目录

网页解析利器：BeautifulSoup

- 最新版本：`v1.1`
- 更新时间：`20201120`

简介

整理好用的Python的HTML网页解析器BeautifulSoup的版本选择，为何叫BeautifulSoup，如何安装，如何用其从html网页源码中提取出特定的内容，顺带介绍了最常用的一些逻辑和代码示例写法，以及总结常用的函数find和find_all的具体用法，并用实例解释BeautifulSoup和正则re的效果对比和优缺点对比，以及常见的注意事项和使用期间的心得体会。

源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

Gitbook源码

- crifan/html_parse_tool_beautifulsoup: 网页解析利器：BeautifulSoup

如何使用此Gitbook源码去生成发布为电子书

详见：crifan/gitbook_template: demo how to use crifan gitbook template and demo

在线浏览

- 网页解析利器：BeautifulSoup book.crifan.com
- 网页解析利器：BeautifulSoup crifan.github.io

离线下载阅读

- 网页解析利器：BeautifulSoup PDF
- 网页解析利器：BeautifulSoup ePub
- 网页解析利器：BeautifulSoup Mobi

版权说明

鸣谢

感谢我的老婆陈雪的包容理解和悉心照料，才使得我 `crifan` 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

更多其他电子书

本人 `crifan` 还写了其他 `100+` 本电子书教程，感兴趣可移步至：

crifan/crifan_ebook_readme: Crifan的电子书的使用说明

BeautifulSoup简介

旧教程

之前已有写过一个旧版本的教程，用 `docbook` 发布的：Python专题教程：BeautifulSoup详解

现已把其内容整理合并到此新版教程。

对于HTML网页的解析，可以使用Python中的正则表达式re去提取所需内容。

但是前提（往往是）被解析的html不够复杂，否则正则就很难写，或者说写不出来。

而对于html网页（和xml）的解析，有个专门的库，叫做：

- `BeautifulSoup`
  - 简称： `bs`
    - 最新版本是 `v4` ，简称： `bs4`
  - 核心功能：解析 `HTML` 和 `XML`
  - 特点
    - 功能强大
    - 支持语法有问题的 `HTML` 的解析

为何叫 `BeautifulSoup`

- `BeautifulSoup`
  - 中文直译： `美味的汤`
    - 个人推测是：
      - -》（让人）喝起来很爽（的汤）
        - -》 `BeautifulSoup` 的目的就是：
          - 让你从网页中提取内容很方便
            - -》让你像喝美味的汤一样的爽

什么时候会用到BeautifulSoup

`BeautifulSoup` 这个技术所属领域：一般来说属于Python的爬虫相关的技术领域范围内

一般是在：已经用 `requests` 等库或框架，爬取得到了网页源码，然后想要从html源码中提取特定的内容时

往往才会用到这个：BeautifulSoup

BeautifulSoup的版本

- 之前： `BeautifulSoup 3`
  - 只支持 `Python 2`

        - Python官网（在20200101之后）已不再继续维护 `Python 2` 了
            - 现在已经是20200216了，大家也都尽量不再用Python 2，而改用 `Python 3` 了
    - 最后版本：3.2.2
        - 截至：2019-10-05
    - 安装包：
        - `Debian / Ubuntu` ： `python-beautifulsoup`
        - `Fedora` ： `python-BeautifulSoup`
- 最新： `BeautifulSoup 4`
    - 支持： `Python 2` (`2.7` +)和 `Python 3`
    - 最新版本是： `Beautiful Soup 4.8.2`
        - 截至：2019-12-24
    - 基于BeautifulSoup 4有个： `bs4`
        - 安装包
            - `Debian / Ubuntu`
                - `Python 2` ： `python-bs4`
                - `Python 3` ： `python3-bs4`
            - `Fedora`
                - `python-beautifulsoup4`

官网文档

- 主入口
    - Beautiful Soup: We called him Tortoise because he taught us
        - https://www.crummy.com/software/BeautifulSoup/
- api文档
    - 英文
        - Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation
            - https://www.crummy.com/software/BeautifulSoup/bs4/doc/
    - 中文
        - Beautiful Soup 4.4.0 文档 — Beautiful Soup 4.2.0 documentation
            - https://www.crummy.com/software/BeautifulSoup/bs4/doc.zh/
            - 或
            - https://www.crummy.com/software/BeautifulSoup/bs4/doc/index.zh.html

