中國人民大學
RENMIN UNIVERSITY OF CHINA

第2编 Python数据分析

第4讲 爬虫

余力

buaayuli@ruc.edu.cn

中國人民大學
RENMIN UNIVERSITY OF CHINA

01. urllib

爬虫

- 爬虫，即网络爬虫，大家可以理解为在网络上爬行的一直蜘蛛，互联网就比作一张大网，而爬虫便是在这张网上爬来爬去的蜘蛛咯，如果它遇到资源，那么它就会抓取下来。
- URL 与源代码
- **学习内容：**
  - Urllib
  - 正则表达式
  - BeautifuleSoup

urllib

- urllib 是 Python 标准库中用于网络请求的库。该库有四个模块，分别是
  - **urllib.request**，
  - urllib.error，
  - urllib.parse，
  - urllib.robotparser

1. 简单的例子

```
import urllib.request
response = urllib.request.urlopen("http://www.baidu.com")
print response.read()
```

urlopen(url, data, timeout)

第一个参数url即为URL，

第二个参数data是访问URL时要传送的数据，

第三个timeout是设置超时时间。

2.构造Request

```
import urllib.request
request =
urllib.request.Request("http://www.sohu.com")
response = urllib.request.urlopen(request)
print response.read() .decode('utf-8')
```

```
import urllib2
response = urllib.request.urlopen("http://www.
sohu.com")
print response.read()
```

设置headers

可以在**headers**中设置**agent**

```
import urllib
url = 'http://www.server.com/login'
user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
headers = { 'User-Agent' : user_agent }
values = {'username' : 'cqc',  'password' : 'XXXX' }
data = urllib.urlencode(values)
request = urllib.request.Request(url, data, headers)
response = urllib.request.urlopen(request)
page = response.read()
```

中国人民大学
RENMIN UNIVERSITY OF CHINA

02. 正则表达式

例子

```
s ='<html><body><h1>hello world</h1></body></html>'

start_index = s.find('<h1>')

end_index = s.find('</h1>')

s[start_index+4 : end_index]

s ='<html><body><h1>hello world</h1><h1>good morning

world</h1></body></html>'
```

正则表达式是什么

- **正则表达**式，又称正规表示式、正规表示法、正规表达式、规则表达式、常规表示法（英语：**Regular Expression**，在代码中常简写为regex、regexp或RE）
- 正则表达式使用单个字符串来描述、匹配一系列匹配某个句法规则的字符串

```
import re
key = r"<html><body><h1>hello
world</h1></body></html>"
p = r"(?<=<h1>).+?(?=</h1>)"
pattern = re.compile (p) #我们在编译这段正则表达式
matcher = re.search (pattern,key) #搜索符合正则表达式的部分
print matcher.group(0)#打印出来
```

重要的符号

.字符在正则表达式代表着可以代表任何一个字符（包括它本身）

+的作用是将前面一个字符或一个子表达式重复一遍或者多遍。

*跟在其他符号后面表达可以匹配到它0次或多次

[]代表匹配里面的字符中的任意一个

[^]代表除了内部包含的字符以外都能匹配

```
key = r"afiouwehrfuichuxiuhong@hit.edu.cnaskdjhfiosueh"
p1 = r"chuxiuhong@hit\.edu\.cn"
pattern1 = re.compile(p1)

key = r"http://www.nsfbuhwe.com and
https://www.auhfisna.com"
p1 = r"https*://"#看那个星号 0次或多次
pattern1 = re.compile(p1)

key = r"lalala<hTml>hello</Html>heiheihei"
p1 = r"<[Hh][Tt][Mm][Ll]>.+?</[Hh][Tt][Mm][Ll]>"
pattern1 = re.compile(p1)

key = r"mat cat hat pat"
p1 = r"[^p]at"#这代表除了p以外都匹配
pattern1 = re.compile(p1)
```

常用表达式

正则表达式	代表的匹配字符
[0-9]	0123456789任意之一
[a-z]	小写字母任意之一
[A-Z]	大写字母任意之一
\d	等同于[0-9]
\D	等同于[^0-9]匹配非数字
\w	等同[a-z0-9A-Z_]匹配大小写字母、数字和下划线
\W	等同于[^a-z0-9A-Z_]等同于上一条取非

