



# 引言

- 在上一章中，构建了一个爬虫，可以通过跟踪链接的方式下载所需的网页。虽然这个例子很有意思，却不够实用，因为爬虫在下载网页之后又将结果丢弃了。
- 现在，让这个爬虫从某些网页中抽取一些数据，然后实现某些事情，这种做法也称为**抓取**（Scraping）。
- 首先，介绍一些浏览器工具，用于查看网页内容；然后，介绍3种抽取网页数据的方法，分别是正则表达式、Beautiful Soup 和Lxml；最后，对比这3种抓取方法。



# 内容提要

## 第2章 数据抓取

分析网页

抓取网页数据的方法

使用网页浏览器控制台

CSS选择器与XPath选择器

抓取结果




## 2.1 分析网页

- 要想理解一个网页的结构如何，可以使用查看源代码的方法。在大多数浏览器中，都可以在页面上右击选择“查看网页源代码”选项，获取网页的源代码。



### Example web scraping website

Flag:   
Area: 244,820 square kilometres  
Population: 62,348,447  
Iso: GB  
Country (District): United Kingdom  
Capital: London  
Continent: EU  
Tld: .uk  
Currency Code: GBP  
Currency Name: Pound  
Phone: 44  
Postal Code Format: @# #@@|@## #@@|@@# #@@|@@## #@@  
Postal Code Regex: ^([A-Z]{2}[A-Z]{2}){1}([A-Z]{3}[A-Z]{2}){1}([A-Z]  
Z]{2}){1}([GIR0AA])\$  
Languages: en-GB,cy-GB,gd  
Neighbours: IE  
[Edit](#)





## 2.1 分析网页

- 对于示例网站来说，感兴趣的数据是在国家（或地区）中。以查看英国示例页面（<http://example.python-scrapping.com/places/default/view/United-Kingdom-233>）为例，可以在源代码中找到一个包含国家（或地区）数据的表格。

```
<table>
  <tr id="places_flag_img__row"><td class="w2p_fl"><label class="readonly"
for="places_flag_img" id="places_flag_img__label">Flag: </label></td><td
class="w2p_fw"></td><td
class="w2p_fc"></td></tr>
  .....
  <tr id="places_neighbours__row"><td class="w2p_fl"><label class="readonly"
for="places_neighbours" id="places_neighbours__label">Neighbours:
</label></td><td class="w2p_fw"><div><a href="/places/default/iso/IE">IE
</a></div></td><td class="w2p_fc"></td></tr>
</table>
```



## 2.1 分析网页

- 对于浏览器解析而言，网页源码中缺失空白符和格式并无大碍，但在阅读时却会造成一定的困难。
- 要想更好地理解该表格，可以借助浏览器工具。在Chrome、Firefox、360等浏览器中，只需右击页面上的某个元素（在抓取时感兴趣的元素），然后选择“审查元素”选项，然后在网页下面会打开一个面板，其中包含了选定元素的HTML层次结构。
- 例如，在国家面积数据处右击，选择选择“审查元素”



## 2.1 分析网页

### Example web scraping website

第2章 数据抓取

Flag:		返回(B)	
Area:	244,820	前进(O)	Alt+ 向右箭头
Population:	62,348,4	重新加载(R)	Ctrl+R
Iso:	GB	发送链接到手机	
Country (District):	United Kingdom	网页另存为(S)...	Ctrl+S
Capital:	London	添加到收藏夹(F)...	
Continent:	EU	全选(A)	Ctrl+A
Tld:	.uk	打印(I)...	Ctrl+P
Currency Code:	GBP	编码(E)	>
Currency Name:	Pound	使用迅雷下载全部链接	
Phone:	44	查看网页源代码(V)	Ctrl+U
Postal Code Format:	@# #@	审查元素(N)	
Postal Code Regex:	^([A-Z]{2})([A-Z]{2})	属性(P)	
Languages:	en-GB, c		
Neighbours:	IE		
<a href="#">Edit</a>			

@@#@ #@|GIR0AA  
([A-Z]{2}\d{3}[A-Z]{2})|([A-Z]\d[A-Z]\d[A-Z]{2})

# 第二章 数据抓取

Flag:

td.w2p\_fw 810.83 × 25

Area:

244,820 square kilometres

Population:

62,348,447

Iso:

GB

Country (District):

United Kingdom

Elements Console Sources Network Performance Memory Application Security Audits 1

Styles Computed Event Listeners

Filter :hov .cls +

element.style { }

td.w2p\_fw { padding-right: 7px; }

td.w2p\_fw { vertical-align: top; }

td { padding-bottom: 1px; }

td { vertical-align: top; }

```
<div class="span12">
  <form action="#" enctype="multipart/form-data" method="post">
    <table>
      <tbody>
        <tr id="places_flag_img_row">...</tr>
        <tr id="places_area_row">
          <td class="w2p_fw">244,820 square kilometres</td> == $0
          <td class="w2p_fw">...</td>
        </tr>
        <tr id="places_population_row">...</tr>
        <tr id="places_iso_row">...</tr>
        <tr id="places_country_or_district_row">...</tr>
        <tr id="places_capital_row">...</tr>
        <tr id="places_continent_row">...</tr>
        <tr id="places_tld_row">...</tr>
        <tr id="places_currency_code_row">...</tr>
        <tr id="places_currency_name_row">...</tr>
      </tbody>
    </table>
  </form>
</div>
```

html body div #main div form table tbody tr#places\_area\_row td.w2p\_fw

- 可以看到，<table>元素位于一个<form>元素中。国家（或地区）属性包含在带有不同 CSS ID（id="places\_flag\_img\_\_row"，id="places\_area\_\_row"等）的<tr>（即表格的行）元素中。
- 可以点击某个元素，通过层次结构定位到页面中看到数据。

## 第2章 数据抓取

```
...
    <div class="span12">
      <form action="#" enctype="multipart/form-data" method="post">
        <table>
          <tbody>
            <tr id="places_flag_img__row">...</tr>
            <tr id="places_area__row">
              <td class="w2p_fl">...</td>
              <td class="w2p_fw">244,820 square kilometres</td> == $0
              <td class="w2p_fc"></td>
            </tr>
            <tr id="places_population__row">
              <td class="w2p_fl">...</td>
              <td class="w2p_fw">62,348,447</td>
              <td class="w2p_fc"></td>
            </tr>
            <tr id="places_iso__row">...</tr>
            <tr id="places_country_or_district__row">...</tr>
            <tr id="places_capital__row">...</tr>
          </tbody>
        </table>
      </form>
    </div>
  </div>
</body>
</html>
```

- 当通过点击<tr>元素旁边的箭头，进一步展开时，可以注意到，每一行都包含一个类(class)名为w2p\_fw的<td>（即表格的列）元素，这些元素都是<tr>元素的子元素。
- 现在，已经了解了该表格的HTML层次结构，并且已经获得了从页面中抓取这些数据的必要信息。





## 2.2 三种网页抓取方法

- 现在，已经了解了该网页的结构，下面介绍3种抓取其中数据的方法。
  - 首先是正则表达式
  - 然后是流行的Beautiful Soup模块
  - 最后是强大的Lxml模块



## 2.2.1 使用正则表达式抓取

- 使用正则模块re对HTML源代码进行字符串匹配
  - 从 <http://example.python-scraping.com/places/default/view/United-Kingdom-233> 页面抓取国家（或地区）面积信息。
  - 办法：使用正则表达式匹配<td>元素中的内容  
`regex = r'<td class="w2p_fw">(.*?)</td>'`

```
import sys
sys.path.append("../ch1") # 自定义包ch1添加到系统路径，以便导入
# 直接使用相对目录，还需要在PyCharm中将ch1包设为源根，以便IDE识别
import re
from commonFunctions import download
from pprint import pprint
url = 'http://example.python-scraping.com/places/default/view/United-Kingdom-233'
html = download(url)
regex = r'<td class="w2p_fw">(.*?)</td>'
result = re.findall(pattern=regex, string=html)

pprint(result) # 表格中与当前正则表达式匹配的单元格有多个。
```



## 2.2.1 使用正则表达式抓取

Downloading: <http://example.python-scrapping.com/places/default/view/United-Kingdom-233>

网页字符集为: utf-8

```
['EU</a>',
```

```
.....
```

```
'<div><a href="/places/default/iso/IE">IE </a></div>']
```

- 表格中与当前正则表达式匹配的单元格有多个，因为国家（或地区）属性都是用了 `<td class="w2p_fw">` 标签。如果只想抓取国家（或地区）面积，可以只选择第2个匹配的元素。

```
print('Area:', result[1])
```

```
# 输出: Area: 244,820 square kilometres
```



## 2.2.1 使用正则表达式抓取

- 如果网页结构发生变化（例如，行的次序），该方案可能会失效。一个可能的办法：让正则表达式更加明确，可以将其父元素<tr>也加入进来，由于该元素具有ID属性，所以应该是唯一的。

■ 正则表达式修改为：

```
regex = r"<tr id='places_area__row'><td class='w2p_fl'><label  
class='readonly' for='places_area' id='places_area__label'>Area:  
</label></td><td class='w2p_fw'>(.*?)</td>"
```

- 注意，regex中不要插入换行符（即换行书写），否则匹配不到。
- 再次运行结果结果如下：  
[ '244,820 square kilometres']



