



# Python 爬虫利器二之 BeautifulSoup 的用法

吕 崔庆才 | 2015-03-11 | Python | 68968 | 20k | 18 分钟

上一节我们介绍了正则表达式，它的内容其实还是蛮多的，如果一个正则匹配稍有差池，那可能程序就处在永久的循环之中，而且有的小伙伴们也对写正则表达式的写法用得不熟练，没关系，我们还有一个更强大的工具，叫 BeautifulSoup，有了它我们可以很方便地提取出 HTML 或 XML 标签中的内容，实在是方便，这一节就让我们一起来感受一下 BeautifulSoup 的强大吧。

## 1. BeautifulSoup 的简介

简单来说，Beautiful Soup 是 python 的一个库，最主要的功能是从网页抓取数据。官方解释如下：

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。它是一个工具箱，通过解析文档为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序。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。你不需要考虑编码方式，除非文档没有指定一个编码方式，这时，Beautiful Soup 就不能自动识别编码方式了。然后，你仅仅需要说明一下原始编码方式就可以了。Beautiful Soup 已成为和 lxml、html6lib 一样出色的 python 解释器，为用户灵活地提供不同的解析策略或强劲的速度。

废话不多说，我们来试一下吧～

## 2. BeautifulSoup 安装

Beautiful Soup 3 目前已经停止开发，推荐在现在的项目中使用 BeautifulSoup 4，不过它已经被移植到 BS4 了，也就是说导入时我们需要 import bs4。所以这里我们用的版本是 BeautifulSoup 4.3.2 (简称 BS4)，另外据说 BS4 对 Python3 的支持不够好，不过我用的是 Python2.7.7，如果有小伙伴用的是 Python3 版本，可以考虑下载 BS3 版本。可以利用 pip 或者 easy\_install 来安装，以下两种方法均可

```
1 easy_install beautifulsoup4
```



```
1 pip install beautifulsoup4
```

↑ 0%

如果想安装最新的版本，请直接下载安装包来手动安装，也是十分方便的方法。在这里我安装的是 Beautiful Soup 4.3.2 Beautiful Soup 3.2.1Beautiful Soup 4.3.2 下载完成之后解压 运行下面的命令即可完成安装



```
1 sudo python setup.py install
```

然后需要安装 lxml



```
1 easy_install lxml
```



```
1 pip install lxml
```

另一个可供选择的解析器是纯 Python 实现的 html5lib，html5lib 的解析方式与浏览器相同，可以选择下列方法来安装 html5lib:



```
1 easy_install html5lib
```



```
1 pip install html5lib
```

Beautiful Soup 支持 Python 标准库中的 HTML 解析器，还支持一些第三方的解析器，如果我们不安装它，则 Python 会使用 Python 默认的解析器，lxml 解析器更加强大，速度更快，推荐安装。

<thead">

解析器

使用方法

优势

劣势

Python 标准库

BeautifulSoup(markup, "html.parser")

- Python 的内置标准库
- 执行速度适中
- 文档容错能力强

↑ 0%

- Python 2.7.3 or 3.2.2) 前 的版本中文档容错能力差

lxml HTML 解析器

BeautifulSoup(markup, "lxml")

- 速度快
- 文档容错能力强
- 需要安装 C 语言库

lxml XML 解析器

BeautifulSoup(markup, ["lxml", "xml"])BeautifulSoup(markup, "xml")

- 速度快
- 唯一支持 XML 的解析器
- 需要安装 C 语言库

html5lib

BeautifulSoup(markup, "html5lib")

- 最好的容错性
- 以浏览器的方式解析文档
- 生成 HTML5 格式的文档
- 速度慢
- 不依赖外部扩展

### 3. 开启 BeautifulSoup 之旅

在这里先分享官方文档链接，不过内容是有些多，也不够条理，在此本文章做一下整理方便大家参考。 [官方文档](#)