元字符	说明
.	代表任意字符
|	逻辑或操作符
[ ]	匹配内部的任一字符或子表达式
[^]	对字符集和取非
-	定义一个区间
\	对下一字符取非（通常是普通变特殊，特殊变普通）
*	匹配前面的字符或者子表达式**0次或多次**
*?	惰性匹配上一个
+	匹配前一个字符或子表达式**一次或多次**
+?	惰性匹配上一个
?	匹配前一个字符或子表达式0次或1次重复
{n}	匹配前一个字符或子表达式
{m,n}	匹配前一个字符或子表达式至少m次至多n次
{n,}	匹配前一个字符或者子表达式至少n次
{n,}?	前一个的惰性匹配
^	匹配字符串的开头
\A	匹配字符串开头
$	匹配字符串结束
[\b]	退格字符
\c	匹配一个控制字符
\d	匹配**任意数字**
\D	匹配**数字以外**的字符
\t	匹配制表符
\w	匹配任意**数字字母**下划线
\W	不匹配**数字字母**下划线

模块方法

- **re.match()** 总是从字符串“开头”去匹配，并返回匹配的字符串的match对象。所以当我用re.match()函数去匹配字符串非开头部分的字符串时，会返回NONE。
  - str1 = 'Hello World!'
  - print(**re.match**(r'e',str1))
  - 结果为:NONE
- **group() and groups()**
  - 以元组形式返回全部分组截获的字符串。
  - 相当于调用group(1,2,…last)
  - **>>>str1='Hello World!'**
  - **re.match(r'H',str1).group()**

```
import re
line = "This is the last one"
res = re.match( r'(.*) is (.*?) .*', line, re.M|re.I)
if res:
   print "res.group() : ", res.group()
   print "res.group(1) : ", res.group(1)
   print "res.group(2) : ", res.group(2)
   print "res.groups() : ", res.groups()
else:
   print "No match!!"
```

- re.M|re.I：这两参数表示**多行匹配**|**不区分大小写**，同时生效

- **re.search()**函数将对整个字符串进行搜索，并返回第一个匹配的字符串的match对象。
  - **str1 = 'Hello World!'**
  - **print(re.search(r'e',str1))**
  - 输出结果：<_sre.SRE_Match object; span=(1, 2), match='e'>

- re.search()和re.match()函数返回match对象包括分组时，group(0)返回【完整匹配】的字符串，group(1)及以上分别返回各分组字符串。groups()函数返回各分组组成的元组对象。

re.match与re.search的区别

- match()函数只检测RE是不是在string的开始位置匹配，
- search()会扫描整个string查找匹配；
- match()只有在0位置匹配成功的话才有返回，如果不是开始位置匹配成功的话，match()就返回none。
  - print(re.match('super', 'superstition').span()) 会返回(0, 5)
  - print(re.match('super', 'insuperable')) 则返回None
- search()会扫描整个字符串并返回第一个成功的匹配
  - print(re.search('super', 'superstition').span())返回(0, 5)
  - print(re.search('super', 'insuperable').span())返回(2, 7)

re.findall(pattern, string)

- 找到 RE 匹配的所有子串，并把它们作为一个列表返回。这个匹配是从左到右有序地返回。如果无匹配，返回空列表。
- 当正则表达式中**含有多个圆括号()时**，列表的元素为**多个字符串组成的元组**，而且**元组中字符串个数与括号对数相同**，并且字符串排放顺序跟括号出现的顺序一致（一般看左括号'（ ‘就行），字符串内容与每个括号内的正则表达式想对应。
- 当正则表达式中**只带有一个圆括号时**，列表中的元素为**字符串**，并且该字符串的内容与括号中的正则表达式相对应。（注意：列表中的字符串只是圆括号中的内容，不是整个正则表达式所匹配的内容。）
- 当正则表达式中**没有圆括号时**，列表中的字符串表示**整个正则表达式匹配的内容**。

