解析采集到的网页

学习目标：

- 了解网页数据的提取方法；
- 掌握使用正则表达式解析网页的方法；
- 掌握使用 Beautiful Soup 解析网页的方法；
- 掌握使用 lxml 工具解析网页的方法。

HTML 网页数据的解析和提取是 Python 网络数据采集开发中非常关键的步骤。HTML 网页的解析和提取有很多方法，本章将从使用正则表达式解析、使用 Beautiful Soup 解析、使用 lxml 解析这三个方面进行讲解。

3.1 使用正则表达式解析

简单地说，正则表达式是一种可以用于模式匹配和替换的强大工具。在几乎所有基于 UNIX/Linux 系统的软件工具中都能找到正则表达式的痕迹，如 Perl 或 PHP 脚本语言。此外，JavaScript 这种脚本语言也提供了对正则表达式的支持。现在，正则表达式已经成为一个通用的工具，被各类技术人员广泛使用。

3.1.1 基本语法与使用

正则表达式是一个很强大的字符串处理工具，几乎任何关于字符串的操作都可以使用正则表达式完成。作为一个数据采集工作者，几乎每天都要和字符串打交道，正则表达式更是不可或缺的技能。正则表达式在不同开发语言中的使用方式可能不一样，不过只要学会了任意一门语言的正则表达式的用法，其他语言中大部分也只是换了一个函数名称而已，本质上都是一样的。

首先，Python 中的正则表达式大致分为以下几部分。

元字符

模式

函数

re 内置对象用法

分组用法

环视用法

其次，所有关于正则表达式的操作都使用 Python 标准库中的 re 模块完成，部分正则表达式如表 3-1 所示。

表 3-1　部分正则表达式

模式	描　　述	模式	描　　述
.	匹配任意字符(不包括换行符)	\A	匹配字符串开始位置，忽略多行模式
^	匹配开始位置，多行模式下匹配每一行的开始	\Z	匹配字符串结束位置，忽略多行模式
$	匹配结束位置，多行模式下匹配每一行的结束	\b	匹配位于单词开始或结束位置的空字符串
*	匹配前一个元字符 0 到多次	\B	匹配不在单词开始或结束位置的空字符串
+	匹配前一个元字符 1 到多次	\d	匹配一个数字，相当于[0-9]
?	匹配前一个元字符 0 到 1 次	\D	匹配非数字，相当于[^0-9]
{m,n}	匹配前一个元字符 m 到 n 次	\s	匹配任意空白字符，相当于[\t\n\r\f\v]

3.1.2　Python 与正则表达式

正则表达式是一个特殊的字符序列，它能帮助你方便地检查一个字符串是否与某种模式匹配。Python 自 1.5 版本开始就增加了 re 模块，提供 Perl 风格的正则表达式模式。re 模块使 Python 语言拥有全部的正则表达式功能。compile 函数根据一个模式字符串和可选的标志参数生成一个正则表达式对象，该对象拥有一系列用于正则表达式匹配和替换的方法。

re 模块也提供了与这些方法的功能完全一致的函数，这些函数使用一个模式字符串作为第一个参数。下面主要介绍 Python 中常用的正则表达式处理函数。

1. re.match 函数

re.match 函数尝试从字符串的起始位置匹配一个模式，如果不是在起始位置匹配成功，则返回 none，该函数的语法如下。

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

函数参数说明如表 3-2 所示。

表 3-2　re.match 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

参　数	描　述
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。参见：正则表达式修饰符-可选标志

若匹配成功，则 re.match 函数返回一个匹配的对象；否则返回 None。可以使用 group(num)或 groups()方法匹配对象函数以获取匹配表达式。

具体示例代码如下。

```
#!/usr/bin/python
#-*- coding: UTF-8 -*-
import re
print(re.match('www', 'www.tup.tsinghua.edu.cn').span())    #在起始位置匹配
print(re.match('com', 'www.tup.tsinghua.edu.cn'))           #不在起始位置匹配
```

以上示例运行后的输出结果如下。

```
(0, 3)
None
```

2. re.search 函数

re.search 函数用于扫描整个字符串，并返回第一个成功的匹配。

该函数的语法如下。

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

函数参数说明如表 3-3 所示。

表 3-3　re.search 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

若匹配成功，则 re.search 函数会返回一个匹配的对象；否则返回 None。

具体示例代码如下。

```
#!/usr/bin/python
#-*- coding: UTF-8 -*-
import re
print(re.search('www', 'www.tup.tsinghua.edu.cn').span())   #在起始位置匹配
print(re.search('com', 'www.tup.tsinghua.edu.cn').span())   #不在起始位置匹配
```

