5. bs4 針對HTML解析出的 四大對象種類

Beautiful Soup將複雜HTML文檔轉換成一個複雜的樹形結構,每個節點都是Python對象,所有對象可以歸納為4種:

1. Tag
2. NavigableString
3. BeautifulSoup
4. Comment

**（1）Tag**

Tag 是什麼？通俗點講就是HTML 中的一個個標籤，例如:

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

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

上面的title a 等等HTML 標籤加上里麵包括的內容就是Tag，下面我們來感受一下怎樣用Beautiful Soup 來方便地獲取Tags

下面每一段代碼中註釋部分即為運行結果:

print soup.title

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

print soup.head

#<head><title>The Dormouse's story</title></head>

print soup.a

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

print soup.p

#<p class="title" name=dromouse><b>The Dormouse's story</b></p>

我們可以利用soup加標籤名輕鬆地獲取這些標籤的內容，是不是感覺比正則表達式方便多了？不過有一點是，它查找的是在所有內容中的第一個符合要求的標籤(似find)，如果要查詢所有的標籤，我們在後面進行介紹。

我們可以驗證一下這些對象的類型:

print type(soup.a)

#<class 'bs4.element.Tag'>

對於Tag，它有兩個重要的屬性，是name 和attrs，下面我們分別來感受一下。

**name**

print soup.name

print soup.head.name

#[document]

#head

soup 對象本身比較特殊，它的name 即為[document]，對於其他內部標籤，輸出的值便為標籤本身的名稱。

**attrs**

print soup.p.attrs

#{'class': ['title'], 'name': 'dromouse'}

在這裡，我們把p 標籤的所有屬性打印輸出了出來，得到的類型是一個字典。

如果我們想要單獨獲取某個屬性，可以這樣，例如我們獲取它的class 叫什麼。

print soup.p['class']

#['title']

還可以這樣，利用get方法，傳入屬性的名稱，二者是等價的。

print soup.p.get('class')

#['title']

我們可以對這些屬性和內容等等進行修改，例如:

soup.p['class']="newClass"

print soup.p

#<p class="newClass" name=dromouse><b>The Dormouse's story</b></p>

還可以對這個屬性進行刪除，例如:

del soup.p['class']

print soup.p

#<p name=dromouse><b>The Dormouse's story</b></p>

不過，對於修改刪除的操作，不是我們的主要用途，在此不做詳細介紹了，如果有需要，請查看前面提供的官方文檔

**（2）NavigableString**

既然我們已經得到了標籤的內容，那麼問題來了，我們要想獲取標籤內部的文字怎麼辦呢？很簡單，用.string 即可，例如:

print soup.p.string

#The Dormouse's story

這樣我們就輕鬆獲取到了標籤裡面的內容，想想如果用正則表達式要多麻煩。它的類型是一個 NavigableString，翻譯過來叫可以遍歷的字符串，不過我們最好還是稱它英文名字吧。

來檢查一下它的類型

print type(soup.p.string)

#<class 'bs4.element.NavigableString'>

**（3）BeautifulSoup**

BeautifulSoup對象表示的是一個文檔的全部內容.大部分時候,可以把它當作 Tag對象，是一個特殊的Tag，我們可以分別獲取它的類型，名稱，以及屬性來感受一下。

print type(soup.name)

#<type 'unicode'>

print soup.name

# [document]

print soup.attrs

#{} 空字典

**（4）Comment**

Comment對像是一個特殊類型的 NavigableString對象，其實輸出的內容仍然不包括註釋符號，但是如果不好好處理它，可能會對我們的文本處理造成意想不到的麻煩。

我們找一個帶註釋的標籤:

print soup.a

print soup.a.string

print type(soup.a.string)

運行結果如下:

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

Elsie

<class 'bs4.element.Comment'>

a 標籤裡的內容實際上是註釋，但是如果我們利用.string 來輸出它的內容，我們發現它已經把註釋符號去掉了，所以這可能會給我們帶來不必要的麻煩。

另外我們打印輸出下它的類型，發現它是一個Comment 類型，所以，我們在使用前最好做一下判斷，判斷代碼如下:

