?xml version="1.0" encoding="utf8"?>
<bookstore>
        <book>
                <title lang="eng">Harry Potter</title>
                <price>29.99</price>
        </book>
        <book>
                <title lang="eng">Learning XML</title>
                <price>39.95</price>
        </book>
</bookstore>"""
```

选取节点

以下为基本路径的表达方式，记住 XPath 的路径表达式都是基于某个节点之上的，例如最初的当前节点一般是根节点，这与 Linux 下路径切换原理是一样的。

表达式	描述
nodename	选取已匹配节点下名为 nodename 的子元素节点。
/	如果以 / 开头，表示从根节点作为选取起点。
//	在已匹配节点后代中选取节点，不考虑目标节点的位置。
.	选取当前节点。
..	选取当前节点的父元素节点。
@	选取属性。

```
# 得到根节点
>>> root = etree.fromstring(xml)
>>> print root
<Element bookstore at 0x2c9cc88>

# 选取所有book子元素
>>> root.xpath('book')
[<Element book at 0x2d88878>, <Element book at 0x2d888c8>]

# 选取根节点bookstore
>>> root.xpath('/bookstore')
[<Element bookstore at 0x2c9cc88>]

# 选取所有book子元素的title子元素
>>> root.xpath('book/title')
[<Element title at 0x2d88878>, <Element title at 0x2d888c8>]
```

预判（Predicates）

预判是用来查找某个特定的节点或者符合某种条件的节点，预判表达式位于方括号中。

```
# 选取bookstore的第一个book子元素
>>> root.xpath('/bookstore/book[1]')
[<Element book at 0x2ca20a8>]
```

通配符

通配符	描述
*	匹配任何元素。
@*	匹配任何属性。
node()	匹配任何类型的节点。

```
# 选取 bookstore 所有子元素
>>> root.xpath('/bookstore/*')
[<Element book at 0x2d888c8>, <Element book at 0x2ca20a8>]

# 选取根节点的所有后代元素
>>> root.xpath('//*')
[<Element bookstore at 0x2c9cc88>, <Element book at 0x2d888c8>, <Element title at 0x2d888
738>, <Element price at 0x2d88878>, <Element book at 0x2ca20a8>, <Element title at 0x2d8
8940>, <Element price at 0x2d88a08>]
```

或条件选取

使用 "|" 运算符，你可以选取符合“或”条件的若干路径。

```
# 选取所有book的title元素或者price元素
>>> root.xpath('//book/title|//book/price')
[<Element title at 0x2d88738>, <Element price at 0x2d88940>, <Element title at 0x2ca20a8
>, <Element price at 0x2d88a08>]
```

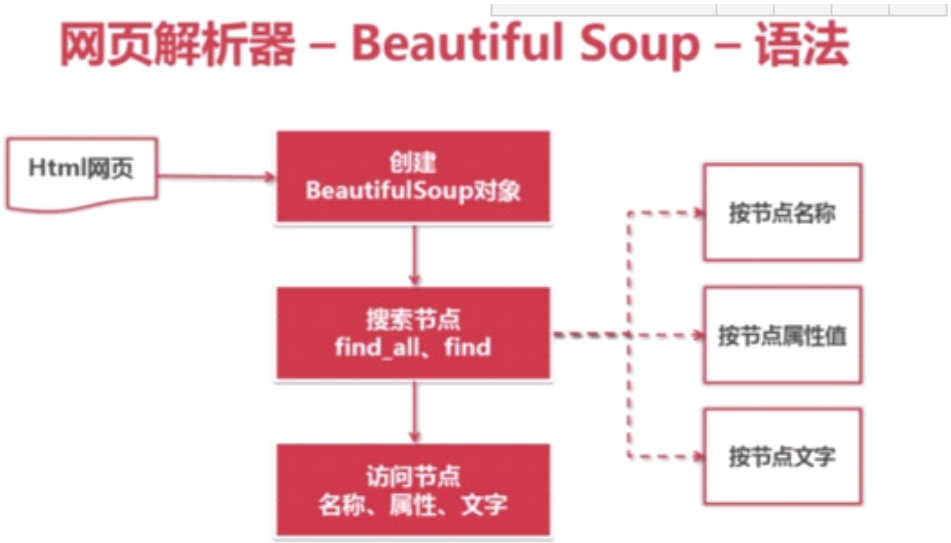
3、Beautiful Soup解析网页

Beautiful Soup ： 是Python第三方库，用于从HTML或XML中提取数据,官网：
https://www.crummy.com/software/BeautifulSoup/bs4/doc/

安装BeautifulSoup，
pip install beautifulsoup4 （如果没有安装pip，先安装pip）

测试是否安装成功
```
import bs4
print bs4
```
如安装成功，结果如下
<module 'bs4' from '/usr/lib/python2.7/dist-packages/bs4/__init__.pyc'>

Beautiful Soup使用方法：



1）创建BeautifulSoup对象