以上示例运行后的输出结果如下。

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

3. re.match 函数与 re.search 函数的区别

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

具体示例代码如下。

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs";
matchObj = re.match( r'dogs', line, re.M|re.I)
if matchObj:
    print ("match --> matchObj.group() : ", matchObj.group())
else:
    print("No match!!" )
matchObj = re.search( r'dogs', line, re.M|re.I)
if matchObj:
    print("search --> matchObj.group() : ", matchObj.group())
else:
    print("No match!!" )
```

以上示例运行后的输出结果如下。

```
No match!!
search --> matchObj.group() :  dogs
```

下面是正则表达式的相关实例。

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs"
matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
if matchObj:
    print("matchObj.group() : ", matchObj.group())
    print("matchObj.group(1) : ", matchObj.group(1))
    print ("matchObj.group(2) : ", matchObj.group(2))
else:
    print("No match!!")
```

其中，正则表达式如下。

```
r'(.*) are (.*?) .*'
```

首先，这是一个字符串，前面的一个 r 表示字符串为非转义的原始字符串，让编译器忽略反斜杠，也就是忽略转义字符。但是这个字符串中没有反斜杠，所以这个 r 可有可无。

“(.*)”为第一个匹配分组，“.*”代表匹配除换行符之外的所有字符。

“(.*?)”为第二个匹配分组，“.*?”后面的多个问号代表非贪婪模式，即只匹配符合条件的最少字符。

最后的一个“.*”没有被括号包围，所以不是分组，匹配效果和第一个“.*”一样，但是不计入匹配结果。

matchObj.group()等同于 matchObj.group(0)，表示匹配到的完整文本字符。

matchObj.group(1)得到第一组匹配结果，也就是“(.*)”匹配到的。

matchObj.group(2)得到第二组匹配结果，也就是“(.*?)”匹配到的。

因为匹配结果中只有两组，所以填 3 时会报错。

当使用正则表达式爬取面积数据时，首先需要尝试匹配<td>元素中的内容，示例代码如下所示。

```
#-*-coding:utf-8-*-
import urllib3,re
http = urllib3.PoolManager()
r =
http.request('GET','www.runoob.com/places/default/view/Andorra-6')
def scrape(html):
    area = re.findall('<tdclass="w2p_fw">(.*?)</td>',html)[1]
    return area
print(scrape(str(r.data)))
```

从上述结果可以看出，多个国家属性都使用了<td class="w2pfw">标签。要想分离出面积属性，则可以只选择其中的第二个元素，示例如下。

```
re.findall('<td class="w2p_fw">(.*2?)</td>,html)[1]
'244,820 square kilometres'
```

这个迭代版本看起来更好一些，但是网页更新还有很多其他方式，同样使得该正则表达式无法满足。如将双引号变为单引号，或在<td>标签之间添加多余的空格。下面是尝试支持这些可能性的改进版本。

```
re.findall('<tr id="places_area__row"><td
class="w2p_fl"><label for="places_area"
    id="places_area__label">Area: </label></td><td
      class="w2p_fw">(.*?)</td>', html
['244,820 square kilometres']
```

3.2 使用 Beautiful Soup 解析

通过 Requests 库已经可以抓到网页源码了，接下来要从源码中找到并提取数据。BeautifulSoup 是 Python 的一个库，其主要功能是从网页中爬取数据。Beautiful Soup 目前已经被移植到 bs4 库中，也就是说，在导入 BeautifulSoup 时需要先安装 bs4 库。安装 bs4 库后，还需要安装 lxml 库。如果不安装 lxml 库，则会使用 Python 默认的解析器。下面将介绍 Beautiful Soup 4 的安装和使用方法。

3.2.1 Python 网页解析器

网页解析器是用来解析 HTML 网页的工具，它是一个 HTML 网页信息提取工具，就是从 HTML 网页中解析并提取出“需要的、有价值的数据”或者“新的 URL 列表”的工具。