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

print soup.a.string

上面的代碼中，我們首先判斷了它的類型，是否為Comment 類型，然後再進行其他操作，如打印輸出。

6. 遍歷文檔樹

**（1）直接子節點**

* **要點：.contents .children** **屬性**

**.contents**

tag 的.content 屬性可以將tag的子節點以列表的方式輸出:

print soup.head.contents

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

輸出方式為列表，我們可以用列表索引來獲取它的某一個元素:

print soup.head.contents[0]

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

**.children**

它返回的不是一個list，不過我們可以通過遍歷獲取所有子節點。

我們打印輸出.children 看一下，可以發現它是一個list 生成器對象。

print soup.head.children

#<listiterator object at 0x7f71457f5710>

我們怎樣獲得里面的內容呢？很簡單，遍歷一下就好了，代碼及結果如下:

for child in soup.body.children:

print child

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

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

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

and they lived at the bottom of a well.</p>

<p class="story">...</p>

**（2）所有子孫節點**

* 知識點：.descendants 屬性

**.descendants**

.contents和 .children屬性僅包含tag的直接子節點， .descendants屬性可以對所有tag的子孫節點進行遞歸循環，和children類似，我們也需要遍歷獲取其中的內容。

for child in soup.descendants:

print child

運行結果如下，可以發現，所有的節點都被打印出來了，先生最外層的HTML標籤，其次從head 標籤一個個剝離，以此類推。

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

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

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

and they lived at the bottom of a well.</p>

<p class="story">...</p>

</body></html>

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

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

The Dormouse's story

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

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

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

and they lived at the bottom of a well.</p>

<p class="story">...</p>

</body>

<p class="title" name=dromouse><b>The Dormouse's story</b></p>

<b>The Dormouse's story</b>

The Dormouse's story

<p class="story">Once upon a time there were three little sisters; and their names were

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

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

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

and they lived at the bottom of a well.</p>

Once upon a time there were three little sisters; and their names were

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

Elsie

,

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

Lacie

and

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

Tillie

;

and they lived at the bottom of a well.

<p class="story">...</p>

...

**（3）節點內容**

* 知識點：.string 屬性

如果tag只有一個NavigableString類型子節點,那麼這個tag可以使用.string得到子節點。如果一個tag僅有一個子節點,那麼這個tag也可以使用.string方法,輸出結果與當前唯一子節點的.string結果相同。

通俗點說就是：如果一個標籤裡面沒有標籤了，那麼.string 就會返回標籤裡面的內容。如果標籤裡面只有唯一的一個標籤了，那麼.string 也會返回最裡面的內容。例如:

print soup.head.string

#The Dormouse's story

print soup.title.string

#The Dormouse's story

如果tag包含了多個子節點，tag就無法確定，string 方法應該調用哪個子節點的內容， .string 的輸出結果是None。

print soup.html.string

# None

**（4）多個內容**

* **知識點： .strings .stripped\_strings 屬性**

**.strings**

獲取多個內容，不過需要遍歷獲取，比如下面的例子:

for string in soup.strings:

print(repr(string))

# u"The Dormouse's story"

# u'\n\n'

# u"The Dormouse's story"

# u'\n\n'

# u'Once upon a time there were three little sisters; and their names were\n'

# u'Elsie'

# u',\n'

# u'Lacie'

# u' and\n'

# u'Tillie'

# u';\nand they lived at the bottom of a well.'

# u'\n\n'

# u'...'

# u'\n'

**.stripped\_strings**

輸出的字符串中可能包含了很多空格或空行,使用.stripped\_strings可以去除多餘空白內容。

for string in soup.stripped\_strings:

print(repr(string))