旧文档

附上 `BS3 = BeautifulSoup v3` 的旧文档，仅供参考

- Beautiful Soup documentation v3
    - https://www.crummy.com/software/BeautifulSoup/bs3/documentation.html

安装

下面主要以，Mac中 `Python 3` 的 `bs4` 为例，解释如何安装 `BeautifulSoup`：

- `pip3 install bs4`
  - 或：`pip install bs4`
    - 确保你的 `pip` 是 `Python 3` 版本

举例：

```
➜  reVsBeautifulSoup pip3 install bs4
Collecting bs4
  Downloading bs4-0.0.1.tar.gz (1.1 kB)
Collecting beautifulsoup4
  Downloading beautifulsoup4-4.8.2-py3-none-any.whl (106 kB)
     |████████████████████████████████| 106 kB 65 kB/s
Collecting soupsieve>1.2
  Downloading soupsieve-1.9.5-py2.py3-none-any.whl (33 kB)
Building wheels for collected packages: bs4
  Building wheel for bs4 (setup.py) ... done
  Created wheel for bs4: filename bs4-0.0.1-py3-none-any.whl size 1272 sha256 603268b090d3
e1b68d5f70078c71c667ef0adab7433943d30bcbdd288161735f
  Stored in directory: /Users/crifan/Library/Caches/pip/wheels/0a/9e/ba/20e5bbc1afef3a491f
0b3bb74d508f99403aabe76eda2167ca
Successfully built bs4
Installing collected packages: soupsieve, beautifulsoup4, bs4
Successfully installed beautifulsoup4-4.8.2 bs4-0.0.1 soupsieve-1.9.5
```

查看已安装的版本：

```
➜  reVsBeautifulSoup pip3 show bs4
Name: bs4
Version: 0.0.1
Summary: Screen-scraping library
Home-page: https://pypi.python.org/pypi/beautifulsoup4
Author: Leonard Richardson
Author-email: leonardr@segfault.org
License: MIT
Location: /usr/local/lib/python3.7/site-packages
Requires: beautifulsoup4
Required-by:
```

此处已安装的版本是：

- `bs4`：`0.0.1`
  - 依赖库
    - `BeautifulSoup4`：`4.8.2`
    - `soupsieve`：`1.9.5`

使用BeautifulSoup提取html网页内容

接着介绍，如何用 `BeautifulSoup` 去提取HTML网页中的特定内容。

举例：

假设有html如下：

```
    <li class="clearfix"><span>2020-12-31</span>
              <a ahref="/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161&type=" href="javascript:void(0)" style="color:#ff0000;" fbfw="外">2019-2020年度全国各地选调生招录、事业单位人才引进信息汇总——全国各地选调生信息汇总</a>
        </li>
    <li class="clearfix"><span>2020-12-31</span>
              <a ahref="/xsglxt/f/jyxt/anony/showZwxx?zpxxid=41174064&type=" href="javascript:void(0)" style="color:#ff0000;" fbfw="外">学术就业相关资讯——清华大学学生职业发展指导中心</a>
        </li>
...
```

需要：提取中的 `li` 中的 `a` 的 `ahref` 值和文本内容 `content`

用BeautifulSoup提取html的典型步骤

先从html中解析出soup：

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(inputHtml, 'html.parser')
```

再去从soup中提取对应的元素

比如找到所有的 `li` 的元素

```
foundLiList = soup.find_all('li', attrs={"class": "clearfix"})
```

或：

```
foundLiList = soup.find_all('li', class_="clearfix")
```

> 说明：为了防止和Python保留字 `class` 冲突，所以改为 `class_`

找到了 `li` 后，再去找其中的 `a` 元素

```
foundA = eachLi.find("a", attrs={"fbfw":"外"})
```

或者是：直接找 `li` 中的 `a` 元素

```
foundAList = soup.find_all('a', attrs={"fbfw":"外"})
```

或者再加上额外限定条件： `a` 中存在属性 `ahref` ，且值非空

此处搜索条件中，用上了正则的写法

```
import re
ahrefNonEmptyP = re.compile("\S+")
foundAList = soup.find_all('a', attrs={"fbfw":"外", "ahref": ahrefNonEmptyP})
```

注：如果想要查找 有 `ahref` 属性，但值无所谓，任意值均可，则可以用： `True` ，表示存在此属性