### 4. 创建 BeautifulSoup 对象

首先必须要导入 bs4 库

```
1 from bs4 import BeautifulSoup
```

↑10% 我们创建一个字符串，后面的例子我们便会用它来演示

```
1 html = """
2 <html><head><title>The Dormouse's story</title></head>
3 <body>
4 <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
5 <p class="story">Once upon a time there were three little sisters; and t
6 <a href="http://example.com/elsie" class="sister" id="link1"><!-- Elsie
7 <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> a
8 <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>
9 and they lived at the bottom of a well.</p>
10 <p class="story">...</p>
11 """
```

创建 beautifulsoup 对象

```
1 soup = BeautifulSoup(html)
```

另外，我们还可以用本地 HTML 文件来创建对象，例如

```
1 soup = BeautifulSoup(open('index.html'))
```

上面这句代码便是将本地 index.html 文件打开，用它来创建 soup 对象 下面我们来打印一下 soup 对象的内容，格式化输出

```
1 print soup.prettify()
```

```
1 <html>
2 <head>
3 <title>
4   The Dormouse's story
5 </title>
6 </head>
7 <body>
```

```
8     <p class="title" name="dromouse">
9         <b>
10             The Dormouse's story
11         </b>
12     </p>
13     <p class="story">
14         Once upon a time there were three little sisters; and their names were
15         <a class="sister" href="http://example.com/elsie" id="link1">
16             <!-- Elsie -->
17         </a>
18         ,
19         <a class="sister" href="http://example.com/lacie" id="link2">
20             Lacie
21         </a>
22         and
23         <a class="sister" href="http://example.com/tillie" id="link3">
24             Tillie
25         </a>
26         ;
27         and they lived at the bottom of a well.
28     </p>
29     <p class="story">
30         ...
31     </p>
32 </body>
33 </html>
```

以上便是输出结果，格式化打印出了它的内容，这个函数经常用到，小伙伴们要记好咯。

## 5. 四大对象种类


Beautiful Soup 将复杂 HTML 文档转换成一个复杂的树形结构，每个节点都是 Python 对象，所有对象可以归纳为 4 种：

- Tag
- NavigableString
- BeautifulSoup
- Comment

下面我们进行一一介绍

### (1) Tag

Tag 是什么？通俗点讲就是 HTML 中的一个标签，例如



```
1 <title>The Dormouse's story</title>
```

↑ 0%



```
1 <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
```

上面的 title a 等等 HTML 标签加上里面包括的内容就是 Tag，下面我们来感受一下怎样用 BeautifulSoup 来方便地获取 Tags 下面每一段代码中注释部分即为运行结果



```
1 print soup.title
2 #<title>The Dormouse's story</title>
```



```
1 print soup.head
2 #<head><title>The Dormouse's story</title></head>
```




```
1 print soup.a
2 #<a class="sister" href="http://example.com/elsie" id="link1">Elsie
```




```
1 print soup.p
2 #<p class="title" name="dromouse"><b>The Dormouse's story</b></p>
```

我们可以利用 soup 加标签名轻松地获取这些标签的内容，是不是感觉比正则表达式方便多了？不过有一点是，它查找的是在所有内容中的第一个符合要求的标签，如果要查询所有的标签，我们在后面进行介绍。我们可以验证一下这些对象的类型



```
1 print type(soup.a)
2 #<class 'bs4.element.Tag'>
```

对于 Tag，它有两个重要的属性，是 name 和 attrs，下面我们分别来感受一下 name



```
1 print soup.name
2 print soup.head.name
```

```
3  #[document]
4  #head
```

↑soup.p 对象本身比较特殊，它的 name 即为 [document]，对于其他内部标签，输出的值便为标签本身的名称。attrs

```
1  print soup.p.attrs
2  #{'class': ['title'], 'name': 'dromouse'}
```

在这里，我们把 p 标签的所有属性打印输出出来了，得到的类型是一个字典。如果我们想要单独获取某个属性，可以这样，例如我们获取它的 class 叫什么

```
1  print soup.p['class']
2  #['title']
```

还可以这样，利用 get 方法，传入属性的名称，二者是等价的

```
1  print soup.p.get('class')
2  #['title']
```