## 2.2.1 使用正则表达式抓取

- 这个迭代版本看起来更好一些，但网页更新还要很多其他方式，同样可以让该正则表达式无法满足。例如，将双引号改为单引号，<td>标签之间添加多个空格，或者是变更area\_label等。
- 下面是尝试支持这些可能性的改进正则表达式：

regex =

```
r'"<tr id="places_area__row">.*?<td\s+class=["']w2p_fw["']>(.*?)</td>"'
```

- 注意，\s 匹配任何空白字符，包括空格、制表符、换页符等等，等价于 [ \f\n\r\t\v ] 。
- 再次运行结果结果如下：  
[ '244,820 square kilometres']



## 2.2.1 使用正则表达式抓取

- 利用正则表达式，尝试抓取网页中的某个表格数据，例如：

[https://www.fortunechina.com/fortune500/c/2018-07/19/content\\_311046.htm](https://www.fortunechina.com/fortune500/c/2018-07/19/content_311046.htm)

2018年财富世界500强排行榜 - 财 x

+

→ ↻

https://www.fortunechina.com/fortune500/c/2018-07/19/content\_311046.htm

财富FORTUNE

2018年财富世界500强排行榜

🔍 请输入关键词

🗨️

🐦

in

🖨️

Tt

排名	上年排名	公司名称 (中英文)	营业收入 (百万美元)	利润 (百万美元)	国家
1	1	沃尔玛 (WALMART)	500,343	9,862	美国
2	2	国家电网公司 (STATE GRID)	348,903.1	9,533.4	中国
3	3	中国石油化工集团公司 (SINOPEC GROUP)	326,953	1,537.8	中国
4	4	中国石油天然气集团公司 (CHINA NATIONAL PETROLEUM)	326,007.6	-690.5	中国
5	7	荷兰皇家壳牌石油公司 (ROYAL DUTCH SHELL)	311,870	12,977	荷兰
6	5	丰田汽车公司 (TOYOTA MOTOR)	265,172	22,510.1	日本
7	6	大众公司 (VOLKSWAGEN)	260,028.4	13,107.3	德国



## 2.2.1 使用正则表达式抓取

- 利用正则表达式，尝试抓取网页中的某个表格数据，例如：

[https://www.fortunechina.com/fortune500/c/2018-07/19/content\\_311046.htm](https://www.fortunechina.com/fortune500/c/2018-07/19/content_311046.htm)

思路：

- **step 1**：根据表格id属性匹配到表格；
- **step 2**：匹配处表格中的所有<tr>元素找到所有行，存入rows列表；
- **step 3**：循环在每个row中匹配<th>或<td>，找出各列表头或数据。

注意，在使用re.findall方法时，指定参数 `flags = re.S`，以匹配任意字符，包括换行符\n，否则不包括换行符（即匹配输出到换行符就结束）。例如：

```
tables = re.findall(pattern=regex, string=html, flags=re.S)
```



## 2.2.1 使用正则表达式抓取

- 虽然该表达式更容易适应未来变化，但有存在难以构造、可读性差的问题。
- 此外，还有很多其他微小的布局变化也会使该正则表达式无法满足，例如在<td>中添加title属性，或者tr、td元素修改了它们的CSS类或ID。
- 从本例中可以看出，正则表达式为提供了抓取数据的快捷方式，但是该方法过于脆弱，容易在网页更新后出现问题。
- 幸好，还有更好的数据抽取解决方案，例如接下来要介绍的两个抓取库：Beautiful Soup 和 Lxml。





## 2.2.2 使用Beautiful Soup库抓取

- Beautiful Soup是一个非常流行的Python库，它可以解析网页，并提供了定位内容的便捷接口。
- 如果你还没有安装该模块，可以使用下面的命令安装器最新版。

`pip install beautifulsoup4`

- 使用Beautiful Soup的第一步是将已下载的HTML内容解析为soup文档。



## 2.2.2 使用Beautiful Soup库抓取

- 由于许多网页都不具备良好的HTML格式，因此Beautiful Soup需要对其标签开合状态进行修正。
- 例如，在下面的这个简单网页的列表中，存在属性值两侧引号缺失和标签未闭合的问题。

```
<ul class=country_or_district>
```

```
<li>Area
```

```
<li>Population
```

```
</ul>
```

- 如果population列表项被解析为Area列表项的子元素，而不是并列的两个列表项话，在抓取时就会得到错误的结果。



## 2.2.2 使用Beautiful Soup库抓取

- 看一下 Beautiful Soup 是如何处理的：

```
from bs4 import BeautifulSoup
broken_html = '<ul class=country_or_district><li>Area<li>Population</ul>'
# parse the HTML
soup = BeautifulSoup(broken_html, features='html.parser')
"""
:param features: Desirable features of the parser to be
| used. This may be the name of a specific parser ("lxml",
| "lxml-xml", "html.parser", or "html5lib") or it may be the
| type of markup to be used ("html", "html5", "xml"). It's
| recommended that you name a specific parser, so that
| BeautifulSoup gives you the same results across platforms
| and virtual environments.
"""
fixed_html = soup.prettify()
print(fixed_html)
```



## 2.2.2 使用Beautiful Soup库抓取

- 运行结果

```
<ul class="country_or_district">
```

```
<li>
```

```
Area
```

```
<li>
```

```
Population
```

```
</li>
```

```
</li>
```

```
</ul>
```

- 可以看到，使用默认的 `html.parser` 并没有得到正确解析的HTML。



## 2.2.2 使用Beautiful Soup库抓取

- 从前面的代码片段可以看出，由于它使用了嵌套的<li>元素，因此可能会导致定位困难。
- 幸运的是，还有其他解析器可以选择。可以安装LXML，或使用html5lib（用**`pip install html5lib`**安装）
- 修改解析器为html5lib后的代码：

```
from bs4 import BeautifulSoup
broken_html = '<ul class=country_or_district><li>Area<li>Population</ul>'
# parse the HTML
soup = BeautifulSoup(broken_html, features='html5lib')
fixed_html = soup.prettify()
print(fixed_html)
```



## 2.2.2 使用Beautiful Soup库抓取

- 运行结果

```
<html>
<head>
</head>
<body>
  <ul class="country_or_district">
    <li>
      Area
    </li>
    <li>
      Population
    </li>
  </ul>
</body>
</html>
```

- 此时，使用了html5lib的Beautiful Soup已经能够正确解释缺失的属性引号以及闭合标签。并且还添加了<html>和<body>标签，使其成为完整的HTML文档。
- 使用lxml解析器，也可以看到类似的结果。



## 2.2.2 使用Beautiful Soup库抓取

- 现在可以使用find()和find\_all()方法定位需要的元素了。

```
.....  
ul = soup.find(name='ul', attrs={'class': 'country_or_district'})  
print(ul.find(name='li')) # return just the first match  
print(ul.find_all(name='li')) # return all matches
```

- 输出：

<li>Area</li>

[<li>Area</li>, <li>Population</li>]



## 2.2.2 使用Beautiful Soup库抓取

- BeautifulSoup中的find(), find\_all(), select()函数
  - **find()**函数：输出第1个匹配对象，即find\_all()[0]。
  - **find\_all()**函数：**findAll**(name=None, attrs={}, recursive=True, text=None, limit=None, \*\*kwargs)
    - 其中最重要的参数是name和kwargs。参数name匹配tags的名字，获得相应的结果集。有几种方法去匹配name，最简单用法是仅仅给定一个tag的名字。
    - 返回一个列表。
  - **select()**函数跟find和find\_all一样，用来选取特定的标签，其选取规则依赖于css，把它叫做css选择器。





## 2.2.2 使用Beautiful Soup库抓取

- find\_all()函数使用示例

- 直接传递标签名，寻找文档中所有tr标签：

```
soup.findAll(name='tr')
```

- 传递正则表达式编译对象，寻找所有以b开头的标签：

```
soup.findAll(name=re.compile('^b'))
```

- 使用列表或字典，寻找所有tr和th标签：

```
soup.findAll(name=['tr', 'th'])
```

```
soup.findAll(name={'tr':True, 'th':True})
```

- 传入布尔值，匹配所有标签：

```
allTags = soup.findAll(name=True)
```



## 2.2.2 使用Beautiful Soup库抓取

### ● find\_all()函数使用示例

- 传入一个布尔型的callable对象作为标签过滤器，它是一个使用Tag对象作为它唯一的参数，并返回布尔值的对象，一般为lamda函数。
- findAll使用的每个作为参数的Tag对象都会传递给这个callable对象，并且如果调用返回True，则这个tag便是匹配的，类似于Filter函数。

□ 查找仅有两个属性的标签：

```
soup.findAll(lambda tag: len(tag.attrs)==2)
```

□ 寻找以单个字符为标签名并且没有属性的标签：

```
findAll(lambda tag: len(tag.name)==1 and not tag.attr)
```



## 2.2.2 使用Beautiful Soup库抓取

- find\_all()函数使用示例

- kwargs参数可用于筛选tag的属性，可以传递字符串，来匹配属性的值。也可以传递正则表达式、列表、特殊值True或None，或者一个可调用的以属性值为参数的对象(注意：这个值可能为None)

