**BeautifulSoup Html解析**

# 基本使用

## 解析器的使用

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | 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 |

## bs4的使用

3种正确的打开姿势：

import requests  
from bs4 import BeautifulSoup  
  
print("-----------------------打开html文档，注意编码-----------------------")  
soup1 = BeautifulSoup(open("index.html", encoding='UTF-8'), "lxml")  
print(soup1.contents)  
  
print("-----------------------直接处理html元素-----------------------")  
soup2 = BeautifulSoup("<html>data测试</html>", "lxml")  
print(soup2.contents)  
  
print("-----------------------解析爬取的html-----------------------")  
url = "http://127.0.0.1:5000/testcss/indexcss"  
r = requests.get(url)  
# 第一个参数：html 标记, 第二个参数：解释器名称  
soup3 = BeautifulSoup(r.text, "lxml")  
print(soup3.contents)

# 对象的种类

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

## Tag

from bs4 import BeautifulSoup  
  
soup = BeautifulSoup('<div class="test1">测试html</div>', "lxml")  
# 获取tag 对象  
tag = soup.div  
print(type(tag))

输出tag的类型

## Name

from bs4 import BeautifulSoup  
  
soup = BeautifulSoup('<div class="test1">测试html</div>', "lxml")  
# 获取tag 对象  
tag = soup.div  
# 输出name  
print(tag.name)  
# 修改name  
tag.name = "p"  
print(tag)

## Attributes

from bs4 import BeautifulSoup  
  
soup = BeautifulSoup('<div class="test1">测试html</div>', "lxml")  
# 获取tag 对象  
tag = soup.div  
  
# 一个tag可能有很多个属性. tag <div class="test1"> 有一个 “class” 的属性,值为 test1 . tag的属性的操作方法与字典相同  
print(tag['class'])  
# 直接获取属性 返回一个字典  
print(tag.attrs)

了解：tag的属性也可以修改，修改方式就跟修改字典是一样的

## 一个属性有多个值

属性有多个值的情况：style class 以及自定义属性

from bs4 import BeautifulSoup  
  
soup = BeautifulSoup('<div class="test1 test2">测试html</div>', "lxml")  
# 获取tag 对象  
tag = soup.div  
  
# 返回一个属性值的列表  
print(tag['class'])

属性只能是单个值的情况，返回的就不是列表

id\_soup = BeautifulSoup('<div id="my id"></div>', "lxml")  
#这个不是一个列表  
print(id\_soup.div['id'])

原因：

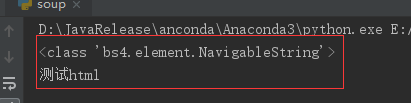
在任何版本的HTML定义中都没有被定义为多值属性, 那么Beautiful Soup会将这个属性作为字符串返回

## 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.那就意味着，tag的字符串拥有NavigableString 类函数

from bs4 import BeautifulSoup  
  
html = '<div class="test1 test2">测试html</div>'  
soup = BeautifulSoup(html, "lxml")  
tag = soup.div  
print(type(tag.string))  
print(tag.string)

输出：



无效的情况：

一个元素内部，还包含了其他元素，这是用string不能取出值。tag.string只能针对元素里面，全是字符串的情况，不能包含元素

html = '<div class="test1 test2">测试html<p rel="test">测试内部的p</p></div>'  
soup = BeautifulSoup(html, "lxml")  
tag = soup.div  
print(type(tag.string))  
print(tag.string)

## BeautifulSoup说明

soup对象，包含的是整个html，因此，针对的tag的操作，并不适用于soup

## 注释及特殊字符串

from bs4 import BeautifulSoup  
  
html = '<div class="test1 test2"><!--hello 这是一段注释?--></div>'  
soup = BeautifulSoup(html, "lxml")  
tag = soup.div  
# 输出 class 'bs4.element.Comment' 表示这是一段注释  
print(type(tag.string))  
# 输出注释里面的内容  
print(tag.string)

注意：

html文档中，以下条件中满足之一

1. div中string不能包含其他内容，只能是一段很纯粹的文本
2. div中包含一个tag元素，这个tag还不能嵌套其他tag
3. 要么是一段注释

总结：string 就表示字符串，不能和其他元素混搭

## 文档格式化输出

from bs4 import BeautifulSoup  
  
html = '<div class="test1 test2"><!--hello 这是一段注释?--></div>'  
soup = BeautifulSoup(html, "lxml")  
tag = soup.div  
  
print(tag.prettify())

# 遍历文档树

## 子节点

划重点：

一个Tag可能包含多个字符串或其它的Tag,这些都是这个Tag的子节点.