我们可以对这些属性和内容等等进行修改，例如

```
1  soup.p['class']="newClass"
2  print soup.p
3  #<p class="newClass" name="dromouse"><b>The Dormouse's story</b></p>
```

还可以对这个属性进行删除，例如

```
1  del soup.p['class']
2  print soup.p
3  #<p name="dromouse"><b>The Dormouse's story</b></p>
```

不过，对于修改删除的操作，不是我们的主要用途，在此不做详细介绍了，如果有需要，请查看前面提供的官方文档

## (2) NavigableString

既然我们已经得到了标签的内容，那么问题来了，我们要想获取标签内部的文字怎么办呢？很简单，用 `.string` 即可，例如

```
↑ 0%  
1 print soup.p.string  
2 #The Dormouse's story
```

这样我们就轻松获取到了标签里面的内容，想想如果用正则表达式要多麻烦。它的类型是一个 `NavigableString`，翻译过来叫 可以遍历的字符串，不过我们最好还是称它英文名字吧。来检查一下它的类型

```
1 print type(soup.p.string)  
2 #<class 'bs4.element.NavigableString'>
```

### (3) BeautifulSoup

BeautifulSoup 对象表示的是一个文档的全部内容。大部分时候，可以把它当作 Tag 对象，是一个特殊的 Tag，我们可以分别获取它的类型，名称，以及属性来感受一下

```
1 print type(soup.name)  
2 #<type 'unicode'>  
3 print soup.name  
4 # [document]  
5 print soup.attrs  
6 #{} 空字典
```

### (4) Comment

Comment 对象是一个特殊类型的 `NavigableString` 对象，其实输出的内容仍然不包括注释符号，但是如果不好好处理它，可能会对我们的文本处理造成意想不到的麻烦。我们找一个带注释的标签

```
1 print soup.a  
2 print soup.a.string  
3 print type(soup.a.string)
```

运行结果如下

```
1  
2
```



```

1 <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -
2 Elsie
3 <class 'bs4.element.Comment'>

```

↑ 0%

a 标签里的内容实际上是注释，但是如果我们利用 `.string` 来输出它的内容，我们发现它已经把注释符号去掉了，所以这可能会给我们带来不必要的麻烦。另外我们打印输出下它的类型，发现它是一个 `Comment` 类型，所以，我们在使用前最好做一下判断，判断代码如下

```

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

```

上面的代码中，我们首先判断了它的类型，是否为 `Comment` 类型，然后再进行其他操作，如打印输出。

## 6. 遍历文档树

### (1) 直接子节点

要点：`.contents` `.children` 属性

`.contents` tag 的 `.content` 属性可以将 tag 的子节点以列表的方式输出

```

1 print soup.head.contents
2 #[<title>The Dormouse's story</title>]

```

输出方式为列表，我们可以用列表索引来获取它的某一个元素

```

1 print soup.head.contents[0]
2 #<title>The Dormouse's story</title>

```

`.children` 它返回的不是一个 `list`，不过我们可以通过遍历获取所有子节点。我们打印输出 `.children` 看一下，可以发现它是一个 `list` 生成器对象

```

1 print soup.head.children
2 #<listiterator object at 0x7f71457f5710>

```

我们怎样获得里面的内容呢？很简单，遍历一下就好了，代码及结果如下

```

1 for child in soup.body.children:
2     print child

```

↑ 0%

```

1 <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
2
3 <p class="story">Once upon a time there were three little sisters; and t
4 <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
5 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> a
6 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
7 and they lived at the bottom of a well.</p>
8
9
10 <p class="story">...</p>

```

## (2) 所有子孙节点

知识点: `.descendants` 属性

`.descendants` `.contents` 和 `.children` 属性仅包含 tag 的直接子节点, `.descendants` 属性可以对所有 tag 的子孙节点进行递归循环, 和 `children` 类似, 我们也需要遍历获取其中的内容。

```

1 for child in soup.descendants:
2     print child

