

# BeautifulSoup4

Beautiful Soup 是一个 HTML/XML 的解析器，主要用于解析和提取 HTML/XML 数据。

它基于 HTML DOM 的，会载入整个文档，解析整个 DOM 树，因此时间和内存开销都会大很多，所以性能要低于 lxml。

BeautifulSoup 用来解析 HTML 比较简单，API 非常人性化，支持 CSS 选择器、Python 标准库中的 HTML 解析器，也支持 lxml 的 XML 解析器。

抓取工具	速度	使用难度	安装难度
正则	最快	困难	无（内置）
BeautifulSoup	慢	最简单	简单
lxml	快	简单	一般

## 安装

```
pip install beautifulsoup4
```

官方文档：[http://beautifulsoup.readthedocs.io/zh\\_CN/v4.4.0](http://beautifulsoup.readthedocs.io/zh_CN/v4.4.0)

---

## 示例

```
from bs4 import BeautifulSoup

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 BeautifulSoup 对象

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

#打开本地 HTML 文件的方式来创建对象

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

#格式化输出 soup 对象的内容

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

- 如果我们在 IPython2 下执行，会给出警告警告：

UserWarning: No parser was explicitly specified, so I'm using the best available HTML parser for this system ("lxml"). This usually isn't a problem, but if you run this code on another system, or in a different virtual environment, it may use a different parser and behave differently.

- 意思是，如果我们没有显式地指定解析器，所以默认使用这个系统的最佳可用 HTML 解析器(“lxml”)。
- 但是我们可以通过 `soup = BeautifulSoup(html, “lxml”)` 方式指定 lxml 解析器。

## 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

# 1. Tag

Tag 通俗点讲就是 HTML 中的一个标签，例如：

```
<head><title>The Dormouse's story</title></head>
<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie
--></a>
<p class="title" name="dromouse"><b>The Dormouse's story</b></p>
```

上面的 title head a p 等等 HTML 标签加上里面包括的内容就是 Tag

## 用 BeautifulSoup 来获取 Tags

```
from bs4 import BeautifulSoup

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 BeautifulSoup 对象
soup = BeautifulSoup(html, 'lxml')
print(soup.title)
print(soup.head)
print(soup.a)
print(soup.p)
print(type(soup.p))
```

可以利用 soup 加标签名轻松地获取这些标签的内容

这些对象的类型是 `bs4.element.Tag`。

注意，它查找的是在所有内容中的第一个符合要求的标签。

对于 Tag，它有两个重要的属性，是 `name` 和 `attrs`

```
#soup 对象本身比较特殊，它的 name 即为 [document]
print(soup.name)
print(soup.head.name)
#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。
print(soup.p.attrs)
#根据名称获取对应的属性值，类型为列表
print(soup.p['class'])
print(soup.p.get('class'))
# 可以对这些属性和内容等等进行修改
soup.p['class'] = "newClass"
print(soup.p)
# 删除属性
del soup.p['class']
print(soup.p)
```

## 2. NavigableString

获取标签内部的文字用 `.string` 即可，例如

```
print(soup.p.string)
# The Dormouse's story

print(type(soup.p.string))
# In [13]: <class 'bs4.element.NavigableString'>
```

## 3. BeautifulSoup

BeautifulSoup 对象表示的是一个文档的内容。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

```
print(type(soup.name))
# <type 'unicode'>

print(soup.name)
# [document]
```

```
print(soup.attrs) # 文档本身的属性为空
# {}
```

## 4. Comment

Comment 对象是一个特殊类型的 NavigableString 对象，其输出注释但不包含注释符号。

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

print(soup.a.string)
# Elsie

print(type(soup.a.string))
# <class 'bs4.element.Comment'>
```

a 标签里的内容实际上是注释，但是如果我们利用 .string 来输出它的内容时，注释符号已经去掉了。

## 遍历文档树

### 1. 直接子节点：.contents .children 属性

#### .content

tag 的 .content 属性可以将 tag 的子节点以列表的方式输出

```
print(soup.head.contents)
print(soup.head.contents[0])
```