网页解析器将从下载到的 URL 数据中提取有价值的数据和生成新的 URL。对于数据提取，可以使用正则表达式和 Beautiful Soup 等方法。正则表达式使用基于字符串的模糊匹配，对于特点比较鲜明的目标数据具有较好的效果，但通用性不高。使用正则表达式进行网页解析的过程如图 3-1 所示。

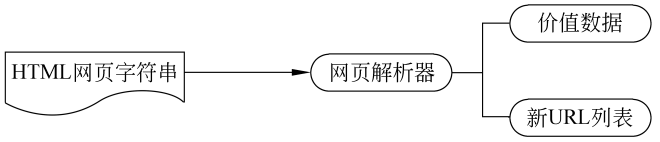

图 3-1 网页解析示意

Python 有以下几种网页解析器：正则表达式、html.parser、Beautiful Soup、lxml。常见的 Python 网页解析工具有 re 正则匹配、Python 自带的 html.parser 模块、第三方库 Beautiful Soup 及 lxm 库。

以上 4 种网页解析工具属于不同类型的解析器，如图 3-2 所示。

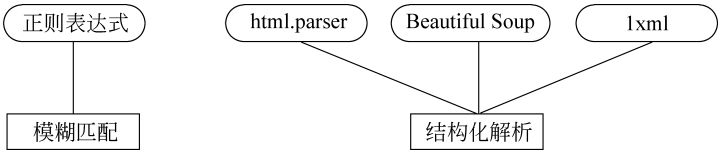

图 3-2 模糊匹配和结构化解析示意

① 模糊匹配。re 正则表达式即为字符串式的模糊匹配模式。

② 结构化解析。Beautiful Soup、html.parser 与 lxml 为结构化解析模式，它们都以 DOM 树结构为标准进行标签结构信息的提取。

DOM 树即文档对象模型(Document Object Model)，其树形标签结构如图 3-3 所示。

结构化解析是指网页解析器将下载的整个 HTML 文档当作一个 Document 对象，然

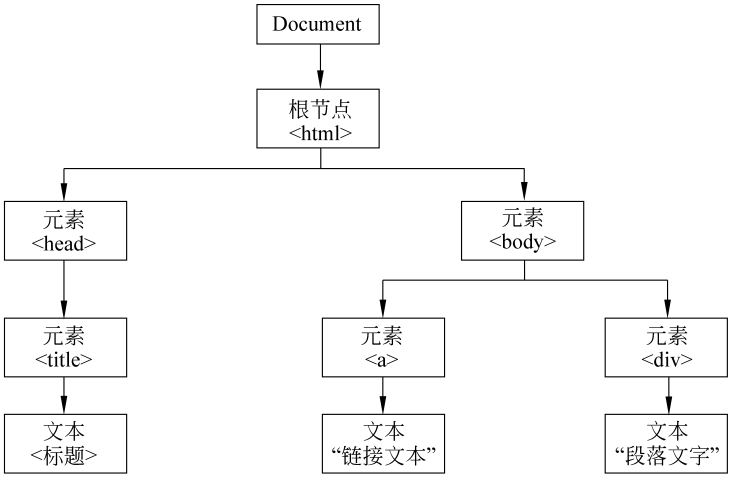

图 3-3 DOM 树结构

后利用其上下结构的标签形式对这个对象的上下级标签进行遍历和信息提取操作。

3.2.2 Beautiful Soup 第三方库

1. Beautiful Soup 简介

Beautiful Soup 是一个可以从 HTML 或 XML 文件中提取数据的 Python 库，它能够通过用户喜欢的转换器实现惯用文档导航、查找和修改文档等操作。在基于 Python 的数据采集开发中，主要用到的是 Beautiful Soup 的查找和提取功能，修改文档方式很少用到。可以说，Beautiful Soup 是一个非常流行的 Python 模块。

简单来说，Beautiful Soup 是 Python 的一个库，其主要功能是从网页上爬取数据。Beautiful Soup 提供一些简单的、Python 式的函数以处理导航、搜索、修改分析树等，它是一个工具箱，通过解析文档为用户提供需要爬取的数据，不需要多少代码就可以写出一个完整的应用程序。Beautiful Soup 可以自动将输入文档转换为 Unicode 编码，将输出文档转换为 UTF-8 编码，不需要考虑编码方式，除非文档没有指定编码方式，这时，Beautiful Soup 就不能自动识别编码方式了，需要说明原始编码方式。Beautiful Soup 已成为和 lxml、html6lib 一样出色的 Python 解析器，可以为用户灵活地提供不同的解析策略或更快的速度。Beautiful Soup 会帮助开发人员节省数小时甚至数天的工作时间。