```
foundAList = soup.find_all('a', attrs={"fbfw":"外", "ahref": True})
```

以及也可以加上style值的限定：

```
ahrefNonEmptyP = re.compile("\S+") # ahref="/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161
&type="
styleColorP = re.compile("color:#[a-zA-Z0-9]+;") # style="color:#ff0000;"
foundAList = soup.find_all('a', attrs={"fbfw":"外", "ahref": ahrefNonEmptyP, "style": style
ColorP})
```

然后对于 `find_all` 找到的是列表，每个元素类型是tag：

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

然后对于每个 `tag` 去通过字典获取属性值， 有2种写法：

- 直接用属性调用

  ```
  ahref = eachA["ahref"]
  ```

- 通过 `attrs` 字典属性

  ```
  ahref = eachA.attrs["ahref"]
  ```

通过 `string` 获取 `文本值` ：

```
contentStr = eachA.string
```

即可获取到需要的值：

```
# ahref=/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161&type=
# contentStr=2019-2020年度全国各地选调生招录、事业单位人才引进信息汇总——全国各地选调生信息汇总
```

如上，就是基本和典型的BeautifulSoup的soup的用法了。

BeautifulSoup常用函数

所有函数和属性

参考 Python & BeautifulSoup - can I use the function findAll repeatedly? - Stack Overflow 中别人回复，可以看出soup的所有属性和函数是：

```
f.HTML_FORMATTERS           f.has_attr
f.XML_FORMATTERS            f.has_key
f.append                    f.hidden
f.attribselect_re           f.index
f.attrs                     f.insert
f.can_be_empty_element      f.insert_after
f.childGenerator            f.insert_before
f.children                  f.isSelfClosing
f.clear                     f.is_empty_element
f.contents                  f.name
f.decode                    f.namespace
f.decode_contents           f.next
f.decompose                 f.nextGenerator
f.descendants               f.nextSibling
f.encode                    f.nextSiblingGenerator
f.encode_contents           f.next_element
f.extract                   f.next_elements
f.fetchNextSiblings         f.next_sibling
f.fetchParents              f.next_siblings
f.fetchPrevious             f.parent
f.fetchPreviousSiblings     f.parentGenerator
f.find                      f.parents
f.findAll                   f.parserClass
f.findAllNext               f.parser_class
f.findAllPrevious           f.prefix
f.findChild                 f.prettify
f.findChildren              f.previous
f.findNext                  f.previousGenerator
f.findNextSibling           f.previousSibling
f.findNextSiblings          f.previousSiblingGenerator
f.findParent                f.previous_element
f.findParents               f.previous_elements
f.findPrevious              f.previous_sibling
f.findPreviousSibling       f.previous_siblings
f.findPreviousSiblings      f.recursiveChildGenerator
f.find_all                  f.renderContents
f.find_all_next             f.replaceWith
f.find_all_previous         f.replaceWithChildren
f.find_next                 f.replace_with
f.find_next_sibling         f.replace_with_children
f.find_next_siblings        f.select
f.find_parent               f.select_one
f.find_parents              f.setup
```

```
f.find_previous              f.string
f.find_previous_sibling      f.strings
f.find_previous_siblings     f.stripped_strings
f.format_string              f.tag_name_re
f.get                        f.text
f.getText                    f.unwrap
f.get_text                   f.wrap
```

供概览了解有哪些属性和功能。

常见属性

- 当前级别
  - 文本
    - 当前节点的文本值
      - `curNode.string`
    - 当前节点的子节点的内容content的列表，即str的list
      - `curNode.contents`
- 同级
  - 兄弟节点
    - 向前
      - 前一个兄弟
        - `curNode.previous_sibling`
      - 前面的所有的兄弟节点
        - `curNode.previous_siblings`
    - 向后
      - 后一个兄弟
        - `curNode.next_sibling`
      - 后面的所有的兄弟节点
        - `curNode.next_siblings`
- 上下级
  - 向上
    - 当前节点的父亲
      - `curNode.parent`
    - 当前节点的（向上查找到的）所有的父亲节点
      - `curNode.parents`
  - 向下
    - 一级
      - （当前 直接的）子节点的列表，即Tag的list
        - `curNode.children`
    - 所有子级
      - 文本
        - 所有子节点的文本
          - `curNode.strings`

        - 去除空行后的所有子节点的文本
            - `curNode.stripped_strings`
    - 节点
        - （其下所有的）子孙节点的列表
            - `curNode.descendants`

常见用法

tag的名字

