Beautiful Soup 库

Beautiful Soup 库是能够解析 HTML 或 XML 文件的功能库

```
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://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>"""
```

例如每一段 HTML 代码都是由尖括号组成的“标签树”， Beautiful Soup 库能够解析、遍历、维护这个“标签树”

```
from bs4 import BeautifulSoup
import requests

r = requests.get("http://www.baidu.com")
soup = BeautifulSoup(r.text, "html.parser")
print(soup.prettify())
```

```
<!DOCTYPE html>
<!--STATUS OK-->
<html>
 <head>
  <meta content="text/html;charset=utf-8" http-equiv="content-type">
   <meta content="IE=Edge" http-equiv="X-UA-Compatible">
    <meta content="always" name="referrer">
     <link     href="http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css"
rel="stylesheet" type="text/css">
      <title>
       ç□¾å°¦ä，  ä，  ï¼  ä½ å°±ç□¥é
      </title>
     </link>
    </meta>
   </meta>
  </meta>
 </head>
 <body link="#0000cc">
  <div id="wrapper">
   <div id="head">
    <div class="head_wrapper">
     <div class="s_form">
      <div class="s_form_wrapper">
       <div id="lg">
        <img                    height="129"                    hidefocus="true"
src="//www.baidu.com/img/bd_logo1.png" width="270">
       </img>
```

```
      </div>
      <form action="//www.baidu.com/s" class="fm" id="form" name="f">
       <input name="bdorz_come" type="hidden" value="1">
        <input name="ie" type="hidden" value="utf-8">
         <input name="f" type="hidden" value="8">
          <input name="rsv_bp" type="hidden" value="1">
           <input name="rsv_idx" type="hidden" value="1">
            <input name="tn" type="hidden" value="baidu">
             <span class="bg s_ipt_wr">
              <input autocomplete="off" autofocus="" class="s_ipt" id="kw"
maxlength="255" name="wd" value=""/>
             </span>
             <span class="bg s_btn_wr">
              <input class="bg s_btn" id="su" type="submit" value="ç□¾åº¦ä,
ä, "/>
             </span>
            </input>
           </input>
          </input>
         </input>
        </input>
       </form>
      </div>
     </div>
     <div id="u1">
      <a class="mnav" href="http://news.baidu.com" name="tj_trnews">
       æ□° é »
      </a>
      <a class="mnav" href="http://www.hao123.com" name="tj_trhao123">
       hao123
      </a>
      <a class="mnav" href="http://map.baidu.com" name="tj_trmap">
       å□° å□¾
      </a>
      <a class="mnav" href="http://v.baidu.com" name="tj_trvideo">
       è§ é¢□
      </a>
      <a class="mnav" href="http://tieba.baidu.com" name="tj_trtieba">
       è´´å□§
      </a>
      <noscript>
       <a                                                    class="lb"
href="http://www.baidu.com/bdorz/login.gif?login&amp;tpl=mn&amp;u=http%3A%
2F%2Fwww.baidu.com%2f%3fbdorz_come%3d1" name="tj_login">
        ç□»å½
       </a>
```

```
      </noscript>
      <script>
       document.write('<a
href="http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u='+
encodeURIComponent(window.location.href+ (window.location.search === "" ?
"?" : "&")+ "bdorz_come=1")+ '" name="tj_login" class="lb">ç□»å½ </a>');
      </script>
      <a    class="bri"    href="//www.baidu.com/more/"    name="tj_briicon"
style="display: block;">
       æ□´å□  äº§å□□
      </a>
     </div>
    </div>
   </div>
   <div id="ftCon">
    <div id="ftConw">
     <p id="lh">
      <a href="http://home.baidu.com">
       å□³äº  ç□¾åº¦
      </a>
      <a href="http://ir.baidu.com">
       About Baidu
      </a>
     </p>
     <p id="cp">
      ©2017 Baidu
      <a href="http://www.baidu.com/duty/">
       ä½¿ç□¨ç□¾åº¦å□□å¿  è¯»
      </a>
      <a class="cp-feedback" href="http://jianyi.baidu.com/">
       æ□□è§  å□□é¦□
      </a>
      äº¬ICPè¯  030173å□•
      <img src="//www.baidu.com/img/gs.gif">
      </img>
     </p>
    </div>
   </div>
  </div>
 </body>
</html>


Process finished with exit code 0
```

注意代码 soup = BeautifulSoup(r.text, "html.parser") 是将用 requests.get 方法获得的服务器返回给我们 HTML 文件（也就是 r.text）转换成 BeautifulSoup 类，这样我们就可以对这个 HTML 进行解析了。

可以将 HTML 文件、标签树、BeautifulSoup 类理解为一一对应的关系，转换成 soup 变量，我们就可以进行一系

列操作了，例如如下操作
soup.prettify()就是将 HTML 文件进行更好的展示的函数，使得每一个标签对（如 ... ）都能对齐，中间即为内容，这样看起来更方便

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc)

print(soup.prettify())
# <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://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="link2">
#     Tillie
#    </a>
#    ; and they lived at the bottom of a well.
#   </p>
#   <p class="story">
#    ...
#   </p>
#  </body>
# </html>
```

这是开头文档的 prettify()之后的形式

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。