2. 安装 Beautiful soup 4

(1) 安装方法

接下来介绍 Beautiful Soup 在 Python 网络数据采集开发中的使用，该模块可以解析网页，并提供定位内容的便捷接口。如果还没有安装该模块，则可以使用以下命令安装其最新版本。

```
#pip install beautifulsoup4
```

beautifulsoup 4(简称 bs4)通过 PyPi 发布,如果无法使用系统包管理安装,那么也可以通过 easy_install 或 pip 进行安装。包的名字是 beautiful soup 4,兼容 Python 2 和 Python 3。

```
#yum install libxslt-devel python-devel
#pip install lxml
```

在 PyPi 中还有一个名字是 Beautiful Soup 的包,它是 Beautiful Soup 3 的发布版本,因为很多项目还在使用 bs3,所以 Beautiful Soup 包依然有效。但如果你在编写新项目,那么你应该安装的是 bs4。

如果没有安装 easy_install 或 pip,也可以下载 bs4 的源码,然后通过 setup.py 进行安装。

```
#python setup.py install
```

如果上述安装方法都行不通,那么 Beautiful Soup 的发布协议允许用户将 bs4 的代码打包在项目中,这样一来无须安装即可使用。在 Python 2.7 和 Python 3.2 版本下开发 Beautiful Soup,理论上 Beautiful Soup 应该在所有当前的 Python 版本中都能正常工作。

Beautiful Soup 在发布时被打包成 Python 2 版本的代码,在 Python 3 环境下安装时会自动转换成 Python 3 的代码,如果没有一个安装的过程,那么代码就不会被转换。如果代码抛出了 ImportError 的异常“No module named HTMLParser”,则表明在 Python 3 版本中执行了 Python 2 版本的代码。如果代码抛出了 ImportError 的异常“No module named html.parser”,则表明在 Python 2 版本中执行了 Python 3 版本的代码。如果遇到上述两种情况,最好的解决方法是重新安装 Beautiful soup 4。如果在 ROOT_TAG_NAME = u'[document]'代码处遇到 SyntaxError"Invalid syntax"错误,则需要将 bs4 的 Python 代码版本从 Python 2 转换到 Python 3,然后重新安装 bs4 即可。

```
#python3 setup.py install
```

或在 bs4 的目录中执行 Python 的代码版本转换脚本。

```
#2to3-3.2 -w bs4
```

Beautiful Soup 支持 Python 标准库中的 HTML 解析器,还支持一些第三方解析器,其中一个是 lxml。根据操作系统的不同,可以选择下列方法安装 lxml。

```
#yum install Python-html5lib
#pip install html5lib
```

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

```
#apt-get install Python-html5lib
#easy_install html5lib
pip install html5lib
```

(2) 安装测试

安装完成后是一个 Python 线下自带的、简洁的集成开发环境，测试 bs4 是否安装成功的示例如下。

```
from bs4 import Beautiful Soup
import bs4
print bs4
```

3. Beautiful Soup 语法

使用 Beautiful Soup 的一般流程有以下 3 个步骤。

① 创建 Beautiful Soup 对象。

② 使用 Beautiful Soup 对象的操作方法 find_all 与 find 进行解读搜索，示例如下。

```
soup.find_all('a')
soup.find('a')
```

③ 利用 DOM 结构的标签特性进行更为详细的节点信息提取。

4. 使用方法

① 创建 Beautiful Soup 对象(即 DOM 对象)，示例如下。

```
#引入 Beautiful Soup 库
      from bs4 import Beatiful Soup
#根据 HTML 网页字符串结构创建 Beatiful Soup 对象
      soup=Beautiful Soup(html_doc,                    #HTML 文档字符串
'html.parser',                                         #HTML 解析器
from_encoding='utf-8'                                  #HTML 文档编码
)
```