```
curTagStr = eachSoupNode.name
```

得到：

```
<XCUIElementTypeStaticText type="XCUIElementTypeStaticText" value="兴业 信用卡" name="兴业
信用卡" label="兴业 信用卡" enabled="true" visible="true" x="99" y="593" width="81" height="
18"/>
```

中的tag名：`XCUIElementTypeStaticText`

节点的attrs属性是dict字典

```
curAttrib = eachSoupNode.attrs
```

就是一个dict了，对于：

```
<XCUIElementTypeButton type="XCUIElementTypeButton" enabled="true" visible="true" x="0" y="
0" width="414" height="691">
```

值是：

```
{'enabled': 'true', 'height': '691', 'type': 'XCUIElementTypeButton', 'visible': 'true', '
width': '414', 'x': '0', 'y': '0'}
```

另外例子：

html：

```
<h4>
    <a href="../sanguozhanji/" target="_blank" title="三国战纪"><em
            class='keyword'>三国</em>战纪(官方正版)</a>
    <span>
        20年经典风靡街机厅
    </span>
</h4>
```

获取属性：

```
h4Soup = dtSoup.find("h4")
h4aSoup = h4Soup.find("a")
h4aAttrDict = h4aSoup.attrs # h4aAttrDict={'href': '../sanguozhanji/', 'target': '_blank',
 'title': '三国战纪'}
aHref = h4aAttrDict["href"] # '../sanguozhanji/'
aTitle = h4aAttrDict["title"] # '三国战纪'
```

删除某个属性

官网文档：attributes

就像删除dict中的某个key一样：

```
del curNode.attrs["keyToDelete"]
```

或：

```
curNodeAttributeDict = curNode.attrs
del curNodeAttributeDict["keyToDelete"]
```

常见函数操作

soup.find

- 官网文档
  - 中文
    - find(name, attrs, recursive, string, **kwargs)
  - 英文
    - find(name, attrs, recursive, string, **kwargs)

更多实际用法举例：

```
    # <h1 class="h1user">crifan</h1>

    # method 1: no designate para name
    h1userSoup = soup.find("h1", {"class":"h1user"})

    # method 2: use para name
    h1userSoup = soup.find(name "h1", attrs {"class":"h1user"})

    h1userUnicodeStr = h1userSoup.string
```

修改其中内容：

注：只能改（Tag的）中的属性的值，不能改（Tag的）的值本身

```
soup.body.div.h1.string = changedToString

soup.body.div.h1['class'] = "newH1User"
```

soup.findall

- 官网文档
  - 中文
    - find_all(name, attrs, recursive, string, **kwargs)
  - 英文
    - find_all(name, attrs, recursive, string, limit, **kwargs)

默认findall会返回匹配的所有的元素

想要限制返回个数，可以加 `limit`

`soup.find_all('title', limit=2)`

特殊：

`find` == `limit=1` 的 `findall`

即：如下是相同含义

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

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

decompose 删除节点

```
nodeToDelete.decompose()
```

官网文档：decompose()

BeautifulSoup和re详细对比

下面就通过具体的例子，即：

以之前回复的这个帖子，来详细解释：

- 如何从HTML中提取所需内容
  - BeautifulSoup的写法
    - `find` 的用法
    - `find_all` 的用法
  - re正则的写法
    - `re.findall` 的用法
    - `re.finditer` 的用法

详细代码如下：

```
# Function: 通过对比说明如何用BeautifulSoup和正则re去提取html中的内容
#   举例所用需求来自此帖:
#     python正则表达式提取空列表-CSDN论坛
#     https://bbs.csdn.net/topics/395845984
#   后已经整理至教程:
#     网页解析利器: BeautifulSoup
#     http://book.crifan.com/books/html_parse_tool_beautifulsoup/website
# Author: Crifan Li
# Update: 20200216

import codecs
from bs4 import BeautifulSoup
import re

respHtmlFile = "responseHtml.html"