Beautiful Soup提供了许多操作和遍历子节点的属性.

注意: Beautiful Soup中字符串节点不支持这些属性,因为字符串没有子节点

### tag的名字

通过名字获取元素

soup = BeautifulSoup(html, "lxml")  
# 获取整个head  
tag = soup.head  
# 获取整个body  
tag = soup.body  
print(tag.prettify())

逐级获取

soup = BeautifulSoup(html, "lxml")  
# 通过文档树调用，逐级获取tag  
tag = soup.body.span  
tag = soup.body.div.p  
print(tag.prettify())

通过点取属性的方式只能获得当前名字的第一个tag

soup = BeautifulSoup(html, "lxml")  
tag = soup.p  
print(tag.prettify())

通过find\_all获取所有指定标签

soup = BeautifulSoup(html, "lxml")  
tag =soup.find\_all('p')  
print(tag)

### .contents 和 .children

tag的 .contents 属性可以将tag的【直接】子节点以列表的方式输出

soup = BeautifulSoup(html, "lxml")  
tag = soup.div  
print(tag.contents)

通过tag的 .children 生成器,可以对tag的子节点进行循环

soup = BeautifulSoup(html, "lxml")  
tag = soup.div  
  
for child in tag.children:  
 print(child)

### .descendants

tag的 . descendants属性可以将tag的【所有】子节点以列表的方式输出

soup = BeautifulSoup(html, "lxml")  
tag = soup.div  
  
for child in tag.descendants:  
 print(child)

### .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

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

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

### .strings 和 stripped\_strings

如果tag中包含多个字符串,可以使用 .strings 来循环获取：

soup = BeautifulSoup(html, "lxml")  
tag = soup.div  
  
for string in tag.strings:  
 print(repr(string))

排除其中的空格 换行等

soup = BeautifulSoup(html, "lxml")  
tag = soup.div  
  
for string in tag.stripped\_strings:  
 print(repr(string))

## 父节点

### .parent

通过 .parent 属性来获取某个元素的父节点

soup = BeautifulSoup(html, "lxml")  
# 注意，这里有两个span，返回的是前一个，这个前后是指代码编写的前后，而不是层级的上下  
tag = soup.div.span  
print(tag.parent)

### .parents

获取元素的所有父节点，知道返回none

soup = BeautifulSoup(html, "lxml")  
tag = soup.div.span  
  
for parent in tag.parents:  
 if parent is None:  
 print(parent)  
 else:  
 print(parent.name)

## 兄弟节点

### .next\_sibling 和 .previous\_sibling

获取文档第一个p的后面一个p

soup = BeautifulSoup(html, "lxml")  
tag = soup.p.next\_sibling  
print(tag.prettify())

获取div的前面一个

soup = BeautifulSoup(html, "lxml")  
tag = soup.div.previous\_sibling  
print(tag.prettify())

严重注意:别被你的眼睛欺骗了

<span>这是一个span</span><div>这是一个span</div><p>这是一个P</p><b>这是文档上的第二个 P</b><t>这是二个P</t>

与

<span>这是一个span</span>  
<div>这是一个span</div>  
<p>这是一个P</p><b>这是文档上的第二个 P</b>  
<t>这是二个P</t>

的区别：

第二段代码，div前是换行符，换行符是一个字符串，在使用的时候，是不能调用香相关的api的

### .next\_siblings 和 .previous\_siblings

soup = BeautifulSoup(html, "lxml")  
print('----------------------查找所有后续元素-------------')  
for sibling in soup.div.next\_siblings:  
 print(repr(sibling))  
  
print('----------------------查找id所有前面的元素-------------')  
for sibling in soup.find(id="test").previous\_siblings:  
 print(repr(sibling))

## 回退和前进

### .next\_element 和 .previous\_element

注意区别：

from bs4 import BeautifulSoup  
  
html = '''  
  
<a href="http://example.com/elsie" class="sister" id="link1">第1个链接</a>  
<a href="http://example.com/lacie" class="sister" id="link2">第2个链接</a> 测试不正规的html文档1  
<a href="http://example.com/tillie" class="sister" id="link3">第3个链接</a>;  
测试不正规的html文档2.</p>  
  
  
  