② 搜索节点(find_all,find)，方法如下。

- soup.find_all()：查找所有符合查询条件的标签节点，并返回一个列表。
- soup.find()：查找符合查询条件的第一个标签节点。

实例 1：搜索所有<a>标签，示例如下。

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

实例 2：查找所有标签名为 a 且链接符合“/view/123.html”的节点。

实现方法 1 的示例如下。

```
soup.find_all('a',href='/view/123.html')
```

实现方法 2 的示例如下。

```
soup.find_all('a',href=re.compile(r'/view/\d+\.html'))
```

实例 3：查找所有标签名为 a、class 属性为 abc、文字为 Python 的节点，示例如下。

```
soup.findall('a',class_='abc',string='Python')
```

③ 访问节点信息。

当得到节点＜a href＝'/view/123.html'class＝'doc_link'＞I love Python＜a＞时，首先需要获取节点名称，示例如下。

```
node.name
```

其次是获取查找到的 a 节点的 href 属性，示例如下。

```
node['href']
```

或者示例如下。

```
node.get('href')
```

最后是获取查找到的 a 节点的字符串内容，示例如下。

```
node.get_text()
```

5. Beautiful Soup 信息提取实例

相关示例的实现代码如下。

```
from bs4 import Beautiful Soup
html_doc="""<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><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://www.tup.tsinghua.edu.cn/elsie"class= "sister"id="link1">
Elsie</a>,<a href="http://www.tup.tsinghua.edu.cn/lacie"class="sister"id="
link2"> Lacie</a>and<a
href="http://www.tup.tsinghua.edu.cn/tillie"class="sister"id="link3">
Tillie</a>;
and they lived at the bottom of a well.</p>
```

```
<p class="story">…</p>"""
links=Beautiful Soup(html_doc,'lxml')
print(links.name,links.a['href'],links.get_text())
```

将一段文档传入 Beautiful Soup 的构造方法就能得到一个文档的对象。下面是传入一段字符串或一个文件句柄的示例。

```
from bs4 import Beautiful Soup
soup = Beautiful Soup(open("index.html"))
soup = Beautiful Soup("<html>data</html>")
```

首先,文档被转换成 Unicode,并且 HTML 的实例都被转换成了 Unicode 编码,代码如下。

```
Beautiful Soup("Sacr&eacute; bleu!")
<html><head></head><body>Sacré bleu!</body></html>
```

然后,Beautiful Soup 选择最合适的解析器解析这段文档。如果手动指定解析器,那么 Beautiful Soup 会选择指定的解析器解析文档。

下面的一段 HTML 代码将作为例子在本书后续章节中被多次用到,这是《爱丽丝梦游仙境》中的一段内容(后文中简称这段内容为“爱丽丝”)。

```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><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://www.tup.tsinghua.edu.cn/elsie" class="sister" id="link1">
Elsie</a>,
<a href="http://www.tup.tsinghua.edu.cn/lacie" class="sister" id="link2">
Lacie</a> and
<a href="http://www.tup.tsinghua.edu.cn/tillie" class="sister" id="link3">
Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">…</p>
```

使用 Beautiful Soup 解析这段代码,能够得到一个 Beautiful Soup 对象,并能按照标准的缩进格式进行输出,具体如下。

```
soup = Beautiful Soup(html_doc,'lxml')
<html>
<head>
```

```
  <title>
   The Dormouse's story
  </title>
 </head>
 <body>
  <p class="title">
   <b>
    The Dormouse's story
   </b>
  </p>
  <p class="story">
   Once upon a time there were three little sisters; and their names were
    <a class="sister" href="http://www.tup.tsinghua.edu.cn/elsie" id="
link1">
    Elsie
   </a>
   ,
    <a class="sister" href="http://www.tup.tsinghua.edu.cn/lacie" id="
link2">
    Lacie
   </a>
   and
    <a class="sister" href="http://www.tup.tsinghua.edu.cn/tillie" id="
link2">
    Tillie
   </a>
  </p>
  <p class="story">
   ...
  </p>
 </body>
</html>
```