# 第一次: 初始化，保存html到文件
import requests
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH
TML, like Gecko) Chrome/80.0.3987.106 Safari/537.36'}
url = 'http://career.cic.tsinghua.edu.cn/xsglxt/f/jyxt/anony/xxfb'
respHtml = requests.get(url, headers=headers).text
# save html to file for later debug
with open(respHtmlFile, "w") as htmlFp:
  htmlFp.write(respHtml)
  htmlFp.close()

# # 后续调试: 从文件中读取html代码，方便调试
# def loadTextFromFile(fullFilename, fileEncoding="utf-8"):
#   """load file text content from file"""
#   with codecs.open(fullFilename, 'r', encoding=fileEncoding) as fp:
#     allText = fp.read()
#     # logging.debug("Complete load text from %s", fullFilename)
#     return allText
```

```
# respHtml = loadTextFromFile(respHtmlFile)

"""
【要处理的html的源码】

    <li class="clearfix"><span>2020-12-31</span>

                <a ahref="/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161&type=" href="javascript:void(0)" style="color:#ff0000;" fbfw="外">2019-2020年度全国各地选调生招录、事业单位人才引进信息汇总——全国各地选调生信息汇总</a>

        </li>



    <li class="clearfix"><span>2020-12-31</span>

                <a ahref="/xsglxt/f/jyxt/anony/showZwxx?zpxxid=41174064&type=" href="javascript:void(0)" style="color:#ff0000;" fbfw="外">学术就业相关资讯——清华大学学生职业发展指导中心</a>

        </li>
  ...

【背景解释】
上述html元素结构是:
li
  span
  a
    中文文字

【需求说明】
假如要提取的是:
每个li中a的:
  ahref的链接地址
  中文文字
"""

################################################################################
# 用正则re提取html内容
################################################################################
print("="*20, "用正则re提取html内容", "="*20)

print("="*10, "方式1: 用re.findall一次性找2个值", "="*10)

# 方式1: 匹配整个li的部分
wholeLiP = '<li\s+class="clearfix"><span>.*?</span>\s*<a\s+ahref="(.*?)"\s+href="javascript:void\(0\)"\s+style="color:.*?;"\s+fbfw="外">(.*?)</a>\s*</li>'
print("wholeLiP=%s" % wholeLiP)

