**4. 创建 Beautiful Soup 对象**

首先必须要导入 bs4 库

from bs4 import BeautifulSoup

我们创建一个字符串，后面的例子我们便会用它来演示

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)

另外，我们还可以用本地 HTML 文件来创建对象，例如

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

上面这句代码便是将本地 index.html 文件打开，用它来创建 soup 对象

下面我们来打印一下 soup 对象的内容，格式化输出

print soup.prettify()

<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>

以上便是输出结果，格式化打印出了它的内容，这个函数经常用到，小伙伴们要记好咯。  
**5. 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:

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

下面我们进行一一介绍  
**（1）Tag**

Tag 是什么？通俗点讲就是 HTML 中的一个个标签，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 标签加上里面包括的内容就是 Tag，下面我们来感受一下怎样用 Beautiful Soup 来方便地获取 Tags

下面每一段代码中注释部分即为运行结果

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>

我们可以利用 soup加标签名轻松地获取这些标签的内容，是不是感觉比正则表达式方便多了？不过有一点是，它查找的是在所有内容中的第一个符合要求的标签，如果要查询所有的标签，我们在后面进行介绍。

我们可以验证一下这些对象的类型

print type(soup.a)

#<class 'bs4.element.Tag'>

对于 Tag，它有两个重要的属性，是 name 和 attrs，下面我们分别来感受一下

name

print soup.name

print soup.head.name

#[document]

#head

soup 对象本身比较特殊，它的 name 即为 [document]，对于其他内部标签，输出的值便为标签本身的名称。

attrs

print soup.p.attrs

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

在这里，我们把 p 标签的所有属性打印输出了出来，得到的类型是一个字典。

如果我们想要单独获取某个属性，可以这样，例如我们获取它的 class 叫什么

print soup.p['class']

#['title']

还可以这样，利用get方法，传入属性的名称，二者是等价的

print soup.p.get('class')

#['title']

我们可以对这些属性和内容等等进行修改，例如

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

print soup.p

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

还可以对这个属性进行删除，例如

del soup.p['class']

print soup.p

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

不过，对于修改删除的操作，不是我们的主要用途，在此不做详细介绍了，如果有需要，请查看前面提供的官方文档  
**（2）NavigableString**

既然我们已经得到了标签的内容，那么问题来了，我们要想获取标签内部的文字怎么办呢？很简单，用 .string 即可，例如

print soup.p.string

#The Dormouse's story

这样我们就轻松获取到了标签里面的内容，想想如果用正则表达式要多麻烦。它的类型是一个 NavigableString，翻译过来叫 可以遍历的字符串，不过我们最好还是称它英文名字吧。

print type(soup.p.string)

#<class 'bs4.element.NavigableString'>

来检查一下它的类型

print type(soup.p.string)

#<class 'bs4.element.NavigableString'>

**（3）BeautifulSoup**

BeautifulSoup 对象表示的是一个文档的全部内容.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，以及属性来感受一下

print type(soup.name)

#<type 'unicode'>

print soup.name

# [document]

print soup.attrs

#{} 空字典

**（4）Comment**

Comment 对象是一个特殊类型的 NavigableString 对象，其实输出的内容仍然不包括注释符号，但是如果不好好处理它，可能会对我们的文本处理造成意想不到的麻烦。

我们找一个带注释的标签

print soup.a

print soup.a.string

print type(soup.a.string)

运行结果如下

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

Elsie

<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，我们发现它已经把注释符号去掉了，所以这可能会给我们带来不必要的麻烦。

另外我们打印输出下它的类型，发现它是一个 Comment 类型，所以，我们在使用前最好做一下判断，判断代码如下

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

print soup.a.string

上面的代码中，我们首先判断了它的类型，是否为 Comment 类型，然后再进行其他操作，如打印输出。  
**6. 遍历文档树  
（1）直接子节点**

    要点：.contents .children 属性

.contents

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

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

输出方式为列表，我们可以用列表索引来获取它的某一个元素

print soup.head.contents[0]

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

.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

#<listiterator object at 0x7f71457f5710>

我们怎样获得里面的内容呢？很简单，遍历一下就好了，代码及结果如下

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）所有子孙节点**

    知识点：.descendants 属性

**.descendants**

.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

运行结果如下，可以发现，所有的节点都被打印出来了，先生最外层的 HTML标签，其次从 head 标签一个个剥离，以此类推。

<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）节点内容**

    知识点：.string 属性

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。

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

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

**（4）多个内容**

    知识点： .strings .stripped\_strings 属性

**.strings**

获取多个内容，不过需要遍历获取，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**

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'...'