```

运行结果如下, 可以发现, 所有的节点都被打印出来了, 先生最外层的 HTML 标签, 其次从 head 标签一个个剥离, 以此类推。

```

1 <html><head><title>The Dormouse's story</title></head>
2 <body>
3 <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
4 <p class="story">Once upon a time there were three little sisters; and t
5 <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
6 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> a
7 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
8 and they lived at the bottom of a well.</p>
9 <p class="story">...</p>
10 </body></html>
11 <head><title>The Dormouse's story</title></head>
12 <title>The Dormouse's story</title>

```

```
13 The Dormouse's story
14
15
16 <body>
17 <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
18 <p class="story">Once upon a time there were three little sisters; and t
19 <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
20 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> a
21 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
22 and they lived at the bottom of a well.</p>
23 <p class="story">...</p>
24 </body>
25
26
27 <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
28 <b>The Dormouse's story</b>
29 The Dormouse's story
30
31
32 <p class="story">Once upon a time there were three little sisters; and t
33 <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
34 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> a
35 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
36 and they lived at the bottom of a well.</p>
37 Once upon a time there were three little sisters; and their names were
38
39 <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
40 Elsie
41 ,
42
43 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
44 Lacie
45 and
46
47 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
48 Tillie
49 ;
50 and they lived at the bottom of a well.
51
52
53 <p class="story">...</p>
54 ...
```

### (3) 节点内容

知识点: .string 属性

如果 tag 只有一个 NavigableString 类型子节点，那么这个 tag 可以使用 .string 得到子节点。如果一个 tag 仅有一个子节点，那么这个 tag 也可以使用 .string 方法，输出结果与当前唯一子节点的 .string 结果相同。通俗点说就是：如果一个标签里面没有标签了，那么 .string 就会返回标签里面的内容。如果标签里面只有唯一的一个标签了，那么 .string 也会返回最里面的内容。例如

```
1 print soup.head.string
2 #The Dormouse's story
3 print soup.title.string
4 #The Dormouse's story
```

如果 tag 包含了多个子节点，tag 就无法确定，string 方法应该调用哪个子节点的内容，.string 的输出结果是 None

```
1 print soup.html.string
2 # None
```

## (4) 多个内容

知识点：.strings .stripped\_strings 属性

.strings 获取多个内容，不过需要遍历获取，比如下面的例子

```
1 for string in soup.strings:
2     print(repr(string))
3     # u"The Dormouse's story"
4     # u'\n\n'
5     # u"The Dormouse's story"
6     # u'\n\n'
7     # u'Once upon a time there were three little sisters; and their name
8     # u'Elsie'
9     # u',\n'
10    # u'Lacie'
11    # u' and\n'
12    # u'Tillie'
13    # u';\nand they lived at the bottom of a well.'
14    # u'\n\n'
15    # u'...'
16    # u'\n'
```

`.stripped_strings` 输出的字符串中可能包含了很多空格或空行，使用 `.stripped_strings` 可以去除多余空白内容

↑ 0%

```
1 for string in soup.stripped_strings:
2     print(repr(string))
3     # u"The Dormouse's story"
4     # u"The Dormouse's story"
5     # u'Once upon a time there were three little sisters; and their name
6     # u'Elsie'
7     # u','
8     # u'Lacie'
9     # u'and'
10    # u'Tillie'
11    # u';\nand they lived at the bottom of a well.'
12    # u'...'
```

## (5) 父节点

知识点: `.parent` 属性

```
1 p = soup.p
2 print p.parent.name
3 #body
```

```
1 content = soup.head.title.string
2 print content.parent.name
3 #title
```

## (6) 全部父节点

知识点: `.parents` 属性

通过元素的 `.parents` 属性可以递归得到元素的所有父辈节点，例如

```
1 content = soup.head.title.string
2 for parent in content.parents:
3     print parent.name
```

```

1  title
2  head
3  html
↑ 0%
4  [document]