'''  
soup = BeautifulSoup(html, "lxml")  
last\_a\_tag = soup.find("a", id="link1")  
# 取的第1个连接的内容  
print('-这里取当前元素的内容----------------------')  
print(last\_a\_tag.next\_element)  
print('-这里不输出内容，因为有换行----------------------')  
print(last\_a\_tag.next\_element.next\_element)  
print('-这里前进到第2行，输出link2----------------------')  
print(last\_a\_tag.next\_element.next\_element.next\_element)  
print('-对比：next\_sibling----------------------')  
# 这里next\_sibling 会直接找下一个元素，不会当前元素内部，因此输出回车行  
print(last\_a\_tag.next\_sibling)  
print('-这里next\_sibling将取到第2行元素----------------------')  
print(last\_a\_tag.next\_sibling.next\_sibling)

previous\_element与之相反

### .next\_elements 和 .previous\_elements

按上一步的规则取完

last\_a\_tag = soup.find("a", id="link1")  
for element in last\_a\_tag.next\_elements:  
 print(repr(element))

# 搜索文档树

## 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<div>标签

补充：查找的是元素，不是内容

from bs4 import BeautifulSoup  
  
html = '''  
  
<html lang="en">  
<head>  
 <meta charset="utf-8"/>  
 <title>Title</title>  
 <style type="text/css">  
 span + p {  
 background: black;  
 color: red;  
 }  
 </style>  
</head>  
<body>  
<span>这是一个span</span>  
<div>这是一个span</div>  
<div>这是二个span</div>  
<p>这是一个P</p><b>这是文档上的第二个 P</b>  
<t>这是二个P</t>  
</body>  
</html>  
  
  
'''  
soup = BeautifulSoup(html, "lxml")  
#返回所有的div  
tags = soup.find\_all('div')  
print(tags)

## 正则表达式

soup = BeautifulSoup(html, "lxml")  
import re

#标签中b开头的   
for tag in soup.find\_all(re.compile("^b")):  
 print(tag.name)

soup = BeautifulSoup(html, "lxml")  
import re  
#标签中包含t 的   
for tag in soup.find\_all(re.compile("t")):  
 print(tag.name)

## 列表

一次查找多个元素

soup = BeautifulSoup(html, "lxml")  
tags = soup.find\_all(["span", "div"])  
for i in tags:  
 print(i)

补充：find查找顺序：

按深度优先遍历算法查找

## True

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

查找所有元素

## 外部方法

from bs4 import BeautifulSoup  
  
html = '''  
  
<html lang="en">  
<head>  
 <meta charset="utf-8"/>  
 <title>Title</title>  
 <style type="text/css">  
 span + p {  
 background: black;  
 color: red;  
 }  
 </style>  
</head>  
<body>  
<span class="test" id="testspan">这是一个span</span>  
<div>这是一个span <div>这是三个span</div> </div>  
<div class="test">这是二个span</div>  
<p>这是一个P</p><b>这是文档上的第二个 P</b>  
<t>这是二个P</t>  
</body>  
</html>  
  
  
'''  
soup = BeautifulSoup(html, "lxml")  
  
def has\_class\_but\_no\_id(tag):  
 return tag.has\_attr('class') and not tag.has\_attr('id')  
  
tags = soup.find\_all(has\_class\_but\_no\_id)  
for i in tags:  
 print(i)

## fina\_all根据文档内容查找

参数使用text即可

soup = BeautifulSoup(html, "lxml")  
  
import re  
  
tags = soup.find\_all(text=re.compile("二个"))  
for i in tags:  
 print(i)

## keyword 参数

### 属性名称+对应的属性值

soup = BeautifulSoup(html, "lxml")  
  
tags = soup.find\_all(id="testspan")  
for i in tags:  
 print(i)

soup = BeautifulSoup(html, "lxml")  
  
tags = soup.find\_all(attr="hello")  
for i in tags:  
 print(i)

### 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

soup.find\_all(id=True)

### 使用多个指定名字的参数可以同时过滤tag的多个属性

soup = BeautifulSoup(html, "lxml")  
  
tags = soup.find\_all(attr=re.compile("hello"), id='p1')  
for i in tags:  
 print(i)

### 定义一个字典参数来搜索包含特殊属性的tag

soup = BeautifulSoup(html, "lxml")  
  
dic={"attr": "hello"}  
tags = soup.find\_all(attrs=dic)  
for i in tags:  
 print(i)