- 查找align属性为center的标签：

`soup.findAll(align='center')`

- 查找align属性为center或left的标签：

`soup.findAll(align=['center','left'])`

- 查找id属性以para结尾的标签：

`soup.findAll(id=re.compile('para$'))`

- True匹配给定属性为任意值的标签，None则匹配属性值为空的标签。查找align属性不为空的标签：

`soup.findAll(align=True)`



## 2.2.2 使用Beautiful Soup库抓取

- find\_all()函数使用示例

- 同时指定标签名和单个属性：

```
soup.findAll(name='a', href='jd.com')
```

- 同时指定标签名和多个属性：

```
soup.findAll(name='a', href='jd.com', title='京东')
```

```
soup.findAll(name='a',  
             attr={'href':'jd.com', 'title':'京东'})
```



## 2.2.2 使用Beautiful Soup库抓取

### ● select()函数使用示例

- 找出所有tr标签: `soup.select('tr')`
- 找出所有th或td标签: `soup.select('th,td')`
- 找出所有**id属性**为places\_area\_\_row的元素(id值前面需加#): `soup.select('#places_area__row')`
- 找出所有**class属性**为w2p\_fw的元素(class值前面需加.): `soup.select('.w2p_fw')`
- 通过**其他属性**查找: `soup.select('[href="jd.com"]')`
- 组合查找, 可以分为两种: ① 在一个tag中进行两个条件的查找; ② 树状一层一层之间的查找, 在分析html结构是是非常常见的, 层和层之间用空格分开.
  - 选择标签名为a, id属性为link2的tag:  
`soup.select('a#link2')`
  - 从body开始, 查找其中所有的p标签的a标签, 且a标签的id属性为link2:  
`soup.select('body p a#link2')`

, features='html5lib'



## 2.2.2 使用Beautiful Soup库抓取

第2章  
数据抓取

- 示例：抽取示例网站 url = 'http://example.python-scrapping.com/places/default/view/United-Kingdom-233'  
中国（或地区）面积数据----使用find()函数实现

```
...
from bs4 import BeautifulSoup
url = 'http://example.python-scrapping.com/places/default/view/United-Kingdom-233'
html = download(url)
# parse the HTML
soup = BeautifulSoup(html, features='html5lib') # 默认解析器则为html.parser

tr = soup.find(attrs={'id': 'places_area__row'}) # locate the area row
td = tr.find(attrs={'class': 'w2p_fw'}) # locate the data element
area = td.text # extract the text from the data element
print(area)
```



## 2.2.2 使用Beautiful Soup库抓取

- 运行结果：  
**244,820 square kilometres**
- 这段代码虽然比正则表达式的代码更加复杂，但又更容易构造和理解。而且，像多余的空格和标签属性这种布局上的小变化，也无需再担心了。
- 即使页面中包含了不完整的HTML，Beautiful Soup也能帮助整理该页面，从而可以从非常不完整的网站代码中抽取数据。



## 2.2.2 使用Beautiful Soup库抓取

- 示例：抽取示例网站 url = 'http://example.python-scrapping.com/places/default/view/United-Kingdom-233'  
国家（或地区）面积数据----使用select()函数实现

```
html = download(url)
# parse the HTML
soup = BeautifulSoup(html, features='html5lib')
```

```
tr = soup.select('tr#places_area__row')[0] # id="places_area__row"
td = tr.select('td.w2p_fw')[0]
area = td.text # extract the text from the data element
print(area)
```

```
# 直接逐层查找定位元素，层和层之间用空格分开
selector='tr#places_area__row td.w2p_fw'
td2 = soup.select(selector)[0]
area = td2.text # extract the text from the data element
print(area)
```





## 2.2.3 使用Lxml库抓取

- Lxml是基于libxml2这一解析库构建的 Python库，它使用C语言编写，解析速度比Beautiful Soup更快。
- 安装lxml: `pip install lxml`

```
D:\.....\PycharmProjects>pip install lxml
Collecting lxml
  Downloading lxml-4.5.2-cp38-cp38-win32.whl (3.2 MB)
    |████████████████████████████████████████| 3.2 MB 63 kB/s
Installing collected packages: lxml
Successfully installed lxml-4.5.2
D:\.....\PycharmProjects>
```

- 和Beautiful Soup一样，使用lxml模块的第一步也是将有可能不合法的HTML解析为统一格式。



## 2.2.3 使用Lxml库抓取

- 示例：使用该模块解析同一个不完整HTML的例子。

```
from lxml.html import fromstring, tostring

broken_html = '<ul class=country_or_district><li>Area<li>Population</ul>'

# parse the HTML
tree = fromstring(html=broken_html)
fixed_html = tostring(doc=tree, pretty_print=True) # type:bytes # 返回的
是字节数组
print(fixed_html)

fixed_html2 = fixed_html.decode(encoding='utf-8') # bytes-->str
print(fixed_html2)
```



## 2.2.3 使用Lxml库抓取

### ● 运行结果

```
b'<ul class="country_or_district">\n<li>Area</li>\n<li>Population</li>\n</ul>\n'
```

```
<ul class="country_or_district">
<li>Area</li>
<li>Population</li>
</ul>
```

- 同样地，lxml也可以正确解析属性两侧缺失的引号，并闭合标签。
- 不过该模块没有额外添加<html>和<body>标签。这些都不是标准XML的要求。因此，对于lxml说，插入它们并不是必要的。



## 2.2.3 使用Lxml库抓取

- 解析完成输入内容之后，进入选择元素的步骤。此时lxml有几种不同的方法，比如XPath选择器和类似Beautiful Soup的find()方法。
- 不过在本例中，将会使用CSS选择器，因为它更加简洁，并且能够在解析动态内容时得以复用。
- 要想使用CSS选择器，需要先安装cssselect库。

```
D:\.....\PycharmProjects>pip install cssselect
Collecting cssselect
  Downloading cssselect-1.1.0-py2.py3-none-any.whl (16 kB)
Installing collected packages: cssselect
Successfully installed cssselect-1.1.0
D:\.....\PycharmProjects>
```



## 2.2.3 使用Lxml库抓取

- 示例：使用lxml的CSS选择器抽取示例页面中的面积数

```
...  
from lxml.html import fromstring  
  
url = 'http://example.python-scraping.com/places/default/view/United-Kingdom-233'  
html = download(url)  
# parse the HTML  
tree = fromstring(html=html)  
td = tree.cssselect('tr#places_area__row > td.w2p_fw')[0]  
area = td.text_content() # 迭代所有子元素并返回每个元素的相关文本  
print(area)
```

- `cssselect()` 方法的返回是一个列表。
- CSS选择器语法解释：选择表格中ID为`places_area__row`的行元素`<tr>`，然后选择class为`w2p_fw`的子表格（单元格）`<td>`数据标签。



## 2.2.3 使用Lxml库抓取

- 由于cssselect()方法的返回是一个列表，需要获取其中的第一个结果，并调用text\_content()方法，以迭代所有子元素并返回每个元素的相关文本。
- 在本例中，尽管只有一个元素，但是该功能对于更加复杂的抽取示例来说非常有用。

```
trs = tree.cssselect('tr#places_area__row') # 选择满足条件的所有行，  
cssselect()方法的返回是一个列表  
for tr in trs:  
    print(tr.text_content()) # 迭代所有子元素并返回每个元素的相关文本  
print(trs[0].text_content())
```

- 结果：

Area: 244,820 square kilometres

Area: 244,820 square kilometres



## 2.2.3 使用Lxml库抓取

- 由于cssselect()方法的返回是一个列表，需要获取其中的第一个结果，并调用text\_content()方法，以迭代所有子元素并返回每个元素的相关文本。
- 在本例中，尽管只有一个元素，但是该功能对于更加复杂的抽取示例来说非常有用。

```
trs = tree.cssselect('tr') # 选择满足条件的所有行
for tr in trs:
    print(tr.text_content()) # 迭代所有子元素并返回每个元素的相关文本
```

- 结果：

**Flag:**

**Area: 244,820 square kilometres**

**Population: 62,348,447**

.....