# u"The Dormouse's story"

# u"The Dormouse's story"

# u'Once upon a time there were three little sisters; and their names were'

# u'Elsie'

# u','

# u'Lacie'

# u'and'

# u'Tillie'

# u';\nand they lived at the bottom of a well.'

# u'...'

**（5）父節點**

* **知識點： .parent 屬性**

p = soup.p

print p.parent.name

#body

content = soup.head.title.string

print content.parent.name

#title

**（6）全部父節點**

* **知識點：.parents 屬性**

通過元素的.parents屬性可以遞歸得到元素的所有父輩節點，例如:

content = soup.head.title.string

for parent in content.parents:

print parent.name

title

head

html

[document]

**（7）兄弟節點**

* **知識點：.next\_sibling .previous\_sibling 屬性**

兄弟節點可以理解為和本節點處在統一級的節點，.next\_sibling 屬性獲取了該節點的下一個兄弟節點，.previous\_sibling 則與之相反，如果節點不存在，則返回None

* 注意：實際文檔中的tag的.next\_sibling 和.previous\_sibling 屬性通常是字符串或空白，因為空白或者換行也可以被視作一個節點，所以得到的結果可能是空白或者換行

print soup.p.next\_sibling

# 實際該處為空白

print soup.p.prev\_sibling

#None 沒有前一個訂節點，返回None

print soup.p.next\_sibling.next\_sibling

#<p class="story">Once upon a time there were three little sisters; and their names were

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

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

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

#and they lived at the bottom of a well.</p>

#下一個節點的下一個兄弟節點是我們可以看到的節點

**（8）全部兄弟節點**

* **知識點：.next\_siblings .previous\_siblings 屬性**

通過.next\_siblings和.previous\_siblings屬性可以對當前節點的兄弟節點迭代輸出。

for sibling in soup.a.next\_siblings:

print(repr(sibling))

# u',\n'

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

# u' and\n'

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

# u'; and they lived at the bottom of a well.'

# None

**（9）前後節點**

* **知識點：.next\_element .previous\_element 屬性**

與 .next\_sibling .previous\_sibling 不同，它並不是針對於兄弟節點，而是在所有節點，不分層次。

比如head 節點為:

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

那麼它的下一個節點便是title，它是不分層次關係的。

print soup.head.next\_element

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

**（10）所有前後節點**

* **知識點：.next\_elements .previous\_elements 屬性**

通過.next\_elements和.previous\_elements的迭代器就可以向前或向後訪問文檔的解析內容,就好像文檔正在被解析一樣:

for element in last\_a\_tag.next\_elements:

print(repr(element))

# u'Tillie'

# u';\nand they lived at the bottom of a well.'

# u'\n\n'

# <p class="story">...</p>

# u'...'

# u'\n'

# None

7.搜索文檔樹

**（1）find\_all( name , attrs , recursive , text , \*\*kwargs )**

find\_all()方法搜索當前tag的所有tag子節點,並判斷是否符合過濾器的條件

**1）name 參數**

name參數可以查找所有名字為 name的tag,字符串對象會被自動忽略掉

**A.傳字符串**

最簡單的過濾器是字符串.在搜索方法中傳入一個字符串參數,Beautiful Soup會查找與字符串完整匹配的內容,下面的例子用於查找文檔中所有的<b>標籤:

soup.find\_all('b')

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

print soup.find\_all('a')

#[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>, <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

**B.傳正則表達式**

如果傳入正則表達式作為參數,Beautiful Soup會通過正則表達式的match()來匹配內容.下面例子中找出所有以b開頭的標籤,這表示<body>和<b>標籤都應該被找到。

import re

for tag in soup.find\_all(re.compile("^b")):

print(tag.name)

# body

# b

**C.傳列表**

如果傳入列表參數，Beautiful Soup會將與列表中任一元素匹配的內容返回，下面代碼找到文檔中所有<a>標籤和<b>標籤。

soup.find\_all(["a", "b"])

