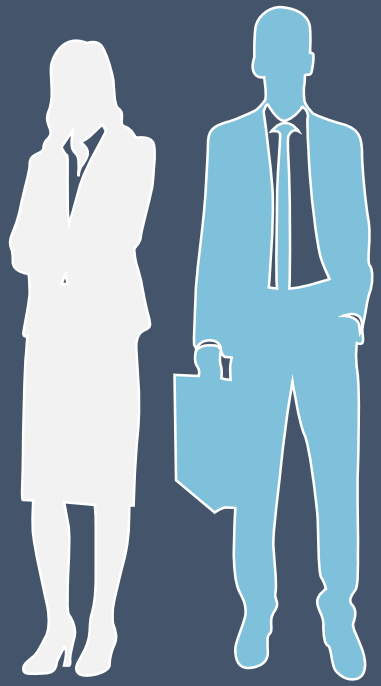
AQF Analyst of Quantitative Finance



使用BeautifulSoup进行网页解析

朱映秋

CONTENTS

Beautiful Soup库安装

- **相关链接**
  - 官方文档：https://www.crummy.com/software/BeautifulSoup/bs4/doc
  - PyPI：https://pypi.python.org/pypi/beautifulsoup4
- **pip安装**
  - Beautiful Soup的HTML和XML解析器是依赖于lxml库的，所以在此之前请确保已经成功安装好了lxml库。
  - 这里推荐使用pip来安装，安装命令如下：pip install beautifulsoup4

Beautiful Soup简介

简介

- Beautiful Soup是一款用于解析HTML或XML的Python库，它可以方便地从网页中提取数据。该库提供了一些简单的、符合Python风格的函数，用于处理导航、搜索和修改解析树等功能。作为一个工具箱，Beautiful Soup能够解析文档，并为用户提供所需的抓取数据。由于其简单易用的特点，使用该库编写一个完整的应用程序所需的代码量很少。
- Beautiful Soup自动将输入文档转换为Unicode编码，输出文档转换为UTF-8编码。不需要考虑编码方式，除非文档没有指定一个编码方式，这时仅仅需要说明一下原始编码方式就可以了。
- Beautiful Soup已成为和lxml、html5lib一样出色的Python解释器，为用户灵活地提供不同的解析策略或强劲的速度。

解析器

- Beautiful Soup在解析时实际上依赖解析器，它除了支持Python标准库中的HTML解析器外，还支持一些第三方解析器。

Beautiful Soup支持的解析器

解析器	使用方法	优点	缺点
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 3.2.2之前的版本文容错能力差
lxml HTML解析器	BeautifulSoup(markup, "lxml")	速度快、文档容错能力强	依赖C语言库
lxml XML解析器	BeautifulSoup(markup, "xml")	速度快、唯一支持XML的解析器	依赖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	解析速度相对较慢、内存占用较高

- 这些解析器各有优势和劣势，根据具体的需求选择适合的解析器可以提高解析效率和灵活性。

- 如果使用lxml，那么在初始化Beautiful Soup时，可以把第二个参数改为lxml即可：

```
# 引入BeautifulSoup包
from bs4 import BeautifulSoup
```

```
# 产生一个soup对象
soup = BeautifulSoup('<p>Hello</p>', 'lxml')
```

```
print(soup.p.string)
```

```
Hello
```

CONTENTS

- **直接调用节点的名称就可以选择节点元素，再调用 string 属性就可以得到节点内的文本了，这种选择方式速度非常快。如果单个节点结构层次非常清晰，可以选用这种方式来解析。**
  - 首先打印输出 title 节点的选择结果，输出结果正是 title 节点加里面的文字内容。
  - 输出它的类型，是 bs4.element.Tag 类型，这是 Beautiful Soup 中一个重要的数据结构。经过选择器选择后，选择结果都是这种 Tag 类型。
  - Tag 具有一些属性，比如 string 属性，调用该属性，可以得到节点的文本内容，所以接下来的输出结果正是节点的文本内容。
  - 接下来选择了 head 节点，结果也是节点加其内部的所有内容。
  - 选择了 p 节点。不过这次情况比较特殊，结果是第一个 p 节点的内容，后面的几个 p 节点并没有选到。也就是说，当有多个节点时，这种选择方式只会选择到第一个匹配的节点，其他的后面节点都会忽略。