```
from bs4 import BeautifulSoup

#根据HTML网页字符串创建BeautifulSoup对象
soup = BeautifulSoup(
                     html_doc,              # HTML文档字符串
                     'html.parser'          # HTML解析器
                     from_encoding='utf8'   # HTML文档的编码
                     )
```

2）搜索节点（find_all, find）

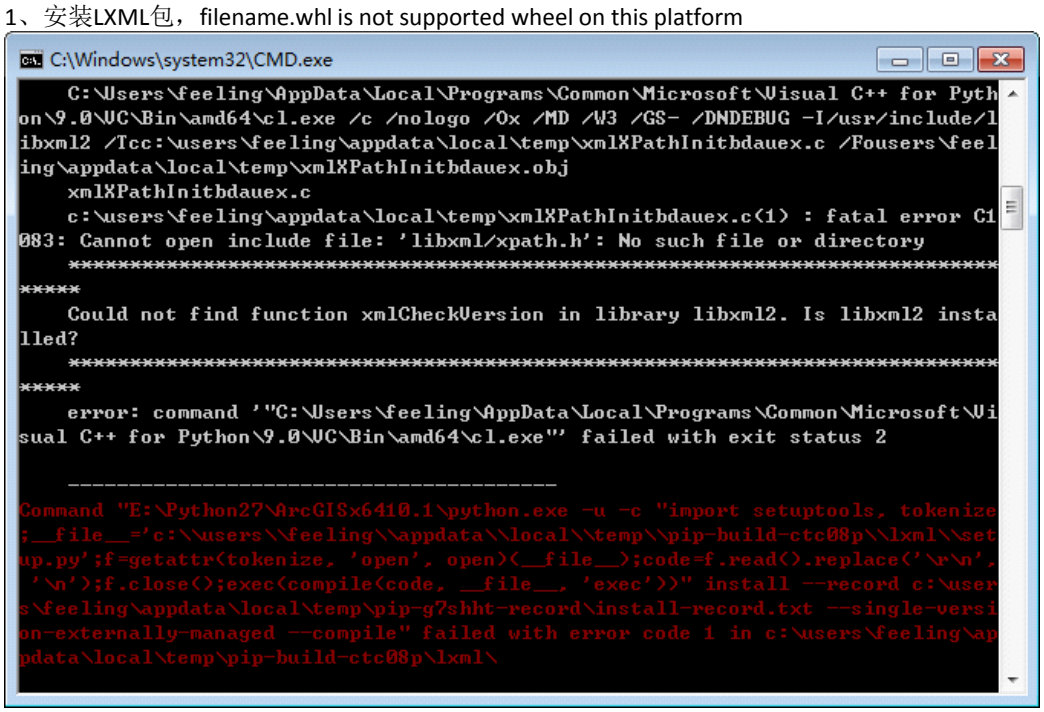
```
#方法: find_all(name, attrs, string)

# 查找所有标签为a的节点
soup.find_all('a')

# 查找所有标签为a，链接符合/view/123.html形式的节点
soup.find_all('a', href='/view/123.htm')

# 传入正则表达式来匹配对应的内容
soup.find_all('a', href=re.compile(r'/view/\d+\.htm'))

# 查找所有标签为idv, class(下面的代码有加下划线，是因为python的关键字有class,防冲突)为 abc， 文字为Python的节点
soup.find_all('div', class_='abc', string='Python')
```

3）访问节点信息：

```
# 得到节点：<a href='1.html'>Python</a>

# 获取查找到的节点的标签名称
node.name

# 获取查找到的a节点的href属性
node['href']

# 获取查找到的a节点的链接文字
node.get_text()
```

四、遇到问题

1、安装LXML包，filename.whl is not supported wheel on this platform



解决方法：

1）安装wheel，命令行运行：
pip install wheel
2）在这里下载对应的.whl文件，注意别改文件名！
安装的不是对应python版本的库，下载的库名中cp27代表python2.7,其它同理
3） 进入.whl所在的文件夹，执行命令即可完成安装
pip install 带后缀的完整文件名

4）如果以上还不行，注意安装文件名称 方法：
在shell中输入import pip; print(pip.pep425tags.get_supported())可以获取到pip支持的文件名还有版本，我这里如下：

```
>>> import pip
>>> print (pip.pep425tags.get_supported())
[('cp27', 'cp27m', 'win_amd64'), ('cp27', 'none', 'win_amd64'), ('py2', 'none', 'win_amd64'), ('cp27', 'none', 'any'), ('cp2', 'none', 'any'), ('py27', 'none', 'any'), ('py2', 'none', 'any'), ('py26', 'none', 'any'), ('py25', 'none', 'any'), ('py24', 'none', 'any'), ('py23', 'none', 'any'), ('py22', 'none', 'any'), ('py21', 'none', 'any'), ('py20', 'none', 'any')]
```

如果安装文件名称不符合你机器支持的，则会安装不成功。

2、获取网页源码时，如果url中带了中文参数则出现以下错误：
UnicodeEncodeError: 'ascii' codec can't encode characters in position 36-37: ordinal not in range

解决方法：

因为中文在URL中进行了url quote处理的，例如：
http://***.com/newsearch/books/?query=你好
实际上的URL是：
http://***.com/newsearch/books/?query=%C4%E3%BA%C3

因此在将url传给urlopen之前，应该对url进行unquote
综上所述，对url先进行编码上的转换，然后再用unquote处理，就能得到可获取的url了。

url地址中含中文，要进行编码转换

```
def encodeurl(url):
   url=url.encode('utf-8')
   url=urllib2.unquote(url)
   return url
```

3、中文内容存csv文件时报以下错
UnicodeEncodeError: 'ascii' codec can't encode character u'\u826f' in position 0: ordinal not in range(128)

解决方法：

python处理utf8编码中文，及打印中文列表和字典

python处理utf8编码中文，及打印中文列表和字典
python处理utf8编码中文，需要在py文件的第一行加入：# -*- coding:utf-8 -*- 或者 #coding=utf-8

打印字符串时，使用print str.encode('utf8');

打印中文列表时，使用循环 for key in list：print key

打印中文字典时，可以使用循环，也可以使用json：

```
import json

print json.dumps(dict, encoding='UTF-8', ensure_ascii=False)
```