# [<b>The Dormouse's story</b>,

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

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

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

**D.傳True**

True可以匹配任何值,下面代碼查找到所有的tag,但是不會返回字符串節點。

for tag in soup.find\_all(True):

print(tag.name)

# html

# head

# title

# body

# p

# b

# p

# a

# a

**E.傳方法**

如果沒有合適過濾器,那麼還可以定義一個方法,方法只接受一個元素參數[[4]](http://www.crummy.com/software/BeautifulSoup/bs4/doc/index.zh.html" \l "id85" \o ") ,如果這個方法返回True表示當前元素匹配並且被找到,如果不是則反回False。

下面方法校驗了當前元素,如果包含class屬性卻不包含id屬性,那麼將返回True :

def has\_class\_but\_no\_id(tag):

return tag.has\_attr('class') and not tag.has\_attr('id')

將這個方法作為參數傳入find\_all()方法,將得到所有<p>標籤:

soup.find\_all(has\_class\_but\_no\_id)

# [<p class="title"><b>The Dormouse's story</b></p>,

# <p class="story">Once upon a time there were...</p>,

# <p class="story">...</p>]

**2）keyword 參數**

* **注意：如果一個指定名字的參數不是搜索內置的參數名,搜索時會把該參數當作指定名字tag的屬性來搜索,如果包含一個名字為id的參數,Beautiful Soup會搜索每個tag的”id ”屬性**

soup.find\_all(id='link2')

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

使用多個指定名字的參數可以同時過濾tag的多個屬性。

soup.find\_all(href=re.compile("elsie"), id='link1')

# [<a class="sister" href="http://example.com/elsie" id="link1">three</a>]

在這裡我們想用class 過濾，不過class 是python 的關鍵詞，這怎麼辦？加個下劃線就可以。

soup.find\_all("a", class\_="sister")

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

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

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

有些tag屬性在搜索不能使用,比如HTML5中的data-\* 屬性。

data\_soup = BeautifulSoup('<div data-foo="value">foo!</div>')

data\_soup.find\_all(data-foo="value")

# SyntaxError: keyword can't be an expression

但是可以通過find\_all()方法的attrs參數定義一個字典參數來搜索包含特殊屬性的tag。

data\_soup.find\_all(attrs={"data-foo": "value"})

# [<div data-foo="value">foo!</div>]

**3）text 參數**

通過text參數可以搜搜文檔中的字符串內容.與name參數的可選值一樣, text參數接受字符串,正則表達式,列表, True。

soup.find\_all(text="Elsie")

# [u'Elsie']

soup.find\_all(text=["Tillie", "Elsie", "Lacie"])

# [u'Elsie', u'Lacie', u'Tillie']

soup.find\_all(text=re.compile("Dormouse"))

[u"The Dormouse's story", u"The Dormouse's story"]

**4）limit 參數**

find\_all()方法返回全部的搜索結構,如果文檔樹很大那麼搜索會很慢.如果我們不需要全部結果,可以使用 limit參數限制返回結果的數量.效果與SQL中的limit關鍵字類似,當搜索到的結果數量達到limit的限制時,就停止搜索返回結果。

文檔樹中有3個tag符合搜索條件,但結果只返回了2個,因為我們限制了返回數量。

soup.find\_all("a", limit=2)

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

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

**5）recursive 參數**

調用tag的find\_all()方法時,Beautiful Soup會檢索當前tag的所有子孫節點,如果只想搜索tag的直接子節點,可以使用參數recursive=False 。

一段簡單的文檔:

<html>

<head>

<title>

The Dormouse's story

</title>

</head>

...