选择元素

```
html = """
<html>
    <head>
        <title>
            A test HTML
        </title>
    </head>

    <body>
        <p class="mytitle">
            A test paper title
        </p>
        <p class="story">
            There is a long story.
        </p>
        <a href="http://example.com/page1">
            Link_1
        </a>
        <a href="http://example.com/page2">
            Link_2
        </a>
        <p class="story">...</p>
    </body>

</html>

"""
```

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(html, 'lxml')
```

```
print(soup.title)
print(type(soup.title))
```

```
<title>
            A test HTML
        </title>
<class 'bs4.element.Tag'>
```

```
print(soup.title.string)
```

```
            A test HTML
```

```
print(soup.head)
```

```
<head>
<title>
            A test HTML
        </title>
</head>
```

- **获取名称**
  - 可以利用 name 属性获取节点的名称。这里还是以上面的文本为例，选取 title 节点，然后调用 name 属性就可以得到节点名称：

```
1  print(soup.title.name)
```

```
title
```

- **获取内容**
  - 可以利用 string 属性获取节点元素包含的文本内容，比如要获取第一个 p 节点的文本：

```
1  print(soup.p.string)
```

```
A test paper title
```

获取属性

- 每个节点可能有多个属性，比如 id 和 class 等，选择这个节点元素后，可以调用 attrs 获取所有属性：

```
1 print(soup.p.attrs)
```

```
{'class': ['mytitle']}
```

```
1 print(soup.p.attrs['class'])
```

```
['mytitle']
```

- 可以看到，attrs的返回结果是字典形式，它把选择的节点的所有属性和属性值组合成一个字典。
- 接下来，如果要获取name属性，就相当于从字典中获取某个键值，只需要用中括号加属性名就可以了。比如，要获取name属性，就可以通过attrs['name'] 来得到。

嵌套选择

- 每一个返回结果都是bs4.element.Tag类型，它同样可以继续调用节点进行下一步的选择。比如，获取了head节点元素，我们可以继续调用head来选取其内部的head节点元素。

```
1 print(soup.head.title)
```

```
<title>
A test HTML
</title>
```

```
1 print(type(soup.head.title))
```

```
<class 'bs4.element.Tag'>
```

```
1 print(soup.head.title.string)
```

```
A test HTML
```

- 第一行结果是调用head之后再次调用title而选择的title节点元素。然后打印输出了它的类型，可以看到，它仍然是bs4.element.Tag类型。也就是说，我们在Tag类型的基础上再次选择得到的依然还是Tag类型，每次返回的结果都相同，所以这样就可以做嵌套选择了。
- 最后，输出它的string 属性，也就是节点里的文本内容。

CONTENTS

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,每个节点的对象类型可归纳为4种：
  - 标签（Tag）
  - 内容（NavigableString）
  - 文档的全部内容（BeautifulSoup）
  - 特殊类型的 NavigableString 对象(Comment)

Tag

- Tag 对象可以看成 HTML 中的标签。
  - 分别获取了head标签、title标签、p标签，然后打印出它们的类型，类型都是Tag。
  - 如果Tag对象要获取的标签有多个的话，它只会返回。
  - 所以内容中第一个符合要求的标签。
  - 对象一般含有属性，Tag对象也不例外。它具有两个非常重要的属性，name和attrs。

```
html = """
<html>
    <head>
        <title>
            A test HTML
        </title>
    </head>

    <body>
        <p class="mytitle">
            A test paper title
        </p>
        <p class="story">
            There is a long story.
        </p>
        <a href="http://example.com/page1">
            Link_1
        </a>
        <a href="http://example.com/page2">
            Link_2
        </a>
        <p class="story">...</p>
    </body>

</html>

"""
```

```
print(type(soup.head))
print(type(soup.title))
print(type(soup.p))
```

```
<class 'bs4.element.Tag'>
<class 'bs4.element.Tag'>
<class 'bs4.element.Tag'>
```

```
print(soup.p.name)
print(soup.p.attrs)
```

```
p
{'class': ['mytitle']}
```

Tag

- name
  - name属性是Tag对象的标签名。不过也有特殊的，soup对象的name是 [document] ，对于其他内部标签，输出的值便为标签本身的名称。

```
print(soup.name)

print(soup.title.name)
print(soup.p.name)

print(soup.head.name)
```

```
[document]
title
p
head
```

- attrs
  - attrs属性是Tag对象所包含的属性值，它是一个字典类型。

```
print(soup.head.attrs)
print(soup.a.attrs)
print(soup.p.attrs)
print(soup.p['class'])
```