以下是几个简单的浏览结构化数据的方法。

```
soup.title
 <title>The Dormouse's story</title>
print(soup.title.name)
 u'title'
print(soup.t itle.string)
 u'The Dormouse's story'
print(soup.title.parent.name)
 u'head'
```

```
print(soup.p)
 <p class="title"><b>The Dormouse's story</b></p>
print(soup.p['class'])
 u'title'
print(soup.a)
 <a class="sister" href="http://www.tup.tsinghua.edu.cn/elsie" id="link1">
Elsie</a>
print(soup.find_all('a'))
 [<a class="sister" href="http://www.tup.tsinghua.edu.cn/elsie" id="link1"
>Elsie</a>,
  <a class="sister" href="http://www.tup.tsinghua.edu.cn/lacie" id="link2"
>Lacie</a>,
  <a class="sister" href="http://www.tup.tsinghua.edu.cn/tillie" id="link3"
>Tillie</a>]
print(soup.find(id="link3"))
 <a class="sister" href="http://www.tup.tsinghua.edu.cn/tillie" id="link3"
```

从文档中找到所有<a>标签的链接的代码如下。

```
for link in soup.find_all('a'):
    print(link.get('href'))
     http://www.tup.tsinghua.edu.cn/elsie
     http://www.tup.tsinghua.edu.cn/lacie
     http://www.tup.tsinghua.edu.cn/tillie
```

从文档中获取所有文字内容的代码如下。

```
print(soup.get_text())
 The Dormouse's story
 The Dormouse's story
 Once upon a time there were three little sisters; and their names were
 Elsie,
 Lacie and
 Tillie;
 and they lived at the bottom of a well.
 …
```

6. Beautiful Soup 对象的种类

Beautiful Soup 将复杂的 HTML 文档转换成了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、Beautiful Soup、Comment。

(1) Tag 对象

Tag 对象与 XML 或 HTML 原生文档中的 Tag(标签)相同，相关示例如下。

```
soup = Beautiful Soup('<b class="boldest">Extremely bold</b>')
tag = soup.b
pring(type(tag))
<class 'bs4.element.Tag'>
```

Tag 对象有很多方法和属性，在遍历文档树和搜索文档树中有详细解释。现在介绍 Tag 对象中最重要的属性：name 和 attributes。

① name。每个 Tag 对象都有自己的名字，通过 name 获取，相关示例如下。

```
tag.name
u'b'
```

如果改变了 Tag 对象的 name，则将影响所有通过当前 Beautiful Soup 对象生成的 HTML 文档，相关示例如下。

```
tag.name = "blockquote"
tag
<blockquote class="boldest">Extremely bold</blockquote>
```

② attributes。一个 Tag 对象可能有多个属性，tag ＜b class＝"boldest"＞ 有一个 class 的属性，值为 boldest。Tag 属性的操作方法与字典相同。

```
Tag['class']
['boldest']
```

也可以直接“点”取属性，比如.attrs。

```
Tag.attrs
{'class': ['boldest']}
```

Tag 对象的属性可以被添加、删除或修改。Tag 对象的属性操作方法与字典相同。

```
tag['class'] = 'verybold'
tag['id'] = 1
print(Tag)
<b class="verybold" id="1">Extremely bold</b>
del tag['class']
del tag['id']
print(Tag)
<b>Extremely bold</b>
print(Tag['class'])
Traceback (most recent call last):
  File "<input>", line 1, in <module>
    print(Tag['class'])
```

```
  File "/usr/local/lib/python3.5/site-packages/bs4/element.py", line 997, in
__getitem__
    return self.attrs[key]
KeyError: 'class'
print(tag.get('class'))
None
```

HTML 4 定义了一系列可以包含多个值的属性，虽然在 HTML 5 中移除了一些，但 HTML 5 却增加了更多的其他属性。最常见的多值属性是 class（一个 Tag 对象可以有多个 CSS 的 class），还有一些其他属性，如 rel、rev、accept-charset、headers、accesskey。在 Beautiful Soup 中，多值属性的返回类型是 list，相关示例如下。

```
css_soup = Beautiful Soup('<p class="body strikeout"></p>')
css_soup.p['class']
 ["body", "strikeout"]
css_soup = Beautiful Soup('<p class="body"></p>')
css_soup.p['class']
 ["body"]
```

如果某个属性看起来好像有多个值，但它在任何版本的 HTML 定义中都没有被定义为多值属性，那么 Beautiful Soup 会将这个属性作为字符串返回，相关示例如下。

```
id_soup = Beautiful Soup('<p id="my id"></p>')
id_soup.p['id']
 'my id'
```

将 Tag 对象转换成字符串时，多值属性会合并为一个值，相关示例如下。

```
rel_soup = Beautiful Soup('<p>Back to the <a rel="index">homepage</a></p>')
rel_soup.a['rel']
 ['index']
rel_soup.a['rel'] = ['index', 'contents']
print(rel_soup.p)
 <p>Back to the <a rel="index contents">homepage</a></p>
```