## 按CSS搜索

### 按照CSS类名搜索

### 注意class关键词

soup = BeautifulSoup(html, "lxml")  
tags = soup.find\_all('div',class\_="test")  
for i in tags:  
 print(i)

### class\_ 参数同样接受不同类型的 过滤器 ,字符串,正则表达式,方法或 True

通过正则查找

#通过正则查找  
tags = soup.find\_all(class\_=re.compile("es"))

调用外部函数

soup = BeautifulSoup(html, "lxml")  
  
  
def has\_six\_characters(css\_class):  
 return css\_class is not None and len(css\_class) == 5  
  
tags = soup.find\_all(class\_=has\_six\_characters)  
for i in tags:  
 print(i)

## text参数

全匹配

soup = BeautifulSoup(html, "lxml")  
  
# 匹配文字内容，注意：这里是全部匹配，条件是或  
tags = soup.find\_all(text=["这是二个span", '这是文档上的第二个 P'])  
for i in tags:  
 print(i)

正则：

soup = BeautifulSoup(html, "lxml")  
  
# 用正则就是模糊查找  
tags = soup.find\_all(text=re.compile("二个"))  
for i in tags:  
 print(i)

## 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.

soup = BeautifulSoup(html, "lxml")  
  
tags = soup.find\_all('div',limit=1)  
for i in tags:  
 print(i)

## recursive参数

soup = BeautifulSoup(html, "lxml")  
  
# 查找直接子节点  
# 注意;在使用的时候，需要先 用 soup.body 找到某个tag，然后找对应tag的子节点  
tags = soup.body.find\_all('div', recursive=False)  
for i in tags:  
 print(i)

## find()

soup = BeautifulSoup(html, "lxml")  
  
#find 链式调用 每次选择，只返回列表中的第一个  
tags = soup.find("head").find("title")  
for i in tags:  
 print(i)

## find\_parents() 和 find\_parent()

找直接父元素与找所有父元素

soup = BeautifulSoup(html, "lxml")  
  
#find 链式调用 每次选择，只返回列表中的第一个  
tags = soup.find("div",class\_="test5").find\_parent()  
for i in tags:  
 print(i)

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

往后找同一级所有的临近节点

soup = BeautifulSoup(html, "lxml")  
  
#find 链式调用 每次选择，只返回列表中的第一个  
tags = soup.find("div",class\_="test").find\_next\_siblings()  
for i in tags:  
 print(i)

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

往后找同一级所有的临近节点

soup = BeautifulSoup(html, "lxml")  
  
#find 链式调用 每次选择，只返回列表中的第一个  
tags = soup.find("div",class\_="test").find\_previous\_siblings()  
for i in tags:  
 print(i)

## find\_all\_next() 和 find\_next()

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

内部调用的next\_element

soup = BeautifulSoup(html, "lxml")  
  
tags = soup.find("div",class\_="test").find\_all\_next('p')  
for i in tags:  
 print(i)

## find\_all\_previous() 和 find\_previous()

往前找：包含所有层级

soup = BeautifulSoup(html, "lxml")  
  
tags = soup.find("div",class\_="test").find\_all\_previous('div')  
for i in tags:  
 print(i)

## CSS选择器

Beautiful Soup支持大部分的CSS选择器,在 Tag 或 BeautifulSoup 对象的 .select() 方法中传入字符串参数,即可使用CSS选择器的语法找到tag

soup = BeautifulSoup(html, "lxml")  
  
tags = soup.select("head > title")  
tags = soup.select(".test1")  
tags = soup.select("p[attr]")  
tags = soup.select("#testspan")  
tags = soup.select(".test1 .test5")  
for i in tags:  
 print(i)

# 解析部分文档提高性能

## SoupStrainer

如果仅仅因为想要查找文档中的<div>标签而将整片文档进行解析,浪费内存和时间.

最快的方法是从一开始就把<div>标签以外的东西都忽略掉.

SoupStrainer 类可以定义文档的某段内容,这样搜索文档时就不必先解析整篇文档,只会解析在 SoupStrainer 中定义过的文档.

创建一个 SoupStrainer 对象并作为 parse\_only 参数给 BeautifulSoup 的构造方法即可

## 传给BeautifulSoup

divs = SoupStrainer("div")  
soup=BeautifulSoup(html, "html.parser", parse\_only=divs)  
print(soup.prettify())

## 传给find函数

soup = BeautifulSoup(html, "lxml")  
  
divs = SoupStrainer("div")  
  
tags = soup.find\_all(divs)  
for i in tags:  
 print(i)