**Neighbours: IE**



## 2.3 CSS选择器和浏览器控制台

- 类似在使用cssselect时使用的标记，CSS选择器可以表示选择元素时所使用的模式。
- 下面是一些你需要知道的常用选择器示例

Select any tag:	*
Select by tag <a>:	a
Select by class of "link":	.link
Select by tag <a> with class "link":	a.link
Select by tag <a> with ID "home":	a#home
Select by child <span> of tag <a>:	a > span
Select by tag <a> with attribute title of "Home":	a[title=Home]
Select by tag <th> or <td>:	th,td

cssselect库实现了大部分CSS3选择器的功能

CSS3规范由万维网联盟(World Wide Web Consortium, W3C)  
提出





## 2.3 CSS选择器和浏览器控制台

- 在第一次编写CSS选择器时可能不会十分完美，因此有时测试CSS选择器十分有用。
- 在编写大量无法确定能够工作的Python代码之前，在某个地方调用任何与选择器相关的问题进行测试是一个不错的注意。
- 当一个网站使用jQuery时，可以非常容易的在浏览器控制台中测试CSS选择器。
- 使用包含jQuery的CSS选择器时，你唯一需要知道的语法就是对象选择（例如 `$('div.class_name')`），jQuery使用\$和圆括号来选择对象。

jQuery 是一个 JavaScript 库，极大地简化了 JavaScript 编程。



## 2.3 CSS选择器和浏览器控制台

- 通过 jQuery，您可以选取（查询，query）HTML 元素，并对它们执行“操作”（actions）。
- 基础语法： **`$(selector).action()`**
  - 美元符号\$定义 jQuery
  - 选择字符串（selector）“查询”和“查找”HTML 元素
  - jQuery 的 `action()` 执行对元素的操作
- 实例：
  - `$(this).hide()` - 隐藏当前元素
  - `$('p').hide()` - 隐藏所有 `<p>` 元素
  - `$('p.test').hide()` - 隐藏所有 `class="test"` 的 `<p>` 元素
  - `$('#test').hide()` - 隐藏 `id="test"` 的元素



## 2.3 CSS选择器和浏览器控制台

- jQuery 选择器

- jQuery 选择器允许您对 HTML 元素组或单个元素进行操作。
- jQuery 选择器基于元素的 id、类、类型、属性、属性值等"查找"（或选择）HTML 元素。它基于已经存在的 CSS 选择器，除此之外，它还有一些自定义的选择器。
- jQuery 中所有选择器都以美元符号开头：\$()。

- 元素选择器

- jQuery 元素选择器基于元素名选取元素。例如，在页面中选取所有 <p> 元素：\$("p")。



## 2.3 CSS选择器和浏览器控制台

### ● #id 选择器

- jQuery #id 选择器通过 HTML 元素的 id 属性选取指定的元素。
- 页面中元素的 id 应该是唯一的（实际上也有相同的，但少见），所以您要在页面中选取唯一的元素需要通过 #id 选择器。
- 通过 id 选取元素语法：`$("#test")`表示选组选取id= "test"的元素

### ● .class 选择器

- jQuery 类选择器可以通过指定的 class 查找元素。
- 语法：`$(".test")`



## 2.3 CSS选择器和浏览器控制台

### ● 更多jQuery 选择器示例

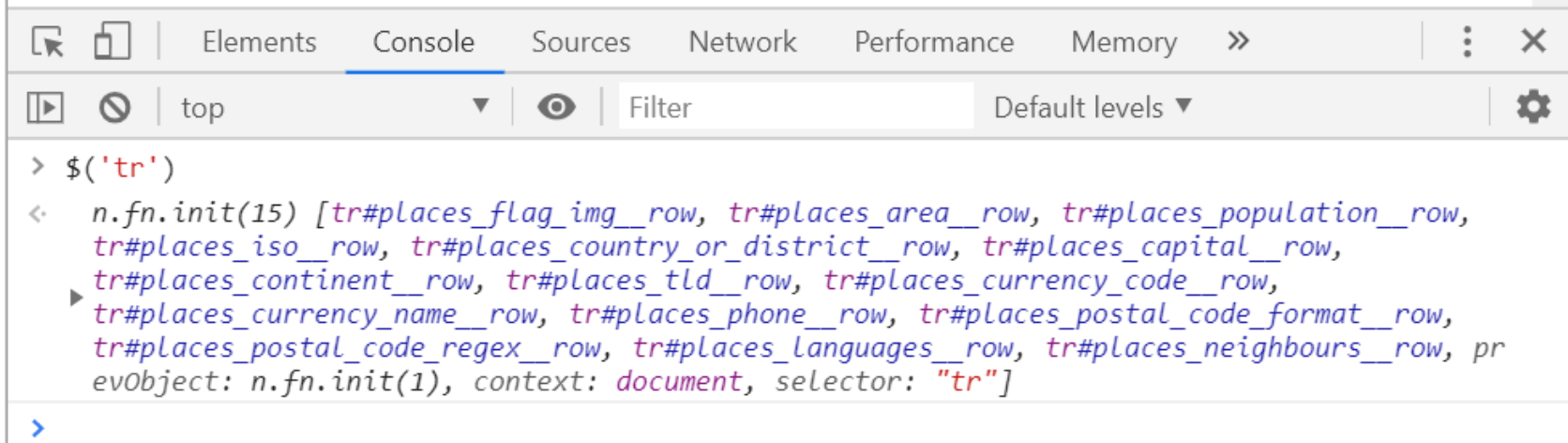
语法	描述
<code>\$("*")</code>	选取所有元素
<code>\$(this)</code>	选取当前 HTML 元素
<code>\$("p.intro")</code>	选取 class 为 intro 的 <p> 元素
<code>\$("p:first")</code>	选取第一个 <p> 元素
<code>\$("ul li:first")</code>	选取第一个 <ul> 元素的第一个 <li> 元素
<code>\$("ul li:first-child")</code>	选取每个 <ul> 元素的第一个 <li> 元素
<code>\$("[href]")</code>	选取带有 href 属性的元素
<code>\$("a[target='_blank']")</code>	选取所有 target 属性值等于 "_blank" 的 <a> 元素
<code>\$("a[target!='_blank']")</code>	选取所有 target 属性值不等于 "_blank" 的 <a> 元素
<code>\$(":button")</code>	选取所有 type="button" 的 <input> 元素 和 <button> 元素
<code>\$("tr:even")</code>	选取偶数位置的 <tr> 元素
<code>\$("tr:odd")</code>	选取奇数位置的 <tr> 元素



## 2.3 CSS选择器和浏览器控制台

- 对于支持jQuery的站点，你可以在浏览器控制台中执行它，可以看到你所选择的对象。

### Example web scraping website



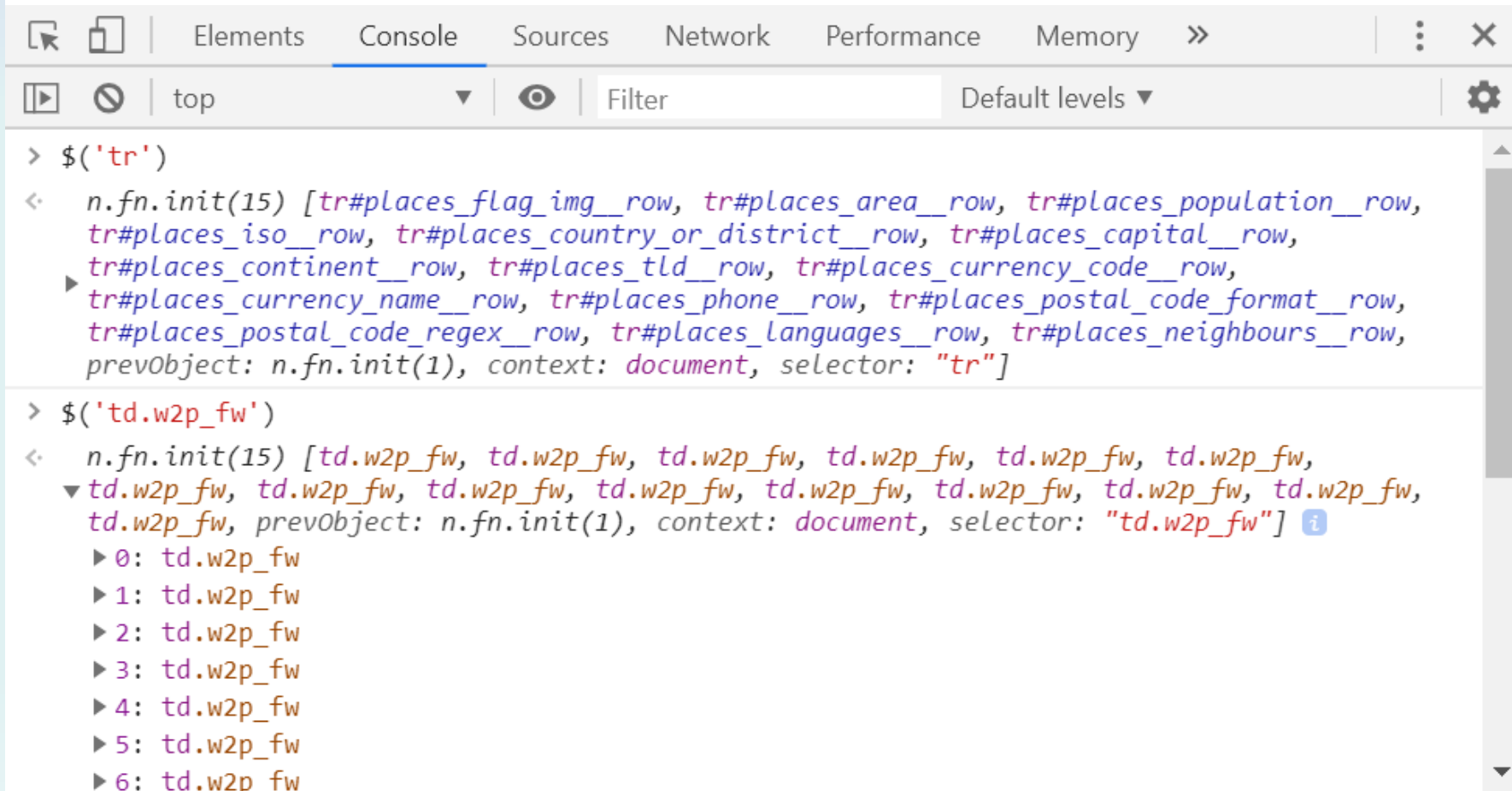
由于已经知道示例网站使用了jQuery(查看源码便知)，可以尝试使用CSS选择器选择