```
Tag: <a href="http://ir.baidu.com">
       About Baidu
      </a>
```

这个就是一段 Tag，name 为 a

Tag

`Tag` 对象与XML或HTML原生文档中的tag相同:

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

Tag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.现在介绍一下tag中最重要的属性: name和attributes

Name

每个tag都有自己的名字,通过 `.name` 来获取:

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

如果改变了tag的name,那将影响所有通过当前Beautiful Soup对象生成的HTML文档:

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

注意第一段代码，如果这段 HTML 文字中有多个标签 b，那么 soup.b 只返回第一个

Attributes

一个tag可能有很多个属性. tag `<b class="boldest">` 有一个 “class” 的属性,值为 “boldest” . tag的属性的操作方法与字典相同:

```
tag['class']
# u'boldest'
```

也可以直接”点”取属性, 比如: `.attrs` :

```
tag.attrs
# {u'class': u'boldest'}
```

tag的属性可以被添加,删除或修改. 再说一次, tag的属性操作方法与字典一样

```
tag['class'] = 'verybold'
tag['id'] = 1
tag
# <blockquote class="verybold" id="1">Extremely bold</blockquote>

del tag['class']
del tag['id']
tag
# <blockquote>Extremely bold</blockquote>

tag['class']
# KeyError: 'class'
print(tag.get('class'))
# None
```

Attributes 是多值属性的意思

多值属性

HTML 4定义了一系列可以包含多个值的属性.在HTML5中移除了一些,却增加更多.最常见的多值的属性是 class (一个tag可以有多个CSS的class). 还有一些属性 `rel` , `rev` , `accept-charset` , `headers` , `accesskey` . 在Beautiful Soup中多值属性的返回类型是list:

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

css_soup = BeautifulSoup('<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

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

将tag转换成字符串时,多值属性会合并为一个值

```
rel_soup = BeautifulSoup('<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tag中不包含多值属性

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

可以遍历的字符串

字符串常被包含在tag内.Beautiful Soup用 `NavigableString` 类来包装tag中的字符串:

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

一个 `NavigableString` 字符串与Python中的Unicode字符串相同,并且还支持包含在 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:

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

tag中包含的字符串不能编辑,但是可以被替换成其它的字符串,用 replace_with() 方法:

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

.string

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:

```
title_tag.string
# u'The Dormouse's story'
```

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:

```
head_tag.contents
# [<title>The Dormouse's story</title>]

head_tag.string
# u'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` :

```
print(soup.html.string)
# None
```

.strings 和 stripped_strings

如果tag中包含多个字符串 [2] ,可以使用 `.strings` 来循环获取:

```
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'
```

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:

```
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'...'
```

全部是空格的行会被忽略掉,段首和段末的空白会被删除

过滤器

介绍 `find_all()` 方法前，先介绍一下过滤器的类型 [3] ，这些过滤器贯穿整个搜索的API. 过滤器可以被用在tag的name中，节点的属性中，字符串中或他们的混合中.

字符串 ¶

最简单的过滤器是字符串. 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<b>标签:

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

如果传入字节码参数，Beautiful Soup会当作UTF-8编码，可以传入一段Unicode 编码来避免Beautiful Soup解析编码出错

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<a>标签和<b>标签:

```
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>]
```

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

```
for tag in soup.find_all(True):
    print(tag.name)
# html
# head
# title
# body
# p
# b
# p
# a
# a
# a
# p
```

find_all()

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. 这里有几个例子:

```
soup.find_all("title")
# [<title>The Dormouse's story</title>]

soup.find_all("p", "title")
# [<p class="title"><b>The Dormouse's story</b></p>]

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>]

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

import re
soup.find(text=re.compile("sisters"))
# u'Once upon a time there were three little sisters; and their names were\n'
```

有几个方法很相似，还有几个方法是新的，参数中的 `text` 和 `id` 是什么含义？为什么 `find_all("p", "title")` 返回的是CSS Class为”title”的<p>标签？我们来仔细看一下 `find_all()` 的参数

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.

简单的用法如下:

```
soup.find_all("title")
# [<title>The Dormouse's story</title>]
```

重申: 搜索 `name` 参数的值可以使任一类型的 过滤器 ,字符窜,正则表达式,列表,方法或是 `True` .

keyword 参数

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.

```
soup.find_all(id='link2')
# [<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:

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

搜索指定名字的属性时可以使用的参数值包括 字符串 , 正则表达式 , 列表, True .

下面的例子在文档树中查找所有包含 `id` 属性的tag,无论 `id` 的值是什么:

```
soup.find_all(id=True)
# [<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>]
```

使用多个指定名字的参数可以同时过滤tag的多个属性:

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

有些tag属性在搜索不能使用,比如HTML5中的 data-* 属性:

```
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:

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

find()

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

`find_all()` 方法将返回文档中符合条件的所有tag,尽管有时候我们只想得到一个结果.比如文档中只有一个<body>标签,那么使用 `find_all()` 方法来查找<body>标签就不太合适, 使用 `find_all` 方法并设置 `limit=1` 参数不如直接使用 `find()` 方法.下面两行代码是等价的:

```
soup.find_all('title', limit=1)
# [<title>The Dormouse's story</title>]

soup.find('title')
# <title>The Dormouse's story</title>
```

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.

`find_all()` 方法没有找到目标是返回空列表, `find()` 方法找不到目标时,返回 `None` .

```
print(soup.find("nosuchtag"))
# None
```

`soup.head.title` 是 tag的名字 方法的简写.这个简写的原理就是多次调用当前tag的 `find()` 方法:

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

soup.find("head").find("title")
# <title>The Dormouse's story</title>
```

注意 find()只返回一个结果，且返回的类型是<class 'bs4.element.Tag'>
find_all()返回的是一个列表，返回的类型是<class 'bs4.element.ResultSet'>
Tag 就可以继续使用 find()和 find_all()函数