如果转换的文档是 XML 格式，那么相关标签中就不包含多值属性，相关示例如下。

```
xml_soup = Beautiful Soup('<p class="body strikeout"></p>', 'xml')
xml_soup.p['class']
 u'body strikeout'
```

（2）NavigableString 对象

Beautiful Soup 使用 NavigableString 类包装 Tag 对象中的字符串，相关示例如下。

```
print(tag.string)
 u'Extremely bold'
print(type(tag.string))
 <class 'bs4.element.NavigableString'>
```

NavigableString 字符串与 Python 中的 Unicode 字符串相同，并且支持包含在遍历文档树和搜索文档树中的一些特性。通过 unicode()方法可以直接将 NavigableString 对象转换成 Unicode 字符串，相关示例如下。

```
unicode_string = unicode(tag.string)
unicode_string
 u'Extremely bold'
print(type(unicode_string))
 <type 'unicode'>
```

Tag 对象中包含的字符串不能编辑，但可以被替换成其他字符串。替换时会用到 replace_with()方法，相关示例如下。

```
tag.string.replace_with("No longer bold")
print(tag)
 <blockquote>No longer bold</blockquote>
```

NavigableString 对象支持遍历文档树和搜索文档树中定义的大部分属性，但并非全部。尤其是一个字符串不能包含其他内容(Tag 对象能够包含字符串或其他 Tag 对象)，字符串不支持 contents 或 string 属性或 find()方法。如果想在 Beautiful Soup 之外使用 NavigableString 对象，则需要调用 unicode()方法，将该对象转换成普通的 Unicode 字符串；否则就算 Beautiful Soup 的方法已经执行结束，该对象的输出也会带有对象的引用地址，这样会浪费内存。

(3) Beautiful Soup 对象

Beautiful Soup 对象表示一个文档的全部内容，大多情况下可以把它当作 Tag 对象。Beautiful Soup 对象支持遍历文档树和搜索文档树中描述的大部分方法。因为 Beautiful Soup 对象并不是真正的 HTML 或 XML 的 Tag 对象，所以它没有 name 和 attributes 属性，但查看它的 name 属性却是很方便的，所以 Beautiful Soup 对象包含一个值为“[document]”的特殊属性 name。

```
print(soup.name)
 u'[document]'
```

Beautiful Soup 中定义的其他类型都可能出现在 XML 文档中，如 CData、ProcessingInstruction、Declaration、Doctype。与 Comment 对象类似，这些类都是 NavigableString 的子类，只是添加了一些额外的方法。下面是使用 CData 替代注释的例子。

```
from bs4 import CData
cdata = CData("A CDATA block")
comment.replace_with(cdata)
print(soup.b.prettify())
 <b>
  <![CDATA[A CDATA block]]>
 </b>
```

(4) Comment 对象

Comment 对象是一个特殊类型的 NavigableString 对象,相关示例如下。

```
print(comment)
 u'Hey, buddy. Want to buy a used parser'
```

当在 HTML 文档中出现时,Comment 对象会使用特殊的格式输出,相关示例如下。

```
print(soup.b.prettify())
 <b>
  <!--Hey, buddy. Want to buy a used parser?-->
 </b>
```

7. 遍历文档树

下面用“爱丽丝”文档举例,文档的结构如下。

```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<p class="title"><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://www.tup.tsinghua.edu.cn/elsie" class="sister" id="link1">
Elsie</a>,
<a href="http://www.tup.tsinghua.edu.cn/lacie" class="sister" id="link2">
Lacie</a> and
<a href="http://www.tup.tsinghua.edu.cn/tillie" class="sister" id="link3">
Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">…</p>
"""
from bs4 import Beautiful Soup
soup = Beautiful Soup(html_doc)
```

下面演示怎样从文档的一段内容中找到另一段内容。

(1) 子节点

一个 Tag 对象可能包含多个字符串或其他 Tag 对象,这些都是这个 Tag 对象的子节点。Beautiful Soup 提供了许多操作和遍历子节点的属性。

注意:在 Beautiful Soup 中,字符串节点不支持遍历子节点的属性,这是因为字符串没有子节点。

操作文档树最简单的方法就是告诉它想要获取的 Tag 对象的 name。如果想获取<head>标签,则只需要用到 soup.head,相关示例如下。

```
print(soup.head)
 <head><title>The Dormouse's story</title></head>
print(soup.title)
 <title>The Dormouse's story</title>
```