#### .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

```
print(soup.head.children)
for child in soup.body.children:
    print(child)
```

## 2. 所有子孙节点: `.descendants` 属性

`.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。

```
for child in soup.descendants:
    print(child)
```

## 3. 节点内容: `.string` 属性

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:

```
print(soup.head.string)
#The Dormouse's story
print(soup.title.string)
#The Dormouse's story
```

# 搜索文档树

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

### 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

Beautiful Soup 会查找与字符串完整匹配的内容

**范例：**用于查找文档中所有的<b>标签:

```
print(soup.find_all('b'))
print(soup.find_all('a'))
```

## B. 传正则表达式

Beautiful Soup 会通过正则表达式的 match()来匹配内容.

**范例：**找出所有以 b 开头的标签,这表示<body>和<b>标签都应该被找到

```
import re
for tag in soup.find_all(re.compile("^b")):
    print(tag.name)
```

## C. 传列表

Beautiful Soup 会将与列表中任一元素匹配的内容返回.

**范例：**找到文档中所有<a>标签和<b>标签:

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

## 2) keyword 参数

```
print(soup.find_all(id='link2'))
```

## 3) text 参数

text 参数接受字符串,正则表达式,列表, 可以搜搜文档中的字符串内容

```
print(soup.find_all(text="Elsie"))
print(soup.find_all(text=["Tillie", "Elsie", "Lacie"]))
print(soup.find_all(text=re.compile("Dormouse")))
```

## CSS 选择器

- 写 CSS 时, 标签名不加任何修饰, 类名前加., id 名前加#
- 可以利用类似的 soup.select()方法来筛选元素, 返回结果是 list

- `soup.select_one()` 返回值是 list 的首个。

### (1) 通过标签名查找

```
print(soup.select('title'))
print(soup.select('a'))
print(soup.select('b'))
```

### (2) 通过类名查找

```
print(soup.select('.sister'))
```

### (3) 通过 id 名查找

```
print(soup.select('#link1'))
```

### (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

```
print(soup.select('p #link1'))
```

直接子标签查找，则使用 > 分隔

```
print(soup.select("head > 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。

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

## (6) 获取内容

可以遍历 select 方法返回的结果，然后用 get\_text() 方法来获取它的内容。

```
soup = BeautifulSoup(html, 'lxml')
print(type(soup.select('title')))
print(soup.select('title')[0].get_text())

for title in soup.select('title'):
    print(title.get_text())
```

## 案例：腾讯招聘爬虫

需求：

<http://hr.tencent.com/position.php?&start=10#a>

使用 BeautifulSoup4 解析器，将招聘网页上的职位名称、职位类别、招聘人数、工作地点、发布时间，以及每个职位详情的点击链接存储出来。

## 案例：双色球爬虫

<http://zst.aicai.com/ssq/openInfo/>

需求：爬取双色球历史数据，提取期号，开奖日期，红球，篮球，总投注额，一等奖注数，二等奖奖金，二等奖注数，二等奖奖金，奖池滚存（元）

## 案例：豆瓣图书爬虫

需求：爬取豆瓣图书的标签分类信息，然后爬取不同标签下图书的书名，作者，译者，评价人数，评分，封面链接和简介等信息

<https://book.douban.com/tag/?view=type&icn=index-sorttags-all>

<https://book.douban.com/tag/?icn=index-nav>

[https://book.douban.com/tag/\[此处是标签名\]](https://book.douban.com/tag/[此处是标签名])

[https://book.douban.com/subject/\[书的编号\]/](https://book.douban.com/subject/[书的编号]/)

## 案例：空姐网相册爬虫

需求：爬取空姐网相册中照片，保存在存储在 images 目录下，文件命名：uid + picid + '.jpg'

<http://www.kongjie.com/home.php?mod=space&do=album&view=all&page=1>

## 思考问题

### html 里节点、标签的区别

**标签**是 html 最基本的单位，由尖括号包围的关键词，比如 <html>，通常有开始标签和结

束标签，成对出现比如 <div> 和 </div>，不区分大小写，推荐全小写

**节点** DOM 对 html 文档进行操作时，每个元素都可以称之为一个节点。