- 所有的tr元素: `$('tr')`
- 所有的class为w2p\_fw的td元素: `$('td.w2p_fw')`



## 2.3 CSS选择器和浏览器控制台

- 对于支持jQuery的站点，你可以在浏览器控制台中执行它，可以看到你所选择的对象。





# 第2章 数据抓取

Flag:

td.w2p\_fw 594.17 × 25

Area:

244,820 square kilometres

Population:

62,348,447

Iso:

GB

Country (District):

United Kingdom

Capital:

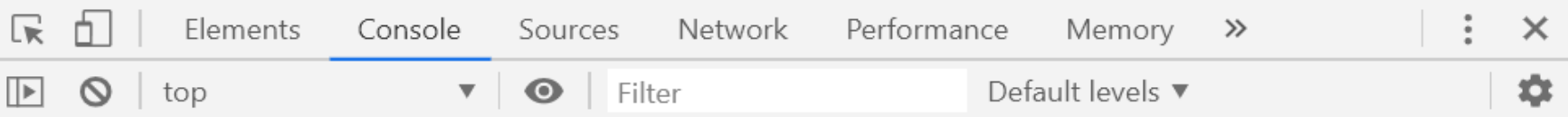
London

Continent:

EU

Tld:

.uk



```
> $('tr')
< n.fn.init(15) [tr#places_flag_img__row, tr#places_area__row, tr#places_population__row, tr#places_iso__row, tr#places_country_or_district__row, tr#places_capital__row, tr#places_continent__row, tr#places_tld__row, tr#places_currency_code__row, tr#places_currency_name__row, tr#places_phone__row, tr#places_postal_code_format__row, tr#places_postal_code_regex__row, tr#places_languages__row, tr#places_neighbours__row, prevObject: n.fn.init(1), context: document, selector: "tr"]
```

```
> $('td.w2p_fw')
< n.fn.init(15) [td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, td.w2p_fw, prevObject: n.fn.init(1), context: document, selector: "td.w2p_fw"]
  0: td.w2p_fw
  1: td.w2p_fw
  2: td.w2p_fw
  3: td.w2p_fw
  4: td.w2p_fw
  5: td.w2p_fw
  6: td.w2p_fw
```

试试将鼠标移动到这些选择器上





## 2.3 CSS选择器和浏览器控制台

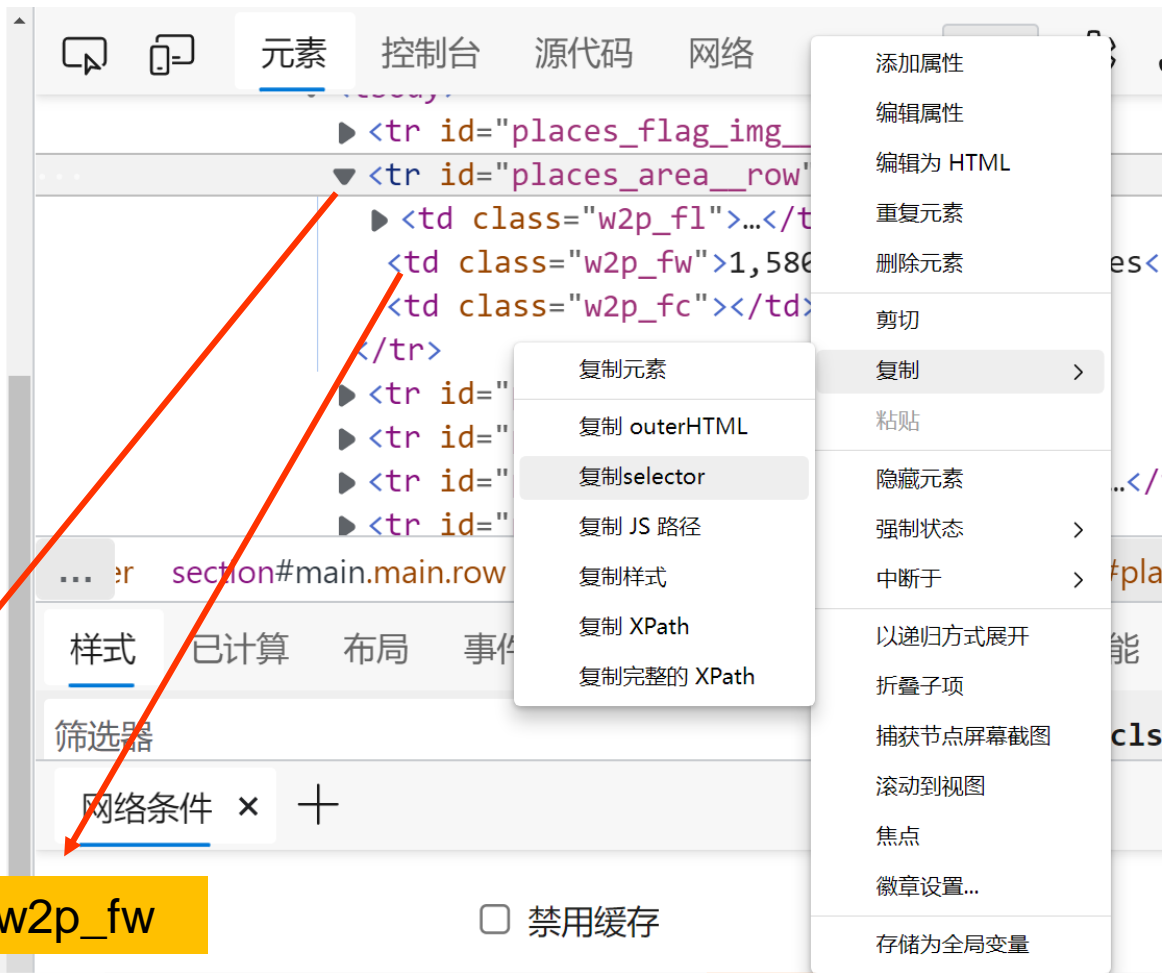
- 通过浏览器审查元素视图获取css选择器或XPath选择器

### 第2章 数据抓取

tr#places_area__row	289.81 × 25
Area:	1,580 square kilometres
Population:	26,711
Iso:	AX
Country (District):	Aland Islands
Capital:	Mariehamn
Continent:	EU
Tld:	.ax
Currency Code:	EUR
Currency Name:	Euro
Phone:	+358-18
Postal Code Format:	#####
Postal Code Regex:	^(?:FI)*(\d{5})\$
Languages:	sv-AX
Neighbours:	
Edit	

#places\_area\_\_row

#places\_area\_\_row > td.w2p\_fw





## 2.3 CSS选择器和浏览器控制台

- 通过浏览器审查元素视图获取css选择器或XPath选择器

### 第2章 数据抓取

tr#places\_area\_\_row 289.81 × 25

Area:	1,580 square kilometres
Population:	26,711
Iso:	AX
Country (District):	Aland Islands
Capital:	Mariehamn
Continent:	EU
Tld:	.ax
Currency Code:	EUR
Currency Name:	Euro
Phone:	+358-18
Postal Code Format:	#####
Postal Code Regex:	^(?:FI)*(\d{5})\$
Languages:	sv-AX
Neighbours:	

Edit

//\*[ @id="places\_area\_\_row"]

//\*[ @id="places\_area\_\_row"]/td[2]

禁用缓存

第50页



## 2.4 XPath选择器

- 即使你已经学会了在控制台中使用lxml和CSS选择器，学习XPath依然是非常有用的，因为lxml在求值之前会将所有的CSS选择器转换为XPath。
- 有时候使用CSS选择器无法正常工作，尤其是在HTML非常不完整或存在格式不当的元素时。
- 尽管像Beautiful Soup和lxml这样的库已经尽了最大努力来纠正解析并清理代码，然而它可能还是无法工作，在这些情况下，XPath可以帮助你基于页面中的层次结构关系构建非常明确的选择器。



## 2.4 XPath选择器

- XPath是一种将XML文档的层次结构描述为关系的方式。因为HTML是由XML元素组成，因此也可以使用XPath从HTML文档中定位和选择元素。
- XPath遵循一些基本的语法规则，并且和CSS选择器有些许相似。下表给出两种方法的一些快速参考：

选择器描述	XPath选择器	CSS选择器
选择所有链接	'//a'	'a'
选择类名为"main"的div元素	'//div[@class="main"]'	'div.main'
选择id属性为"list"的ul元素	'//ul[@id="list"]'	'ul#list'
从所有段落中选择文本	'//p/text()'	'p*'
选择所有类名中包含"test"的div元素	'//div[contains(@class,'test')]'	'div [class*="test"]'
选择所有包含链接或列表的div元素	'//div[a ul]'	'div a,div ul'
选择href属性中包含jd.com的链接	'//a[contains(@href,'jd.com')]'	'a[href*="jd.com"]'



## 2.4 XPath选择器

选择器描述	XPath选择器	CSS选择器
选择所有链接	'//a'	'a'
选择类名为"main"的div元素	'//div[@class="main"]'	'div.main'
选择id属性为"list"的ul元素	'//ul[@id="list"]'	'ul#list'
从所有段落中选择文本	'//p/text()'	'p*'
选择所有类名中包含"test"的div元素	'//div[contains(@class,'test')]'	'div [class*="test"]'
选择所有包含链接或列表的div元素	'//div[a ul]'	'div a,div ul'
选择href属性中包含jd.com的链接	'//a[contains(@href,'jd.com')]'	'a[href*="jd.com"]'