是否使用recursive參數的搜索結果:

soup.html.find\_all("title")

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

soup.html.find\_all("title", recursive=False)

# []

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

它與find\_all()方法唯一的區別是find\_all()方法的返回結果是值包含一個元素的列表,而find()方法直接返回結果。

**（3）find\_parents() find\_parent()**

find\_all()和 find()只搜索當前節點的所有子節點,孫子節點等. find\_parents()和 find\_parent()用來搜索當前節點的父輩節點,搜索方法與普通tag的搜索方法相同,搜索文檔搜索文檔包含的內容。

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

這2個方法通過.next\_siblings屬性對當tag的所有後面解析的兄弟tag節點進行迭代, find\_next\_siblings()方法返回所有符合條件的後面的兄弟節點, find\_next\_sibling()只返回符合條件的後面的第一個tag節點。

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

這2個方法通過.previous\_siblings屬性對當前tag的前面解析的兄弟tag節點進行迭代, find\_previous\_siblings()方法返回所有符合條件的前面的兄弟節點, find\_previous\_sibling()方法返回第一個符合條件的前面的兄弟節點。

**（6）find\_all\_next() find\_next()**

這2個方法通過.next\_elements屬性對當前tag的之後的tag和字符串進行迭代, find\_all\_next()方法返回所有符合條件的節點, find\_next()方法返回第一個符合條件的節點。

**（7）find\_all\_previous() 和find\_previous()**

這2個方法通過.previous\_elements屬性對當前節點前面的tag和字符串進行迭代, find\_all\_previous()方法返回所有符合條件的節點, find\_previous()方法返回第一個符合條件的節點。

* **注：以上（2）（3）（4）（5）（6）（7）方法參數用法與find\_all() 完全相同，原理均類似，在此不再贅述。**

8.CSS選擇器

我們在寫CSS時，標籤名不加任何修飾，類名前加點，id名前加#，在這裡我們也可以利用類似的方法來篩選元素，用到的方法是**soup.select()，**返回類型是**list。**

**（1）通過標籤名查找**

print soup.select('title')

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

print soup.select('a')

#[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>, <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

print soup.select('b')

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

**（2）通過類名查找**

print soup.select('.sister')

#[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>, <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

**（3）通過id 名查找**

print soup.select('#link1')

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

**（4）組合查找**

組合查找即和寫class 文件時，標籤名與類名、id名進行的組合原理是一樣的，例如查找p 標籤中，id 等於link1的內容，二者需要用空格分開。

print soup.select('p #link1')

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

直接子標籤查找:

print soup.select("head > title")

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

**（5）屬性查找**

查找時還可以加入屬性元素，屬性需要用中括號括起來，注意屬性和標籤屬於同一節點，所以中間不能加空格，否則會無法匹配到。

print soup.select('a[class="sister"]')

#[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>, <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

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

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

同樣，屬性仍然可以與上述查找方式組合，不在同一節點的空格隔開，同一節點的不加空格。

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

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

以上的select 方法返回的結果都是列表形式，可以遍歷形式輸出，然後用get\_text() 方法來獲取它的內容。

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()

好，這就是另一種與find\_all 方法有異曲同工之妙的查找方法，是不是感覺很方便？

**FAQ**

* 寫爬蟲有一個問題很多人會來問我「奇怪，我的網頁程式碼有看到，為何爬蟲卻無法抓到？」，這個原因很簡單，因為你看到的是「**[假的](https://www.youtube.com/watch?v=pH75p71HTII" \t "_blank)**」（無誤），是因為網頁內容是動態產生的（以用Javascript產生的內容為例），你所看到的是經過瀏覽器已經將Javascript執行後產生的結果，而一般爬蟲預設是沒有執行這些Javascript，所以當然抓不到，我們要如何判別呢？在Chrome當中如果是右鍵按下[檢視網頁原始碼]，則你所看到的HTML是沒有動態產生內容的，也就是我們一般爬蟲預設抓到的資料；在網頁上游標移動到其中一個元素按右鍵點[檢查]所看到的HTML內容則是經過動態產生的結果，一般爬蟲沒有特別處理是無法得到這些內容的。 這問題有沒有解？答案是有的，需要用其他方式例如讓爬蟲**先模擬瀏覽器取得動態產生內容後**，再去取得分析HTML的方式解決。