foundAllMatchedTupleList = re.findall(wholeLiP, respHtml, re.S)
```

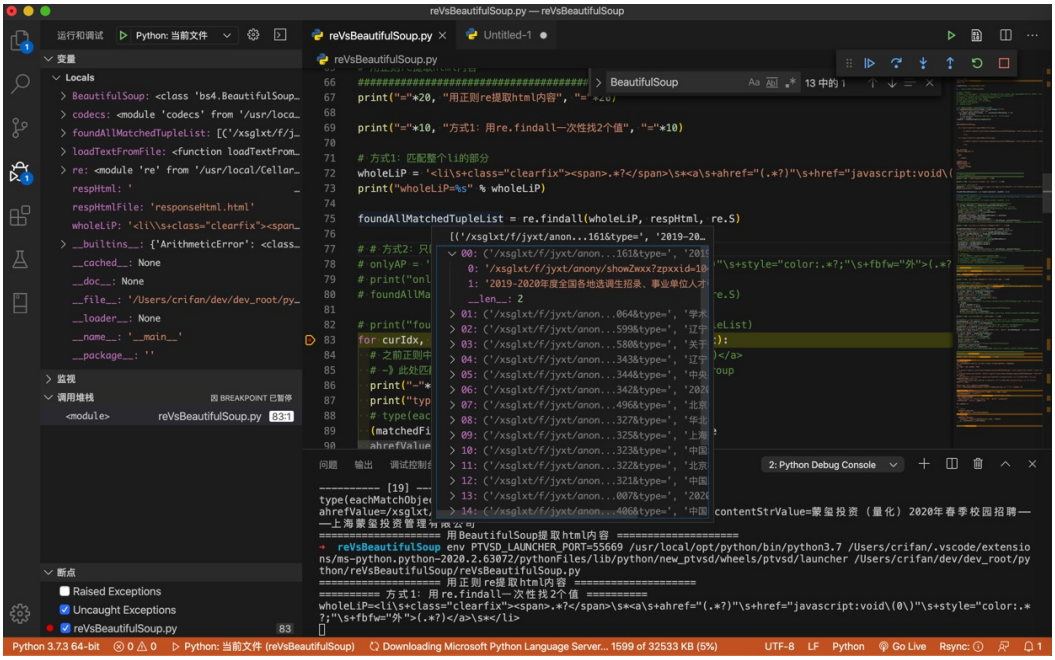
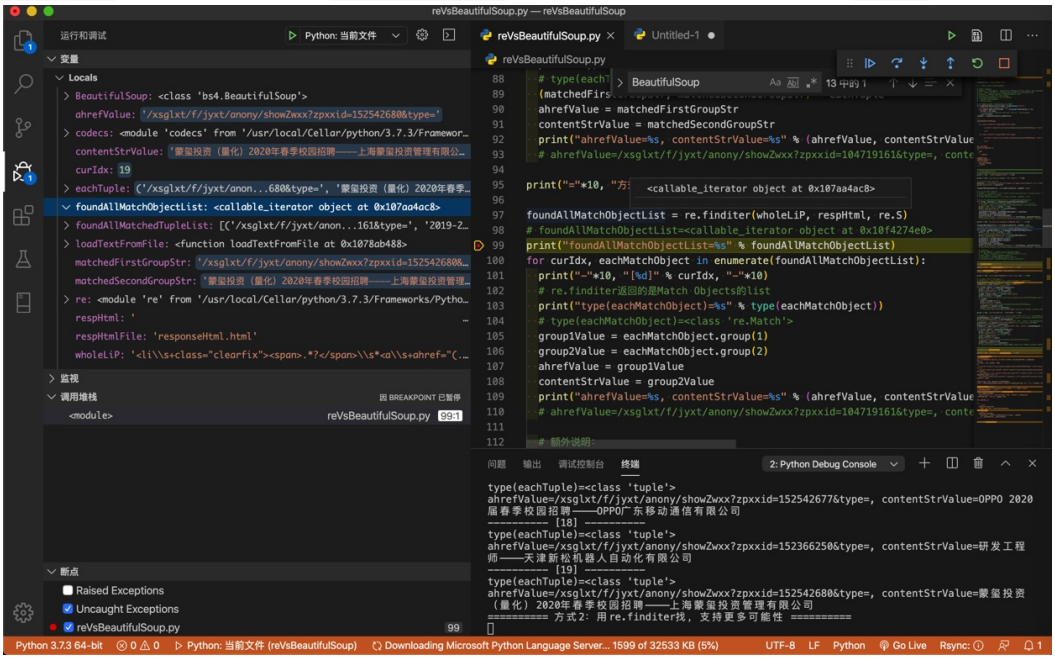
```
# # 方式2：只匹配a的部分
# onlyAP = '<a\s+ahref="(.*?)"\s+href="javascript:void\(0\)"\s+style="color:.*?;"\s+fbfw="
外">(.*?)</a>'
# print("onlyAP=%s" % onlyAP)
# foundAllMatchedTupleList = re.findall(onlyAP, respHtml, re.S)

# print("foundAllMatchedTupleList=%s" % foundAllMatchedTupleList)
for curIdx, eachTuple in enumerate(foundAllMatchedTupleList):
  # 之前正则中有2个括号，对应2个group组：ahref="(.*?)"，和 >(.*?)</a>
  # -》此处匹配到的值是个tuple元素，是2个元素，分别对应着之前的2个group
  print("-"*10, "[%d]" % curIdx, "-"*10)
  print("type(eachTuple)=%s" % type(eachTuple))
  # type(eachTuple)=<class 'tuple'>
  (matchedFirstGroupStr, matchedSecondGroupStr) = eachTuple
  ahrefValue = matchedFirstGroupStr
  contentStrValue = matchedSecondGroupStr
  print("ahrefValue=%s, contentStrValue=%s" % (ahrefValue, contentStrValue))
  # ahrefValue=/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161&type=, contentStrValue=2019-
2020年度全国各地选调生招录、事业单位人才引进信息汇总——全国各地选调生信息汇总

print("="*10, "方式2：用re.finditer找，支持更多可能性", "="*10)