```

## (7) 兄弟节点

知识点: `.next_sibling` `.previous_sibling` 属性

兄弟节点可以理解为和本节点处在统一级的节点, `.next_sibling` 属性获取了该节点的下一个兄弟节点, `.previous_sibling` 则与之相反, 如果节点不存在, 则返回 `None` 注意: 实际文档中的 tag 的 `.next_sibling` 和 `.previous_sibling` 属性通常是字符串或空白, 因为空白或者换行也可以被视作一个节点, 所以得到的结果可能是空白或者换行

```

1  print soup.p.next_sibling
2  #      实际该处为空白
3  print soup.p.prev_sibling
4  #None   没有前一个兄弟节点, 返回 None
5  print soup.p.next_sibling.next_sibling
6  #<p class="story">Once upon a time there were three little sisters; and
7  #<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
8  #<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
9  #<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
10 #and they lived at the bottom of a well.</p>
11 #下一个节点的下一个兄弟节点是我们可以看到的节点

```

## (8) 全部兄弟节点

知识点: `.next_siblings` `.previous_siblings` 属性

通过 `.next_siblings` 和 `.previous_siblings` 属性可以对当前节点的兄弟节点迭代输出

```

1  for sibling in soup.a.next_siblings:
2      print(repr(sibling))
3      # u',\n'
4      # <a class="sister" href="http://example.com/lacie" id="link2">Lacie<
5      # u' and\n'
6      # <a class="sister" href="http://example.com/tillie" id="link3">Tilli
7      # u'; and they lived at the bottom of a well.'
8      # None

```

## (9) 前后节点

知识点: `.next_element` `.previous_element` 属性

↑ 0%

与 `.next_sibling` `.previous_sibling` 不同, 它并不是针对于兄弟节点, 而是在所有节点, 不分层次  
比如 `head` 节点为

```
1 <head><title>The Dormouse's story</title></head>
```

那么它的下一个节点便是 `title`, 它是不分层次关系的

```
1 print soup.head.next_element
2 #<title>The Dormouse's story</title>
```

## (10) 所有前后节点

知识点: `.next_elements` `.previous_elements` 属性

通过 `.next_elements` 和 `.previous_elements` 的迭代器就可以向前或向后访问文档的解析内容, 就好像文档正在被解析一样

```
1 for element in last_a_tag.next_elements:
2     print(repr(element))
3 # u'Tillie'
4 # u';\nand they lived at the bottom of a well.'
5 # u'\n\n'
6 # <p class="story">...</p>
7 # u'...'
8 # u'\n'
9 # None
```

以上是遍历文档树的基本用法。

## 7. 搜索文档树

### (1) `find_all( name , attrs , recursive , text , **kwargs )`

`find_all()` 方法搜索当前 `tag` 的所有 `tag` 子节点, 并判断是否符合过滤器的条件 1) **name** 参数  
`name` 参数可以查找所有名字为 `name` 的 `tag`, 字符串对象会被自动忽略掉 **A. 传字符串** 最简单的过

过滤器是字符串。在搜索方法中传入一个字符串参数，Beautiful Soup 会查找与字符串完整匹配的内容，下面的例子用于查找文档中所有的**标签**

↑ 0%

```
1 soup.find_all('b')
2 # [<b>The Dormouse's story</b>]
```

```
1 print soup.find_all('a')
2 # [<a class="sister" href="http://example.com/elsie" id="link1">!-- Elsie
```

**B. 传正则表达式** 如果传入正则表达式作为参数，Beautiful Soup 会通过正则表达式的 `match()` 来匹配内容。下面例子中找出所有以 `b` 开头的标签，这表示 `<b>` 和 `<body>` 都应该被找到

```
1 import re
2 for tag in soup.find_all(re.compile("^b")):
3     print(tag.name)
4 # body
5 # b
```

**C. 传列表** 如果传入列表参数，Beautiful Soup 会将与列表中任一元素匹配的内容返回。下面代码找到文档中所有**标签和标签**

```
1 soup.find_all(["a", "b"])
2 # [<b>The Dormouse's story</b>,
3 # <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
4 # <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
5 # <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

**D. 传 True** True 可以匹配任何值，下面代码查找到所有的 `tag`，但是不会返回字符串节点

```
1 for tag in soup.find_all(True):
2     print(tag.name)
3 # html
4 # head
5 # title
6 # body
7 # p
8 # b
9 # p
```