- **#正则表达式中没有括号**
- >>> re.findall(r'\w*oo\w*', 'woo this foo is too')
- ['woo', 'foo', 'too']
- **#正则表达式中只有1个括号**
- >>> re.findall(r'.*?(\d+).*?','adsd12343.jl34d5645fd789')
- ['12343', '34', '5645', '789']
- **#正则表达式中有多个括号时**
- add = 'https://www.com.edu//action=?asdfsd and other https://www.baidu.com//a=b'
- re.findall(r'**(w{3}\.)(\w+\.)+(com|edu|cn|net)**',add)
- [('www.', 'com.', 'edu'), ('www.', 'baidu.', 'com')]

re.compile(pattern)

- **s = "this is a python test"**
- **p = re.compile('\w+') #编译正则表达式，获得其对象**
- **print p.findall(s)#用正则表达式对象去匹配内容**
- **print re.findall(p,s)**
- ['this', 'is', 'a', 'python', 'test']

中國人民大學
RENMIN UNIVERSITY OF CHINA

03. BeautifulSoup

简介

- 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
- **beautifulSoup“美味的汤，绿色的浓汤”**

pip install BeautifulSoup4

from bs4 import BeautifulSoup

例子

```
html = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title" name="dromouse"><b>The Dormouse's
story</b></p>
<p class="story">Once upon a time there were three little sisters;
and their names were
<a href="http://example.com/elsie" class="sister" id="link1"><!--
Elsie --></a>,
<a href="http://example.com/lacie" class="sister"
id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister"
id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">...</p>
"""
```

创建对象

- soup=BeautifulSoup(html,'lxml')
- 解析类库
  - Python标准库：BeautifulSoup(html,'html.parser')
  - lxml HTML解析器：BeautifulSoup(html,'lxml')
  - lxml XML解析器：BeautifulSoup(html,['lxml','xml'])
  - lxml XML解析器：BeautifulSoup(html,'xml')
  - BeautifulSoup(html,['lxml'，‘html5lib')

1.基本使用　　2.标准选择器　　3.CSS选择器

Tag选择器

- Tag就是html中的一个标签，用BeautifulSoup就能解析出来Tag的具体内容，具体的格式为soup.name,其中name是html下的标签，具体实例如下：
  - print **soup.title**输出title标签下的内容，包括此标签，这个将会输出<title>The Dormouse's story</title>
  - print **soup.head**

获取Tag名称和属性

Tag有两个重要的属性name和attrs,分别表示名字和属性：

•name:对于Tag，name就是本身，如soup.p.name就是p

•attrs是一个字典类型的，对应的是属性-值，如print soup.p.attrs,输出是{'class': ['title'], 'name': 'dromouse'},

•当然你也可以得到具体值，如print soup.p.attrs['class'],输出的就是[title]是一个列表的类型，因为一个属性可能对应多个值,当然你也可以通过get方法得到属性的，

•如：print soup.p.get('class')。

•还可以直接使用print soup.p['class']

get

- get方法用于得到标签下的属性值，注意这是一个重要的方法，在许多场合都能用到，比如你要得到<img src="#">标签下的图像url,那么就可以用soup.img.get('src')
- print soup.p.get("class")　#得到第一个p标签下的src属性

获取内容：string

- **得到标签下的文本内容，只有在此标签下没有子标签，或者只有一个子标签的情况下才能返回其中的内容，否则返回的是None具体实例如下：**
- print soup.p.string #在上面的一段文本中p标签没有子标签，因此能够正确返回文本的内容
- print soup.html.string #这里得到的就是None,因为这里的html中有很多的子标签
- **print(soup.head.title.string) 嵌套选择**

1：基本使用

get_text()

- 可以获得一个标签中的所有文本内容，包括子孙节点的内容，这是最常用的方法

获取子节点

- **Contents使用**
  - from bs4 import BeautifulSoup
  - soup = BeautifulSoup(html,'lxml')
  - print(**soup.p.contents**)
  - #将p标签下的所有子标签存入到了一个列表中
- **Children使用**
  - **soup.p.children是一个迭代对象，而不是列表，只能通过循环的方式获取素有的信息**
  - for i,child in enumerate(soup.p.children):
  - print(i,child)