foundAllMatchObjectList = re.finditer(wholeLiP, respHtml, re.S)
# foundAllMatchObjectList=<callable_iterator object at 0x10f4274e0>
print("foundAllMatchObjectList=%s" % foundAllMatchObjectList)
for curIdx, eachMatchObject in enumerate(foundAllMatchObjectList):
  print("-"*10, "[%d]" % curIdx, "-"*10)
  # re.finditer返回的是Match Objects的list
  print("type(eachMatchObject)=%s" % type(eachMatchObject))
  # type(eachMatchObject)=<class 're.Match'>
  group1Value = eachMatchObject.group(1)
  group2Value = eachMatchObject.group(2)
  ahrefValue = group1Value
  contentStrValue = group2Value
  print("ahrefValue=%s, contentStrValue=%s" % (ahrefValue, contentStrValue))
  # ahrefValue=/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161&type=, contentStrValue=2019-
2020年度全国各地选调生招录、事业单位人才引进信息汇总——全国各地选调生信息汇总

  # 额外说明：
  # 如果你前面正则中是named group带命名的组，比如：
  # ... ahref="(?P<ahref>.*?)" ... >(?P<contentStr>.*?)</a>
  # 那么也可以通过group name组名去获取值：
  # ahrefValue = eachMatchObject.group("ahref")
  # contentStrValue = eachMatchObject.group("contentStr")

################################################################################
# 用BeautifulSoup提取html内容
################################################################################
print("="*20, "用BeautifulSoup提取html内容", "="*20)

soup = BeautifulSoup(respHtml, 'html.parser')
# print("soup=%s" % soup)
```

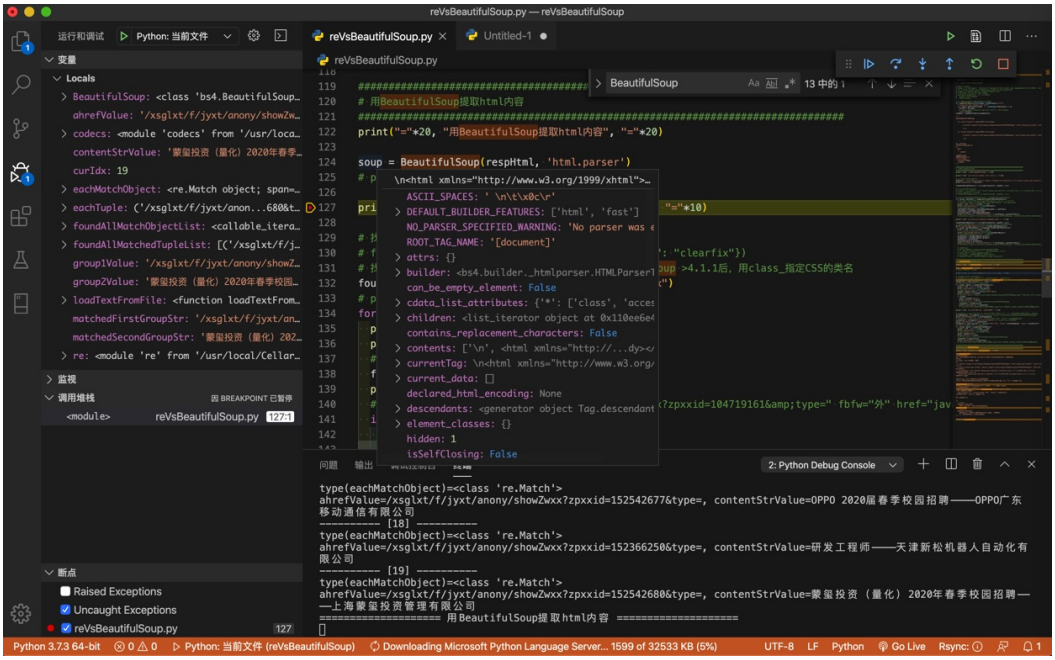
```
print("="*10, "方式1：先找外层的li，再去li中找其中的a", "="*10)