```
10 # a
11 # a
```

**自定义方法** 如果没有合适过滤器，那么还可以定义一个方法，方法只接受一个元素参数 [4]，如果这个方法返回 True 表示当前元素匹配并且被找到，如果不是则返回 False 下面方法校验了当前元素，如果包含 class 属性却不包含 id 属性，那么将返回 True:

```
1 def has_class_but_no_id(tag):
2     return tag.has_attr('class') and not tag.has_attr('id')
```

将这个方法作为参数传入 find\_all () 方法，将得到所有

标签:

```
1 soup.find_all(has_class_but_no_id)
2 # [<p class="title"><b>The Dormouse's story</b></p>,
3 # <p class="story">Once upon a time there were...</p>,
4 # <p class="story">...</p>]
```

## 2) keyword 参数

注意：如果一个指定名字的参数不是搜索内置的参数名，搜索时会把该参数当作指定名字 tag 的属性来搜索，如果包含一个名字为 id 的参数，Beautiful Soup 会搜索每个 tag 的”id”属性

```
1 soup.find_all(id='link2')
2 # [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```

如果传入 href 参数，Beautiful Soup 会搜索每个 tag 的”href”属性

```
1 soup.find_all(href=re.compile("elsie"))
2 # [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>]
```

使用多个指定名字的参数可以同时过滤 tag 的多个属性

```
1 soup.find_all(href=re.compile("elsie"), id='link1')
2 # [<a class="sister" href="http://example.com/elsie" id="link1">three</a>]
```

在这里我们想用 class 过滤，不过 class 是 python 的关键词，这怎么办？加个下划线就可以

```

0% soup.find_all("a", class_="sister")
2 # [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
3 # <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
4 # <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

```

有些 tag 属性在搜索不能使用，比如 HTML5 中的 data-\* 属性

```

1 data_soup = BeautifulSoup('<div data-foo="value">foo!</div>')
2 data_soup.find_all(data-foo="value")
3 # SyntaxError: keyword can't be an expression

```

但是可以通过 find\_all () 方法的 attrs 参数定义一个字典参数来搜索包含特殊属性的 tag

```

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

```

3) 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

```

1 soup.find_all(text="Elsie")
2 # [u'Elsie']
3
4 soup.find_all(text=["Tillie", "Elsie", "Lacie"])
5 # [u'Elsie', u'Lacie', u'Tillie']
6
7 soup.find_all(text=re.compile("Dormouse"))
8 [u"The Dormouse's story", u"The Dormouse's story"]

```

4) 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。效果与 SQL 中的 limit 关键字类似，当搜索到的结果数量达到 limit 的限制时，就停止搜索返回结果。文档树中有 3 个 tag 符合搜索条件，但结果只返回了 2 个，因为我们限制了返回数量

```

1 soup.find_all("a", limit=2)
2 # [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
3 # <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]

```

5) recursive 参数 调用 tag 的 find\_all () 方法时, BeautifulSoup 会检索当前 tag 的所有子孙节点, 如果只想搜索 tag 的直接子节点, 可以使用参数 recursive=False . 一段简单的文档:

↑ 0%

```

1  <html>
2  <head>
3    <title>
4      The Dormouse's story
5    </title>
6  </head>
7  ...

```

是否使用 recursive 参数的搜索结果:

```

1  soup.html.find_all("title")
2  # [<title>The Dormouse's story</title>]
3
4  soup.html.find_all("title", recursive=False)
5  # []

```

## (2) find( name , attrs , recursive , text , \*\*kwargs )

它与 find\_all () 方法唯一的区别是 find\_all () 方法的返回结果是值包含一个元素的列表, 而 find () 方法直接返回结果

## (3) find\_parents() find\_parent()

find\_all () 和 find () 只搜索当前节点的所有子节点, 孙子节点等. find\_parents () 和 find\_parent () 用来搜索当前节点的父辈节点, 搜索方法与普通 tag 的搜索方法相同, 搜索文档搜索文档包含的内容