```
{}
{'href': 'http://example.com/page1'}
{'class': ['mytitle']}
['mytitle']
```

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，可以分别获取它的类型，名称，以及属性。

```
1  print(type(soup))
2  print(soup.attrs)
3  print(soup.name)
```

```
<class 'bs4.BeautifulSoup'>
{}
[document]
```

- soup是我们前面获取解析后网页的全部内容，它的类型是bs4.BeautifulSoup。名字是[document]，属性是空字典。

Comment

- Comment对象是一个特殊类型的NavigableString对象，其实输出的内容不包括注释符号，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。

```
print(type(soup.a))
print(soup.a)
```

```
<class 'bs4.element.Tag'>
<a href="http://example.com/page1">
            Link_1
        </a>
```

```
print(type(soup.a.string))
print(soup.a.string)
```

```
<class 'bs4.element.NavigableString'>

Link_1
```

- 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。

CONTENTS

- 过滤器其实是一个find_all()函数，它会将所有符合条件的内容以列表形式返回。它的构造方法如下：find_all(name, attrs, recursive, text, **kwargs)
  - 共有5个参数
  - find()函数的参数与find_all()函数一样，但是前者是匹配单个结果，后者是匹配多个结果。

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。

```
print(soup.find_all('p'))
```

```
[<p class="mytitle">
            A test paper title
        </p>, <p class="story">
            There is a long story.
        </p>, <p class="story">...</p>]
```

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

```
[<a href="http://example.com/page1">
            Link_1
        </a>, <a href="http://example.com/page2">
            Link_2
        </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。

```
import re

soup.find_all( re.compile('^p') ) # 匹配所有p开头的节点
```

```
[<p class="mytitle">
            A test paper title
        </p>,
 <p class="story">
            There is a long story.
        </p>,
 <p class="story">...</p>]
```

name参数

- 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。

```
print(soup.find_all( ['a','p'] ))
```

```
[<p class="mytitle">
            A test paper title
        </p>, <p class="story">
            There is a long story.
        </p>, <a href="http://example.com/page1">
            Link_1
        </a>, <a href="http://example.com/page2">
            Link_2
        </a>, <p class="story">...</p>]
```

- 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。

```
soup.find_all(True)
```

```
[<html>
 <head>
 <title>
            A test HTML
        </title>
 </head>
 <body>
 <p class="mytitle">
            A test paper title
        </p>
 <p class="story">
```

```
for tag in soup.find_all(True):
    print(tag.name)
```

```
html
head
title
body
p
p
a
a
p
```

attr参数

- attrs参数定义一个字典参数来搜索包含特殊属性的tag。

```
1  print(soup.find_all( attrs={'class': 'mytitle'} ))
```

```
[<p class="mytitle">
            A test paper title
        </p>]
```

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。

```
print(soup.find_all( text='''
            A test HTML
        ''' ))
```

```
['\n            A test HTML\n        ']
```

```
print(soup.find_all( text='A test HTML' ))
```

```
[]
```

```
print(soup.find_all( text=re.compile('A test HTML') ))
```

```
['\n            A test HTML\n        ']
```

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。

```
<a href="http://example.com/page2" id='link2'>
    Link_2
```

```
1 print(soup.find_all( id='link2' ))
```

```
[<a href="http://example.com/page2" id="link2">
        Link_2
    </a>]
```

  - 传入 href 参数,Beautiful Soup会搜索每个tag的”href”属性。

```
1 print(soup.find_all( href=re.compile('example') ))
```

```
[<a href="http://example.com/page1">
        Link_1
    </a>, <a href="http://example.com/page2">
        Link_2
    </a>]
```

  - 想用 class 过滤，不过class是python的关键词，加个下划线就可以。

```
1 print(soup.find_all( 'p', class_='story' ))
```

```
[<p class="story">
        There is a long story.
    </p>, <p class="story">...</p>]
```