上述代码是获取 Tag 对象标签的小窍门,可以在文档树的标签中多次调用这个方法。下面的代码可以获取<body>标签中的第一个<b>标签。

```
print(soup.body.b)
 <b>The Dormouse's story</b>
```

通过点取属性的方式只能获得当前名字的第一个节点,相关示例如下。

```
print(soup.a)
 <a class="sister" href="http://www.tup.tsinghua.edu.cn/elsie" id="link1">
Elsie</a>
```

如果想要得到所有<a>标签,或是通过名字得到比一个标签更多的内容时,就需要用到遍历文档树中描述的方法,如 find_all()方法。

```
print(soup.find_all('a'))
 [<a class="sister" href="http://www.tup.tsinghua.edu.cn/elsie" id="link1"
>Elsie</a>,
  <a class="sister" href="http://www.tup.tsinghua.edu.cn/lacie" id="link2"
>Lacie</a>,
  <a class="sister" href="http://www.tup.tsinghua.edu.cn/tillie" id="link3"
>Tillie</a>]
```

(2) 父节点

继续分析文档树,每个 Tag 对象或字符串都有父节点:被包含在某个 Tag 对象中,可以通过 parent 属性获取某个元素的父节点。在“爱丽丝”文档中,<head>标签是<title>标签的父节点。

```
title_tag = soup.title
print(title_tag)
```

```
 <title>The Dormouse's story</title>
print(title_tag.parent)
 <head><title>The Dormouse's story</title></head>
```

文档 title 的字符串也有父节点，即<title>标签，相关代码如下。

```
title_tag.string.parent
 <title>The Dormouse's story</title>
```

文档的顶层节点，如<html>的父节点是 Beautiful Soup 对象，相关代码如下。

```
html_tag = soup.html
print(type(html_tag.parent))
 <class 'bs4.Beautiful Soup'>
```

Beautiful Soup 对象的 parent 是 None，相关代码如下。

```
print(soup.parent)
 None
```

通过元素的 parents 属性可以递归地得到元素的所有父节点。下面的例子使用 parents 属性遍历了从<a>标签到根节点的所有节点。

```
link = soup.a
print(link)
 <a class="sister" href="http://www.tup.tsinghua.edu.cn/elsie" id="link1">
Elsie</a>
for parent in link.parents:
    if parent is None:
        print(parent)
    else:
        print(parent.name)
 p
 body
 html
 [document]
 None
```

(3) 兄弟节点

对于兄弟节点，可以通过下面的例子理解，具体如下。

```
sibling_soup = Beautiful Soup("<a><b>text1</b><c>text2</c></b></a>")
print(sibling_soup.prettify())
 <html>
```

```
 <body>
  <a>
   <b>
    text1
   </b>
   <c>
    text2
   </c>
  </a>
 </body>
</html>
```

因为<b>标签和<c>标签属于同一层(它们是同一个元素的子节点),所以<b>标签和<c>标签可以被称为兄弟节点。当一段文档以标准格式输出时,兄弟节点有相同的缩进级别,在代码中也可以使用这种关系。

在文档树中,使用 next_sibling 和 previous_sibling 属性查询兄弟节点,相关代码如下。

```
print(sibling_soup.b.next_sibling)
 <c>text2</c>
print(sibling_soup.c.previous_sibling)
 <b>text1</b>
```

<b>标签有 next_sibling 属性,但是没有 previous_sibling 属性,因为<b>标签在同级节点中是第一个。同理,<c>标签有 previous_sibling 属性,却没有 next_sibling 属性,相关代码如下。

```
print(sibling_soup.b.previous_sibling)
 None
print(sibling_soup.c.next_sibling)
 None
```

例子中的字符串 text1 和 text2 不是兄弟节点,因为它们的父节点不同,相关代码如下。

```
Print(sibling_soup.b.string)
 u'text1'
print(sibling_soup.b.string.next_sibling)
 None
```

另外,Tag 对象的 next_sibling 和 previous_sibling 属性通常是字符串或空白,首先看以下代码。