获取子孙节点

- 通过contents以及children都是获取子节点

- 如果想要获取子孙节点可以通过descendants
- print(soup.descendants)同时这种获取的结果也是一个迭代器

2：标准选择器

搜索文档树

- **find_all(name , attrs , recursive , text , **kwargs)**
- **find_all是用于搜索节点中所有符合过滤条件的节点**
- **name参数：是Tag的名字，如p,div,title .....**
- **soup.find_all("p") 查找所有的p标签，返回的是[The Dormouse's story]，可以通过遍历获取每一个节点，如下：**

```
ps=soup.find_all("p")
for p in ps:
    print p.get('class')   #得到p标签下的class属性
```

找到文档中所有<a>标签和标签

```
soup.find_all(["a", "b"])
```

2：标准选择器

搜索文档树

- KeyWords参数，就是传入属性和对应的属性值
- soup.find_all(id='link2')
- soup.find_all(href=re.compile("elsie"))
- soup.find_all(id='link2',class_='title') ,这个将会查找到同时满足这两个属性的标签，这里的class必须用class_传入参数，因为class是python中的关键词

2：标准选择器

搜索文档树

- **text参数**：通过 text 参数可以搜搜文档中的字符串内容.与 name 参数的可选值一样, text 参数接受 字符串，正则表达式，列表, True
- soup.find_all(text="Elsie")
  - # [u'Elsie']
- soup.find_all(text=["Tillie", "Elsie", "Lacie"])
  - # [u'Elsie', u'Lacie', u'Tillie']
- soup.find_all(text=re.compile("Dormouse"))
  - [u"The Dormouse's story", u"The Dormouse's story"]

2：标准选择器

搜索文档树

- **limit参数：** find_all() 方法返回全部的搜索结构,如果文档树很大那么搜索会很慢.如果我们不需要全部结果,可以使用 limit 参数限制返回结果的数量.

- **soup.find_all("a", limit=2)**
  - # [Elsie,
  - # Lacie]

2：标准选择器

搜索文档树

- find(name , attrs , recursive , text , **kwargs)

- find_all() 方法的返回结果是值包含一个元素的列表,
- find() 方法直接返回结果,就是直接返回第一匹配到的元素，不是列表，不用遍历，如soup.find("p").get("class")

3：CSS选择器

CSS选择器

- 我们在写 CSS 时，标签名不加任何修饰，类名前加点，id名前加#，在这里我们也可以利用类似的方法来筛选元素，用到的方法是 soup.select()，返回类型是 list

（1）通过标签名查找

print soup.select('title')

#[<**title**>The Dormouse's story</**title**>]

print soup.select('a')

#[<**a** class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></**a**>,

#<**a** class="sister" href="http://example.com/lacie" id="link2">Lacie</**a**>,

#<**a** class="sister" href="http://example.com/tillie" id="link3">Tillie</**a**>]

print soup.select('b')

#[<**b**>The Dormouse's story</**b**>]

（2）通过类名查找

print soup.select('.sister')

#[<!-- Elsie -->,

#Lacie,

#Tillie]

(3) 通过 id 名查找

print soup.select('#link1')

#[<!-- Elsie -->]

（4）组合查找

组合查找即和写 class 文件时，

标签名与类名、id名进行的组合原理是一样的，

例查找 p 标签中，id 等于 link1的内容，二者需要用空格分开

print soup.select('p #link1')

#[<!-- Elsie -->]

直接子标签查找

print soup.select("head > title")

#[<title>The Dormouse's story</title>]

（5）属性查找

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

#[<a class="sister" href="http://example.com/elsie"

id="link1"><!-- Elsie -->]

同样，属性仍然可以与上述查找方式组合，不在同一节点的空格隔开，同一节点的不加空格

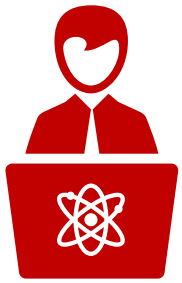
print soup.select('p

a[href="http://example.com/elsie"]')

#[<a class="sister" href="http://example.com/elsie"

id="link1"><!-- Elsie -->]

中國人民大學
RENMIN UNIVERSITY OF CHINA



谢谢大家！