- 从表中可以看到，两种方式有许多相似之处。不过，表中有一些CSS选择器使用\*表示，代表使用CSS选择这些元素是不可能的，已经提供了最佳的替代方案。



## 2.4 XPath选择器

- 通过浏览器审查元素视图获取css选择器或XPath选择器

第2章 数据抓取

tr#places\_area\_\_row 289.81 × 25

Area:	1,580 square kilometres
Population:	26,711
Iso:	AX
Country (District):	Aland Islands
Capital:	Mariehamn
Continent:	EU
Tld:	.ax
Currency Code:	EUR
Currency Name:	Euro
Phone:	+358-18
Postal Code Format:	#####
Postal Code Regex:	^(?:FI)*(\d{5})\$
Languages:	sv-AX
Neighbours:	

Edit

元素 控制台 源代码 网络 性能 内存

▼ <tr id="places\_area\_\_row"> == \$0

- ▶ <td class="w2p\_fl">...</td>
- ▶ <td class="w2p\_fw">1,580 square kilom
- ▶ <td class="w2p\_fc"></td>

</tr>

▶ <tr id="places\_population\_\_row">...</tr>

▶ <tr id="places\_iso\_\_row">...</tr>

▶ <tr id="places\_coun">...</tr>

▶ <tr id="places\_capi">...</tr>

▶ <tr id="places\_cont">...</tr>

▶ <tr id="places\_tld">...</tr>

▶ <tr id="places\_curr">...</tr>

样式 已计算 布局 事件侦听器

筛选器

复制元素

复制 outerHTML

复制selector

复制 JS 路径

复制样式

复制 XPath

复制完整的 XPath

添加属性

编辑为 HTML

重复元素

删除元素

剪切

复制 >

粘贴

隐藏元素

强制状态 >

中断于 >

以递归方式展开

折叠子项

捕获节点屏幕截图

滚动到视图

焦点

徽章设置...

存储为全局变量

//\*[ @id="places\_area\_\_row"]

//\*[ @id="places\_area\_\_row"]/td[2]

禁用缓存



## 2.4 XPath选择器

- 示例：用lxml的XPath选择器抽取示例页面中的面积数据

```
...  
from lxml.html import fromstring  
  
url = 'http://example.python-scraping.com/places/default/view/United-  
Kingdom-233'  
html = download(url)  
# parse the HTML  
tree = fromstring(html=html)  
td = tree.xpath('//tr[@id="places_area__row"]/td[@class="w2p_fw"]')[0]  
area = td.text_content() # 迭代所有子元素并返回每个元素的相关文本  
print(area)
```

```
texts = tree.xpath('//td[@class="w2p_fw"]/text()')  
for text in texts:  
    print(text)
```







## 2.4 XPath选择器


- 和CSS选择器类似，你同样也可以在浏览器控制台中测试XPath选择器。要想实现该目的，只需在页面中使用 `$x('patter_here')` 选择器。
- 例如，想要测试查找带有图片的td元素，来获取国家(或地区)页面中的旗帜数据的话，可以先在浏览器中测试XPath表达式：
  - `$x('//td/img')`
  - `$x('//td/img/@src')`





## 2.4 XPath选择器

img 52.99 × 26.99

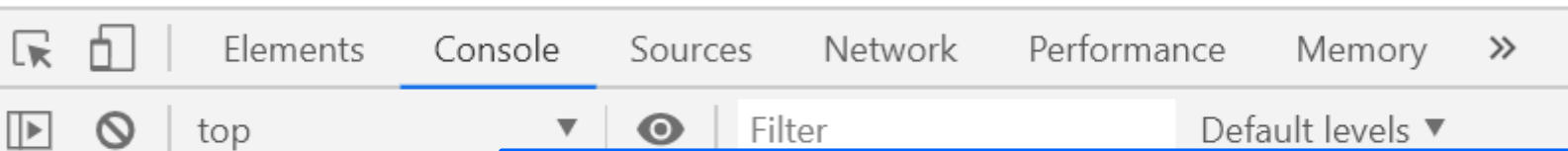
Flag: 

Area: 244,820 square kilometres

Population: 62,348,447

Iso: GB

Country (District): United Kingdom



```
> $x('//td/img')
```

```
< [img] i
  0: img
    length: 1
    __proto__: Array(0)
```

```
> $x('//td/img/@src')
```

```
< [src] i
  0: src
    length: 1
    __proto__: Array(0)
```

```
outerHTML: ""
```

```
> $x('//td/img')[0].innerHTML
```

```
< ""
```

```
1.textContent: "/places/static/images/flags/gb.png"
2.value: "/places/static/images/flags/gb.png"
```

```
> $x('//td/img/@src')[0].textContent
```

```
< "/places/static/images/flags/af.png"
```



```
> $x('//tr')[1].outerHTML
```

```
< "<tr id="places_area__row"><td class="w2p_fl"><label c  
lass="readonly" for="places_area" id="places_area__lab  
el">Area: </label></td><td class="w2p_fw">647,500 squa  
re kilometres</td><td class="w2p_fc"></td></tr>"
```

```
> $x('//tr')[1].innerHTML
```

```
< "<td class="w2p_fl"><label class="readonly" for="place  
s_area" id="places_area__label">Area: </label></td><td  
class="w2p_fw">647,500 square kilometres</td><td class  
="w2p_fc"></td>"
```

```
> $x('//tr')[1].outerText
```

```
< "Area:  
647,500 square kilometres"
```

```
> $x('//tr')[1].innerText
```

```
< "Area:  
647,500 square kilometres"
```

```
> $x('//tr')[1].textContent
```

```
< "Area: 647,500 square kilometres"
```



## 2.4 XPath选择器

- 在这里可以看到，可以使用属性来指明想要抽取的数据（比如，@src）。
- 通过在浏览器的控制台中进行测试，可以凭借获取即时并且易读的结果，节省调试时间。
- 在本章及后续章节，XPath和CSS选择器都会使用到，这样你就可以更加熟悉它们，并且在你提高自己的网站爬虫能力时，对它们的使用更加自信。



## 2.5 LXML和家族树

- lxml同样也有遍历HTML页面中家族树的能力。
- 家族树是什么？当使用浏览器开发者工具来查看页面中的元素时，可以展开或缩进它们，这就是观察HTML的家族关系。网页中的每个元素都包含父亲、兄弟和孩子。这些关系有助于更加容易地遍历页面。
- 例如，当希望查找页面中同一节点深度的所有元素时，就需要查找它们的兄弟；或是希望得到页面中某个特定元素的所有子元素时。
- lxml允许通过简单Python代码大量使用此类关系。



## 2.5 LXML和家族树

- 示例：查看table元素的所有子元素

```
.....  
tree = fromstring(html=html)  
table = tree.xpath('//table')[0] # xpath()方法的返回是一个列表  
children = table.getchildren() # 查看table元素的所有子元素  
pprint(children)
```

```
[<Element tr at 0x2de22a8>,  
 <Element tr at 0x2de23e8>,  
 <Element tr at 0x2de2460>,  
 <Element tr at 0x2ee9848>,  
 <Element tr at 0x2ee9870>,  
 .....  
 <Element tr at 0x2ee9a00>,  
 <Element tr at 0x2ee9988>]
```



## 2.5 LXML和家族树

- 示例：查看table元素的兄弟子元素

```
.....  
print('-' * 40)  
prev_sibling = table.getprevious() # 前一个兄弟元素  
pprint(prev_sibling)  
  
print('-' * 40)  
next_sibling = table.getnext() # 后一个兄弟元素  
pprint(next_sibling)
```

```
-----  
None  
-----
```

```
<Element div at 0x32c7b90>
```



## 2.5 LXML和家族树

- 示例：查看table元素的父元素以及父元素的所有子元素

```
print('-' * 40)
parent = table.getparent() # 父元素
pprint(parent)
print('-' * 40)
pprint(parent.getchildren()) # 父元素form的所有子元素
```

```
-----
<Element form at 0x32c7be0>
```

```
-----
[<Element table at 0x2f421b8>, <Element div at 0x2cfe938>]
```

- 如果需要更加通用的方式来访问页面中的所有元素，则结合XPath表达式遍历家族关系是一个能让你不丢失任何内容的好方式。
- 可以通过识别页面中那些元素附近的内容，来识别页面中某些重要内容。即使该元素没有可识别的CSS选择器，该方法同样也可以工作。



## 2.6 三种抓取方法的性能对比

- 要想更好的对前面介绍的这3种抓取方法评估取舍，需要对其相对效率进行对比。
- 一般情况下，爬虫会抽取网页中的多个字段。因此，为了让对比更加真实，将为本章中的每个爬虫都实现一个扩展版本，用于抽取国家（或地区）网页中的每个可用数据。
- 首先，需要回到浏览器中，检查国家（或地区）页面其他特征的格式。通过使用浏览器的查看功能，可以看到表格的每一行都拥有一个以places\_开头\_\_row结尾的ID。而在这些行中包含的国家（或地区）数据，其格式都和面积示例相同。