其他find方法

- find_parents()：返回所有祖先节点，即返回当前节点的所有上层父节点。
- find_parent()：返回直接父节点，即返回当前节点的直接上层父节点。
- find_next_siblings()：返回后面所有兄弟节点，即返回当前节点之后的所有同级节点。
- find_next_sibling()：返回后面第一个兄弟节点，即返回当前节点之后的第一个同级节点。
- find_previous_siblings()：返回前面所有兄弟节点，即返回当前节点之前的所有同级节点。
- find_previous_sibling()：返回前面第一个兄弟节点，即返回当前节点之前的第一个同级节点。
- find_all_next()：返回节点后所有符合条件的节点，即返回当前节点之后的所有符合条件节点。
- find_next()：返回第一个符合条件的节点，即返回当前节点之后的第一个符合条件的节点。
- find_all_previous()：返回节点前所有符合条件的节点，即返回当前节点之前的所有符合条件的节点。
- find_previous()：返回第一个符合条件的节点，即返回当前节点之前的第一个符合条件的节点。

处理上下文关系

表达式	关系	含义
.contents	直接子节点	可以将tag的子节点以列表的方式输出
.children	直接子节点	返回的类型不是列表，而是可以通过遍历获取所有子节点的迭代器
.descendants	所有子孙节点	返回所有子孙节点的迭代器
.parent	父节点	返回直接父节点
.parents	全部父节点	返回所有父节点的迭代器
.next_siblings	全部兄弟节点	返回该节点的全部下一个兄弟节点，且处在统一级的节点
.previous_siblings	全部兄弟节点	返回该节点的全部上一个兄弟节点，且处在统一级的节点
.next_elements	所有前后节点	返回该节点的后面的所有节点，不分层次关系的
.previous_elements	所有前后节点	返回该节点的前面的所有节点，不分层次关系的
.string	节点内容	返回节点的内容，多个内容则会返回None
.strings	多个内容	返回节点的所有内容，不过需要遍历获取
.stripped_strings	多个内容	若输出的字符串中包含了多空格或空行，可去除多余空白内容

- 还有单个兄弟节点.next_sibling 、.previous_sibling，单个前后节点.next_element、.previous_element

处理上下文关系

```
for child in soup.head.children:
    print(child)
```

```
<title>
            A test HTML
        </title>
```

```
for child in soup.body.children:
    print(child)
```

```
<p class="mytitle">
            A test paper title
        </p>


<p class="story">
            There is a long story.
        </p>


<a href="http://example.com/page1">
            Link_1
        </a>


<a href="http://example.com/page2" id="link2">
            Link_2
        </a>


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

CONTENTS

CSS选择器

- 以CSS语法为匹配标准找到Tag。同样也是使用到一个函数，该函数为select()，返回类型也是list。
  - （1）通过标签名查找

```
1 print(soup.select('title'))
```

```
[<title>
A test HTML
</title>]
```

  - （2）通过class类名查找

```
1 print(soup.select('.story'))
```

```
[<p class="story">
There is a long story.
</p>, <p class="story">...</p>]
```

```
1 print(soup.select('.mytitle'))
```

```
[<p class="mytitle">
A test paper title
</p>]
```

  - （3）通过id名查找

```
1 print(soup.select('#link2'))
```

```
[<a href="http://example.com/page2" id="link2">
Link_2
</a>]
```

- （4）通过组合查找：组合查找即和写标签名与类名、id名进行的组合原理是一样的。 例如查找a标签中，id等于link2的内容。

```
1 print(soup.select('a#link2'))
```

```
[<a href="http://example.com/page2" id="link2">
            Link_2
        </a>]
```

```
1 print(soup.select('head>title'))
```

```
[<title>
            A test HTML
        </title>]
```

- （5）通过属性查找：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。

```
1 print(soup.select('p[class="story"]'))
```

```
[<p class="story">
            There is a long story.
        </p>, <p class="story">...</p>]
```

```
1 print(soup.select('a[id="link2"]'))
```

```
[<a href="http://example.com/page2" id="link2">
            Link_2
        </a>]
```

- （6）get_text()获取内容：以上的select方法返回的结果都是列表形式，可以遍历形式输出，然后用get_text()方法来获取它的内容。

```
print(type(soup.select('p')))
```

```
<class 'bs4.element.ResultSet'>
```

```
print(soup.select('p')[0].get_text())
```

```
A test paper title
```

```
soup.select('p')
```

```
[<p class="mytitle">
            A test paper title
        </p>,
 <p class="story">
            There is a long story.
        </p>,
 <p class="story">...</p>]
```

```
for p in soup.select('p'):
    print(p.get_text())
```

```
A test paper title

There is a long story.

...
```

Thank you!