**（5）父节点**

    知识点： .parent 属性

p = soup.p

print p.parent.name

#body

content = soup.head.title.string

print content.parent.name

#title

**（6）全部父节点**

    知识点：.parents 属性

通过元素的 .parents 属性可以递归得到元素的所有父辈节点，例如

content = soup.head.title.string

for parent in content.parents:

print parent.name

title

head

html

[document]

**（7）兄弟节点**

    知识点：.next\_sibling .previous\_sibling 属性

兄弟节点可以理解为和本节点处在统一级的节点，.next\_sibling 属性获取了该节点的下一个兄弟节点，.previous\_sibling 则与之相反，如果节点不存在，则返回 None

注意：实际文档中的tag的 .next\_sibling 和 .previous\_sibling 属性通常是字符串或空白，因为空白或者换行也可以被视作一个节点，所以得到的结果可能是空白或者换行

print soup.p.next\_sibling

# 实际该处为空白

print soup.p.prev\_sibling

#None 没有前一个兄弟节点，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>

#下一个节点的下一个兄弟节点是我们可以看到的节点

**（8）全部兄弟节点**

    知识点：.next\_siblings .previous\_siblings 属性

通过 .next\_siblings 和 .previous\_siblings 属性可以对当前节点的兄弟节点迭代输出

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）前后节点**

    知识点：.next\_element .previous\_element 属性

与 .next\_sibling .previous\_sibling 不同，它并不是针对于兄弟节点，而是在所有节点，不分层次

比如 head 节点为  
   
<head><title>The Dormouse's story</title></head>

那么它的下一个节点便是 title，它是不分层次关系的

print soup.head.next\_element

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

**（10）所有前后节点**

    知识点：.next\_elements .previous\_elements 属性

通过 .next\_elements 和 .previous\_elements 的迭代器就可以向前或向后访问文档的解析内容,就好像文档正在被解析一样

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.搜索文档树  
（1）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

**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**

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>]

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.传正则表达式**

如果传入正则表达式作为参数,Beautiful Soup会通过正则表达式的 match() 来匹配内容.下面例子中找出所有以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)

# body

# b

**C.传列表**

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>]

**D.传 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

**E.传方法**

如果没有合适过滤器,那么还可以定义一个方法,方法只接受一个元素参数 [4] ,如果这个方法返回 True 表示当前元素匹配并且被找到,如果不是则反回 False

下面方法校验了当前元素,如果包含 class 属性却不包含 id 属性,那么将返回 True:

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

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

将这个方法作为参数传入 find\_all() 方法,将得到所有<p>标签:

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 参数**

    注意：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>]

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>]

在这里我们想用 class 过滤，不过 class 是 python 的关键词，这怎么办？加个下划线就可以

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属性在搜索不能使用,比如HTML5中的 data-\* 属性

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>]

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>]

**3）text 参数**

通过 text 参数可以搜搜文档中的字符串内容.与 name 参数的可选值一样, text 参数接受 字符串 , 正则表达式 , 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 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.效果与SQL中的limit关键字类似,当搜索到的结果数量达到 limit 的限制时,就停止搜索返回结果.

文档树中有3个tag符合搜索条件,但结果只返回了2个,因为我们限制了返回数量

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 参数**

调用tag的 find\_all() 方法时,Beautiful Soup会检索当前tag的所有子孙节点,如果只想搜索tag的直接子节点,可以使用参数 recursive=False .

一段简单的文档:

复制代码代码如下:

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

是否使用 recursive 参数的搜索结果:

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 )**

它与 find\_all() 方法唯一的区别是 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  
**（3）find\_parents() find\_parent()**

find\_all() 和 find() 只搜索当前节点的所有子节点,孙子节点等. find\_parents() 和 find\_parent() 用来搜索当前节点的父辈节点,搜索方法与普通tag的搜索方法相同,搜索文档搜索文档包含的内容  
**（4）find\_next\_siblings() find\_next\_sibling()**

这2个方法通过 .next\_siblings 属性对当 tag 的所有后面解析的兄弟 tag 节点进行迭代, find\_next\_siblings() 方法返回所有符合条件的后面的兄弟节点,find\_next\_sibling() 只返回符合条件的后面的第一个tag节点  
**（5）find\_previous\_siblings() find\_previous\_sibling()**

这2个方法通过 .previous\_siblings 属性对当前 tag 的前面解析的兄弟 tag 节点进行迭代, find\_previous\_siblings() 方法返回所有符合条件的前面的兄弟节点, find\_previous\_sibling() 方法返回第一个符合条件的前面的兄弟节点  
**（6）find\_all\_next() find\_next()**

这2个方法通过 .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  
**（7）find\_all\_previous() 和 find\_previous()**

这2个方法通过 .previous\_elements 属性对当前节点前面的 tag 和字符串进行迭代, find\_all\_previous() 方法返回所有符合条件的节点, find\_previous()方法返回第一个符合条件的节点

    注：以上（2）（3）（4）（5）（6）（7）方法参数用法与 find\_all() 完全相同，原理均类似，在此不再赘述。

**8.CSS选择器**

我们在写 CSS 时，标签名不加任何修饰，类名前加点，id名前加 #，在这里我们也可以利用类似的方法来筛选元素，用到的方法是 soup.select()，返回类型是 list  
**（1）通过标签名查找**

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）通过类名查找**

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）通过 id 名查找**

print soup.select('#link1')