## (4) find\_next\_siblings() find\_next\_sibling()

这 2 个方法通过 .next\_siblings 属性对当 tag 的所有后面解析的兄弟 tag 节点进行迭代, find\_next\_siblings () 方法返回所有符合条件的后面的兄弟节点, find\_next\_sibling () 只返回符合条件的后面的第一个 tag 节点

## (5) find\_previous\_siblings() find\_previous\_sibling()

这 2 个方法通过 `.previous_siblings` 属性对当前 tag 的前面解析的兄弟 tag 节点进行迭代，`find_previous_siblings()` 方法返回所有符合条件的前面的兄弟节点，`find_previous_sibling()` 方法返回第一个符合条件的前面的兄弟节点

↑ 0%

## (6) `find_all_next()` `find_next()`

这 2 个方法通过 `.next_elements` 属性对当前 tag 的之后的 tag 和字符串进行迭代，`find_all_next()` 方法返回所有符合条件的节点，`find_next()` 方法返回第一个符合条件的节点

## (7) `find_all_previous()` 和 `find_previous()`

这 2 个方法通过 `.previous_elements` 属性对当前节点前面的 tag 和字符串进行迭代，`find_all_previous()` 方法返回所有符合条件的节点，`find_previous()` 方法返回第一个符合条件的节点

注：以上 (2) (3) (4) (5) (6) (7) 方法参数用法与 `find_all()` 完全相同，原理均类似，在此不再赘述。

# 8.CSS 选择器

我们在写 CSS 时，标签名不加任何修饰，类名前加 `.`，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



```
1 print soup.select('title')
2 #[<title>The Dormouse's story</title>]
```



```
1 print soup.select('a')
2 #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
```



```
1 print soup.select('b')
2 #[<b>The Dormouse's story</b>]
```

## (2) 通过类名查找



```
1 print soup.select('.sister')
2 #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
```

↑ 0%

### (3) 通过 id 名查找

```
1 print soup.select('#link1')
2 #[<a class="sister" href="http://example.com/elsie" id="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，例如查找 p 标签中，id 等于 link1 的内容，二者需要用空格分开

```
1 print soup.select('p #link1')
2 #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
```

#### 直接子标签查找

```
1 print soup.select("head > title")
2 #[<title>The Dormouse's story</title>]
```

### (5) 属性查找

查找时还可以加入属性元素，属性需要用中括号括起来，注意属性和标签属于同一节点，所以中间不能加空格，否则会无法匹配到。

```
1 print soup.select('a[class="sister"]')
2 #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
```

```
1 print soup.select('a[href="http://example.com/elsie"]')
2 #[<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

```
1 print soup.select('p a[href="http://example.com/elsie"]')
2 #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
```

↑ 0%

以上的 select 方法返回的结果都是列表形式，可以遍历形式输出，然后用 get\_text() 方法来获取它的内容。

```
1 soup = BeautifulSoup(html, 'lxml')
2 print type(soup.select('title'))
3 print soup.select('title')[0].get_text()
4
5 for title in soup.select('title'):
6     print title.get_text()
```

好，这就是另一种与 find\_all 方法有异曲同工之妙的查找方法，是不是感觉很方便？

## 总结

本篇内容比较多，把 BeautifulSoup 的方法进行了大部分整理和总结，不过这还不算完全，仍然有 BeautifulSoup 的修改删除功能，不过这些功能用得比较少，只整理了查找提取的方法，希望对大家有帮助！小伙伴们加油！ 熟练掌握了 BeautifulSoup，一定会给你带来太多方便，加油吧！

[打赏](#)[简单两步用百度网盘轻松收取文件](#)[Win7 用 Eclipse 配置 JSP 开发环境详细教程](#) >

未找到相关的 [Issues](#) 进行评论

请联系 @germey 初始化创建

[使用 GitHub 登录](#)

© 2021  崔庆才 | 静觅 |  2.5m |  37:35 由 Hexo & NexT.Pisces 强力驱动

京 ICP 备 18015597 号 - 1

 0% 站长统计 | 今日 IP [1030] | 今日 PV [2837] | 昨日 IP [1682] | 昨日 PV [4409] | 当前在线 [51]