## 2.6 三种抓取方法的性能对比

Elements Console Sources Network Performance

```
▼ <table>
  ▼ <tbody>
    ▶ <tr id="places_flag_img__row">...</tr>
    ▶ <tr id="places_area__row">...</tr>
    ▶ <tr id="places_population__row">...</tr>
    ▶ <tr id="places_iso__row">...</tr>
    ▶ <tr id="places_country_or_district__row">...</tr>
    ▶ <tr id="places_capital__row">...</tr>
    ▶ <tr id="places_continent__row">...</tr>
    ▶ <tr id="places_tld__row">...</tr>
    ▶ <tr id="places_currency_code__row">...</tr>
    ▶ <tr id="places_currency_name__row">...</tr>
    ▶ <tr id="places_phone__row">...</tr>
    ▶ <tr id="places_postal_code_format__row">...</tr>
    ▶ <tr id="places_postal_code_regex__row">...</tr>
    ▶ <tr id="places_languages__row">...</tr>
    ▶ <tr id="places_neighbours__row">...</tr>
  </tbody>
</table>
```

行id均以places\_  
开头和\_\_row结尾



## 2.6 三种抓取方法的性能对比

Elements Console Sources Network Performance Memory

```
▼ <table>
  ▼ <tbody>
    ▶ <tr id="places_flag_img__row">...</tr>
    ▼ <tr id="places_area__row">
      ▶ <td class="w2p_fl">...</td>
      <td class="w2p_fw">244,820 square kilometres</td>
      <td class="w2p_fc"></td>
    </tr>
    ▼ <tr id="places_population__row">
      ▶ <td class="w2p_fl">...</td>
      <td class="w2p_fw">62,348,447</td>
      <td class="w2p_fc"></td>
    </tr>
    ▶ <tr id="places_iso__row">...</tr>
    ▶ <tr id="places_country_or_district__row">...</tr>
    ▶ <tr id="places_capital__row">...</tr>
    ▶ <tr id="places_continent__row">...</tr>
    ▶ <tr id="places_tld__row">...</tr>
```

各行中国家（或地区）  
属性数据列的格式相同



## 2.6 三种抓取方法的性能对比

- 抓取所有国家(或地区)数据的实现代码，包含三种方法。

第2章

数据抓取

```
# ----- all_scrapers.py -----
```

```
FIELDS = ('area',      # 定义要抓取的各列的标识
```

```
    'population',
    'iso',
    'country_or_district',
    'capital',
    'continent',
    'tld',
    'currency_code',
    'currency_name',
    'phone',
    'postal_code_format',
    'postal_code_regex',
    'languages',
    'neighbours')
```

```
<table>
  <tbody>
    <tr id="places_flag_img__row">...</tr>
    <tr id="places_area__row">
      <td class="w2p_fl">...</td>
      <td class="w2p_fw">244,820 square kilometr</td>
      <td class="w2p_fc"></td>
    </tr>
    <tr id="places_population__row">
      <td class="w2p_fl">...</td>
      <td class="w2p_fw">62,348,447</td>
      <td class="w2p_fc"></td>
    </tr>
    <tr id="places_iso__row">...</tr>
    <tr id="places_country_or_district__row">...</tr>
    <tr id="places_capital__row">...</tr>
    <tr id="places_continent__row">...</tr>
    <tr id="places_tld__row">...</tr>
    <tr id="places_currency_code__row">...</tr>
```



## 2.6 三种抓取方法的性能对比

```
# ----- all_scrapers.py -----  
  
.....  
import re  
  
def re_scraper(html):  
    results = {} # 存放抓取结果  
    for field in FIELDS:  
        regex = '<tr id="places_%s__row">.*?<td  
                class="w2p_fw">(.*?)</td>' % field  
        mo = re.search(regex, html)  
        results[field] = mo.group(1)  
    return results
```



## 2.6 三种抓取方法的性能对比

```
# ----- all_scrapers.py -----  
.....  
from bs4 import BeautifulSoup  
  
def bs_scraper(html):  
    soup = BeautifulSoup(html, features='html.parser')  
    table = soup.find(name='table')  
    results = {}  
    for field in FIELDS:  
        tr = table.find(name='tr', attrs={'id': 'places_%s__row' % field})  
        td = tr.find(name='td', attrs={'class': 'w2p_fw'})  
        results[field] = td.text  
    return results
```



## 2.6 三种抓取方法的性能对比

```
# ----- all_scrapers.py -----  
.....  
from lxml.html import fromstring  
  
def lxml_scraper(html):  
    tree = fromstring(html)  
    results = {}  
    for field in FIELDS:  
        selector = 'table > tr#places_%s__row > td.w2p_fw' % field  
        td = tree.cssselect(selector)[0]  
        results[field] = td.text_content()  
    return results
```

- 选择元素的目前做法：每次先从整个html文档中找到表，然后找到行和列
- 请改进：先选择table，然后再循环从table中选择行和列。



## 2.6 三种抓取方法的性能对比

```
# ----- all_scrapers.py -----  
.....  
def lxml_xpath_scraper(html):  
    tree = fromstring(html)  
    results = {}  
    for field in FIELDS:  
        selector = '//table/tr[@id="places_%s__row"]'  
                    '/td[@class="w2p_fw"]' % field  
        td = tree.xpath(selector)[0]  
        results[field] = td.text_content()  
    return results
```

- 选择元素的目前做法：每次先从整个html文档中找到表，然后找到行和列
- 请改进：先选择table，然后再循环从table中选择行和列。



## 2.7 三种方法的抓取结果

- 现在，已经完成了所有爬虫代码的实现，接下来将测试3种方法的相对性能。
- 具体做法是
  - 首先，从示例网站下载一个(仍然以英国为例)页面；
  - 然后，让每个爬虫都从这个下载后的页面(即html源代码)中执行1000次抓取，每次执行都会检查抓取结果是否正确；
  - 最后，打印各爬虫抓取的总用时。
- 另外，默认情况下，正则表达式模块会缓存搜索结果，为了使其与其他爬虫的对比更加公平，每次抓取前需要使用`re.purge()`方法清除缓存。





## 2.7 三种方法的抓取结果

```
...
import re
import time
from all_scrapers import re_scraper, bs_scraper, lxml_scraper,
lxml_xpath_scraper

NUM_ITERATIONS = 1000 # number of times to test each scraper
url = 'http://example.python-scraping.com/places/
      default/view/United-Kingdom-233'

html = download(url)

scrapers = [('Regular expressions', re_scraper),
            ('BeautifulSoup', bs_scraper),
            ('Lxml', lxml_scraper),
            ('XPath', lxml_xpath_scraper)]
```



## 2.7 三种方法的抓取结果

```
for name, scraper in scrapers:  
    # record start time of scrape  
    start = time.time()
```

```
    for i in range(NUM_ITERATIONS):
```

```
        if scraper == re_scraper:  
            re.purge()
```

```
            result = scraper(html)
```

```
            # check scraped result is as expected
```

```
            assert result['area'] == '244,820 square kilometres', '数据错误!'
```

```
    # record end time of scrape and output the total
```

```
    end = time.time()
```

```
    print('%s: %.2f seconds' % (name, end - start))
```

若不满足，则抛出  
AssertionError异常：  
**AssertionError: 数据错误!**



## 2.7 三种方法的抓取结果

- 某次运行结果:

Regular expressions: 2.05 seconds

BeautifulSoup: 9.14 seconds

Lxml: 2.63 seconds

XPath: 1.12 seconds

- 由于硬件条件的区别，不同的计算机的执行结果也会存在一定差异。不过，每种方法之间的相对差异应当是相似的。
- 从结果中可以看出，在抓取示例网页时，BeautifulSoup的速度大约是其他方法的1/4。
- 实际上这一结果是符合预期的，因为lxml和正则表达式模块都是用C语言编写的，而BeautifulSoup则是纯Python编写的。



## 2.7 三种方法的抓取结果

### ● 抓取总结

抓取方法	性能	使用难度	安装难度
正则表达式	快	困难	简单（内置模块）
Beautiful Soup	慢	简单	简单（纯Python）
Lxml	快	简单	相对困难

- 通常情况下，lxml是抓取数据的最佳选择，这是因为该方法既快速又健壮，而正则表达式和Beautiful Soup或是修改不易，或是速度不快。



## 2.8 为链接爬虫添加抓取回调

- 为链接爬虫添加抓取回调 (callback) , 也就是将数据抓取 (srcaping) 功能集成到链接爬虫(linkCrawler)当中。
- 将数据抓取的功能定义为scrape\_callback(url,html),
- 在链接爬虫中当某个网页下载完成后就调用该函数来抓取数据。
- 下面创建一个高级链接爬虫, 其中国家 (或地区) 网页下载后抓取数据, 脚本文件名称 advanced\_link\_crawler.py,模块结构如下:



## 2.8 为链接爬虫添加抓取回调

```
import urllib.request as request
from http.client import *
import re
import time
from urllib.parse import urljoin
from lxml.html import fromstring

FIELDS = (...)

def scrape_callback(url, html):...

def link_crawler(start_url, link_regex, scrape_callback=None):...

def get_links(html):...

def download(url: str, user_agent='wswp', num_retries=2, charset='UTF-8'):...
```

- 本代码是在第一章中的link\_crawler.py的基础之上加入而爬取函数scrape\_callback(), 并修改link\_crawler()函数, 网页下载完成后就调用scrape\_callback()函数抓取数据。