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

**（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')

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

直接子标签查找

print soup.select("head > title")

#[<title>The Dormouse's story</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("head > title")

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

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

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

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"]')

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

好，这就是另一种与 find\_all 方法有异曲同工之妙的查找方法，是不是感觉很方便？  
**总结**

本篇内容比较多，把 Beautiful Soup 的方法进行了大部分整理和总结，不过这还不算完全，仍然有 Beautiful Soup 的修改删除功能，不过这些功能用得比较少，只整理了查找提取的方法，希望对大家有帮助！小伙伴们加油！

熟练掌握了 Beautiful Soup，一定会给你带来太多方便，加油吧！

**您可能感兴趣的文章:**

* [Python使用Beautiful Soup包编写爬虫时的一些关键点](http://www.jb51.net/article/78375.htm)
* [以视频爬取实例讲解Python爬虫神器Beautiful Soup用法](http://www.jb51.net/article/78366.htm)
* [python基于BeautifulSoup实现抓取网页指定内容的方法](http://www.jb51.net/article/69194.htm)
* [python利用beautifulSoup实现爬虫](http://www.jb51.net/article/55790.htm)
* [python网络编程学习笔记(七)：HTML和XHTML解析(HTMLParser、BeautifulSoup)](http://www.jb51.net/article/50862.htm)
* [Python BeautifulSoup中文乱码问题的2种解决方法](http://www.jb51.net/article/49220.htm)
* [python使用beautifulsoup从爱奇艺网抓取视频播放](http://www.jb51.net/article/46094.htm)
* [python BeautifulSoup使用方法详解](http://www.jb51.net/article/43572.htm)
* [python 解析html之BeautifulSoup](http://www.jb51.net/article/19018.htm)
* [Python的爬虫包Beautiful Soup中用正则表达式来搜索](http://www.jb51.net/article/78380.htm)

**Tags：**[Python](http://img.jb51.net/tag/Python/1.htm)

**相关文章**

* 2015-04-04[Python实现的数据结构与算法之队列详解](http://www.jb51.net/article/64639.htm)
* 2007-02-02[Python学习资料](http://www.jb51.net/article/6818.htm)
* 2009-11-11[python 文件和路径操作函数小结](http://www.jb51.net/article/21007.htm)
* 2014-09-09[python实现在无须过多援引的情况下创建字典的方法](http://www.jb51.net/article/55631.htm)
* 2014-04-04[Python实现的Kmeans++算法实例](http://www.jb51.net/article/49395.htm)
* 2016-01-01[简单介绍Python中的几种数据类型](http://www.jb51.net/article/77351.htm)
* 2015-06-06[python简单获取本机计算机名和IP地址的方法](http://www.jb51.net/article/67181.htm)
* 2015-07-07[python冒泡排序简单实现方法](http://www.jb51.net/article/69208.htm)
* 2015-12-12[python实现爬虫统计学校BBS男女比例（一）](http://www.jb51.net/article/77306.htm)
* 2015-04-04[Python HTMLParser模块解析html获取url实例](http://www.jb51.net/article/63704.htm)
* *1*[Python入门教程 超详细1小时学会](http://www.jb51.net/article/926.htm)
* *2*[python 中文乱码问题深入分析](http://www.jb51.net/article/26543.htm)
* *3*[比较详细Python正则表达式操作指](http://www.jb51.net/article/15707.htm)
* *4*[Python字符串的encode与decode研](http://www.jb51.net/article/17560.htm)
* *5*[python strip()函数 介绍](http://www.jb51.net/article/37287.htm)
* *6*[python 字符串split的用法分享](http://www.jb51.net/article/34998.htm)
* *7*[Python 列表(List)操作方法详解](http://www.jb51.net/article/47978.htm)
* *8*[Python open读写文件实现脚本](http://www.jb51.net/article/15709.htm)
* *9*[Python 字符串操作方法大全](http://www.jb51.net/article/47956.htm)
* *10*[Python 字典(Dictionary)操作详解](http://www.jb51.net/article/47990.htm)

最近更新的内容

* + [在Python中使用lambda高效操作列表的教程](http://www.jb51.net/article/64833.htm)
  + [php使用递归与迭代实现快速排序示例](http://www.jb51.net/article/46102.htm)
  + [python使用xmlrpclib模块实现对百度googl](http://www.jb51.net/article/67114.htm)
  + [Python Tkinter GUI编程入门介绍](http://www.jb51.net/article/61965.htm)
  + [Python脚本实现集群检测和管理功能](http://www.jb51.net/article/61834.htm)
  + [Python单链表的简单实现方法](http://www.jb51.net/article/55545.htm)
  + [Python 文件管理实例详解](http://www.jb51.net/article/74655.htm)
  + [python通过自定义isnumber函数判断字符串](http://www.jb51.net/article/64696.htm)
  + [python比较两个列表是否相等的方法](http://www.jb51.net/article/70117.htm)
  + [wxpython中利用线程防止假死的实现方法](http://www.jb51.net/article/53574.htm)