## 2.8 为链接爬虫添加抓取回调

### ● 修改link\_crawler()函数

```
def link_crawler(start_url, link_regex, scrape_callback=None):  
    """ Crawl from given start URL following links matched by link_regex """  
    crawl_queue = [start_url]  
    # keep track which URL's have been before  
    seen = set(crawl_queue)  
    while crawl_queue:  
        url = crawl_queue.pop() # 弹出队列首元素（链接）  
        html = download(url=url) # 下载页面  
        if html is None:  
            continue  
        # -----抓取网页数据-----  
        if scrape_callback is not None:  
            scrape_callback(url, html)  
  
        # filter for links matching our regular expression  
        for link in get_links(html):  
            .....
```



## 2.8 为链接爬虫添加抓取回调

- 添加抓取回调函数 `scrape_callback()`

```
def scrape_callback(url, html):  
    if re.search(pattern='/view/', string=url):  
        tree = fromstring(html)  
        all_row = []  
        for field in FIELDS:  
            selector = '//table/tr[@id="places_%s__row"]  
                        /td[@class="w2p_fw"]' % field  
            td = tree.xpath(selector)[0]  
            all_row.append(td.text_content())  
  
    print(url, all_row) # 这里抓取数据后仅输出
```

- `re.search(pattern='/view/', string=url)` 用于匹配含有 `/view/` 的url。
- 国家（或地区）的详细信息所在页面的url路径中含有 `/view/`，例如：`http://example.python-scraping.com/places/default/view/United-Kingdom-233`





## 2.8 为链接爬虫添加抓取回调

- 爬取示例网站并抓取数据的测试脚本如下：

```
from advanced_link_crawler import link_crawler, scrape_callback

url = 'http://example.python-scraping.com'
regex = '/places/default/(index|view)/'
link_crawler(url, regex, scrape_callback=scrape_callback)
```

Downloading: http://example.python-scraping.com

Downloading: http://example.python-scraping.com/places/default/index/1

.....

Downloading: http://example.python-scraping.com/places/default/index/24

Downloading: http://example.python-scraping.com/places/default/view/Zimbabwe-246

http://example.python-scraping.com/places/default/view/Zimbabwe-246 ['390,580 square kilometres', '11,651,858', 'ZW', 'Zimbabwe', 'Harare', 'AF', '.zw', 'ZWL', 'Dollar', '263', '', '', 'en-ZW,sn,nr,nd', 'ZA MZ BW ZM ']

.....

Downloading: http://example.python-scraping.com/places/default/view/Afghanistan-1

http://example.python-scraping.com/places/default/view/Afghanistan-1 ['647,500 square kilometres', '29,121,286', 'AF', 'Afghanistan', 'Kabul', 'AS', '.af', 'AFN', 'Afghani', '93', '', '', 'fa-AF,ps,uz-AF,tk', 'TM IR TJ PK UZ ']



## 2.9 保存链接爬虫抓取的数据

- 通常，当抓取网站时，更希望复用得到的数据，而不仅仅是打印出来。
- 因此将对上一个示例进行扩展，将爬虫抓取的数据保存到CSV电子表格当中。
- 这里采用**csv模块**实现将数据写入csv文件。
- 此处将数据抓取功能封装到CsvCallback类中，而不再使用回调函数scrape\_callback()，以便保持写文件时的csv中的writer状态。csv的writer属性在构造方法中进行了实例化处理（创建csv文件并写入有field列表定义的表头信息），然后在\_\_call\_\_()中写入抓取的数据。

```
# ----- csv_allback.py -----
```

```
import csv, re
from lxml.html import fromstring
class CsvCallback:
```

```
    def __init__(self, fields, csvFilePath):
        self.fp = open(csvFilePath, 'tw', encoding='utf-8', newline=")
        self.writer = csv.writer(self.fp)
        self.fields = fields
        self.writer.writerow(self.fields) # 写入表头
        self.fp.flush() # 立即将文件流在缓存区中的数据写入文件
```

```
    def __call__(self, url, html):
        """__call__是一个特殊方法，在对象作为函数被调用时会自动调用该方法。也就是说，csvCallback(url,html)和调用csvCallback.__call__(url,html)等价。"""
```

```
        if re.search(pattern='/view/', string=url):
            tree = fromstring(html)
            all_row = []
            for field in self.fields:
                selector = '//table/tr[@id="places_%s__row"]/td[@class="w2p_fw"]' % field
                td = tree.xpath(selector)[0]
                all_row.append(td.text_content())
            self.writer.writerow(all_row) # 抓取的数据写入CSV文件
            self.fp.flush() # 立即将文件流在缓存区中的数据写入文件
        def close(self):
            self.fp.close()
```

注意在open()内增加一个参数  
**newline=""**，否则写入时，每一行  
数据后面都会自动增加一个空行。



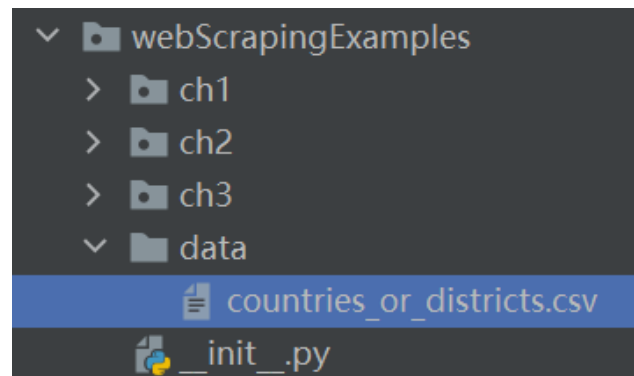
## 2.9 保存链接爬虫抓取的数据

- 爬取示例网站的测试代码如下：

```
from advanced_link_crawler import FIELDS, link_crawler
from cvs_callback import CsvCallback

url = 'http://example.python-scraping.com'
regex = '/places/default/(index|view)/'
csvFilePath = '../data/countries_or_districts.csv'
csvCallback = CsvCallback(fields=FIELDS, csvFilePath=csvFilePath)
link_crawler(url, regex, scrape_callback=csvCallback)
csvCallback.close()
```

注意文件目录结构，  
请先建立data文件夹。





## 2.9 保存链接爬虫抓取的数据

- 运行程序，爬取的数据保存到了csv文件。Perfect!

countries\_or\_districts.csv - Excel Ruan Zongli RZ

文件 开始 插入 页面布局 公式 数据 审阅 视图 帮助 团队 操作说明搜索 共享

A21 430 square kilometres

	A	B	C	D	E	F	G	H	I	J
1	area	population	iso	country_or_district	capital	continent	tld	currency_code	currency_name	phone
2	390,580 square kilometres	11,651,858	ZW	Zimbabwe	Harare	AF	.zw	ZWL	Dollar	26
3	752,614 square kilometres	13,460,305	ZM	Zambia	Lusaka	AF	.zm	ZMW	Kwacha	26
4	527,970 square kilometres	23,495,361	YE	Yemen	Sanaa	AS	.ye	YER	Rial	96
5	266,000 square kilometres	273,008	EH	Western Sahara	El-Aaiun	AF	.eh	MAD	Dirham	21
6	274 square kilometres	16,025	WF	Wallis and Futuna	Mata Utu	OC	.wf	XPF	Franc	68
7	329,560 square kilometres	89,571,130	VN	Vietnam	Hanoi	AS	.vn	VND	Dong	8
8	912,050 square kilometres	27,223,228	VE	Venezuela	Caracas	SA	.ve	VEF	Bolivar	5
9	0 square kilometres	921	VA	Vatican	Vatican City	EU	.va	EUR	Euro	37
10	12,200 square kilometres	221,552	VU	Vanuatu	Port Vila	OC	.vu	VUV	Vatu	67
11	447,400 square kilometres	27,865,738	UZ	Uzbekistan	Tashkent	AS	.uz	UZS	Som	99
12	176,220 square kilometres	3,477,000	UY	Uruguay	Montevideo	SA	.uy	UYU	Peso	59
13	0 square kilometres	0	UM	United States Minor Outlying Islands		OC	.um	USD	Dollar	
14	9,629,091 square kilometres	310,232,863	US	United States	Washington	NA	.us	USD	Dollar	
15	244,820 square kilometres	62,348,447	GB	United Kingdom	London	EU	.uk	GBP	Pound	4
16	82,880 square kilometres	4,975,593	AE	United Arab Emirates	Abu Dhabi	AS	.ae	AED	Dirham	97
17	603,700 square kilometres	45,415,596	UA	Ukraine	Kiev	EU	.ua	UAH	Hryvnia	38

countries\_or\_districts

就绪 平均值: 9954 计数: 14 求和: 19908 100%



## 小结

- 本章介绍了几种抓取网页数据的方法。正则表达式在一次性数据抓取中非常有用，此外还可以避免解析整个网页带来的开销；Beautiful Soup提供了更高层次的接口，同时还能避免过多麻烦的依赖；不过，通常情况下，lxml是的最佳选择，因为它速度快，功能更加丰富
- 还学习了使用浏览器工具和控制台查看HTML页面（源代码和元素），以及定义CSS选择器和XPath选择器来匹配和抽取已下载页面中的内容。
- 在下一章中，将介绍缓存技术，这样就能把网页保存下来，只在爬虫第一次运行时才会下载网页。



谢谢大家!