# 找li的方式1：通过attrs指定属性
# foundLiList = soup.find_all('li', attrs={"class": "clearfix"})
# 找li的方式2：class 在Python中是保留字 -> BeautifulSoup >4.1.1后，用class_指定CSS的类名
foundLiList = soup.find_all('li', class_="clearfix")
# print("foundLiList=%s" % foundLiList)
for curIdx, eachLi in enumerate(foundLiList):
  print("-"*10, "[%d]" % curIdx, "-"*10)
  print("type(eachLi)=%s" % type(eachLi))
  # type(eachLi)=<class 'bs4.element.Tag'>
  foundA = eachLi.find("a", attrs={"fbfw":"外"})
  print("foundA=%s" % foundA)
  # foundA=<a ahref="/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161&amp;type=" fbfw="外" hr
ef="javascript:void(0)" style="color:#ff0000;">2019-2020年度全国各地选调生招录、事业单位人才引
进信息汇总——全国各地选调生信息汇总</a>
  if foundA:
    ahref = foundA["ahref"]
    print("ahref=%s" % ahref)
    # ahref=/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161&type=
    contentStr = foundA.string
    print("contentStr=%s" % contentStr)
    # contentStr=2019-2020年度全国各地选调生招录、事业单位人才引进信息汇总——全国各地选调生信息
汇总

print("="*10, "方式2：直接找a，加上限定条件", "="*10)

# foundAList = soup.find_all('a', attrs={"fbfw":"外"}) # 只加上一个fbfw的限定条件，此处也是可
以的
ahrefNonEmptyP = re.compile("\S+") # ahref="/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161
&type="
print("ahrefNonEmptyP=%s" % ahrefNonEmptyP)
# foundAList = soup.find_all('a', attrs={"fbfw":"外", "ahref": ahrefNonEmptyP})
styleColorP = re.compile("color:#[a-zA-Z0-9]+;") # style="color:#ff0000;"
print("styleColorP=%s" % styleColorP)
foundAList = soup.find_all('a', attrs={"fbfw":"外", "ahref": ahrefNonEmptyP, "style": style
ColorP})
# print("foundAList=%s" % foundAList)
for curIdx, eachA in enumerate(foundAList):
  print("-"*10, "[%d]" % curIdx, "-"*10)
  print("type(eachA)=%s" % type(eachA))
  # type(eachA)=<class 'bs4.element.Tag'>
  ahref = eachA["ahref"]
  print("ahref=%s" % ahref)
  # ahref=/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161&type=
  contentStr = eachA.string
  print("contentStr=%s" % contentStr)
  # contentStr=2019-2020年度全国各地选调生招录、事业单位人才引进信息汇总——全国各地选调生信息汇总


################################################################################
# 对比：re vs BeautifulSoup
################################################################################
```

```
print("=" * 20, "对比：re vs BeautifulSoup", "=" * 20)

reVsBeautifulSoup = """
re正则的缺点：
万一html源代码改动了，即使改动很小，则之前已有的re正则表达式就失效了
举例：
只是a的属性的顺序变化一点点
从
  <a ahref="/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161&type=" href="javascript:void(0)
" style="color:#ff0000;" fbfw="外">2019-2020年度全国各地选调生招录、事业单位人才引进信息汇总——
—全国各地选调生信息汇总</a>
改为：
  <a href="javascript:void(0)" ahref="/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161&type=
" fbfw="外" style="color:#ff0000;">2019-2020年度全国各地选调生招录、事业单位人才引进信息汇总——
—全国各地选调生信息汇总</a>
之前正则：
  '<a\s+ahref="(.*?)"\s+href="javascript:void\(0\)"\s+style="color:.*?;"\s+fbfw="外">(.*?)<
/a>'
就无效了，就要再去改为：
  '<a\s+href="javascript:void\(0\)"\s+ahref="(.*?)"\s+fbfw="外"\s+style="color:.*?;">(.*?)<
/a>'
才可以匹配到。

更别说，万一html中代码有其他更大的变化
甚至是部分语法不规范的html代码，re正则根本就没法写，因为太复杂，复杂到写不出来

BeautifulSoup的优点：
与之相对：上述的，html代码的小改动，比如属性值出现的顺序不同
甚至大点的变化，多出其他属性值
甚至部分语法不规范的html代码，BeautifulSoup都可以很好的内部处理掉
而之前的代码，比如：
  soup.find_all('a', attrs={"fbfw":"外", "ahref": nonEmptyP})
都可以很好的继续工作，而无需改动。

汇总起来就是：

re
  性能：好
  支持html程度：有限
    仅限于不是很复杂的，比较规整的html
BeautifulSoup
  性能：中等
  支持html程度：很好
    不仅支持复杂的html，还支持html内部元素和位置变化
    对于不规范的html也有很好的支持
"""

print(reVsBeautifulSoup)
```

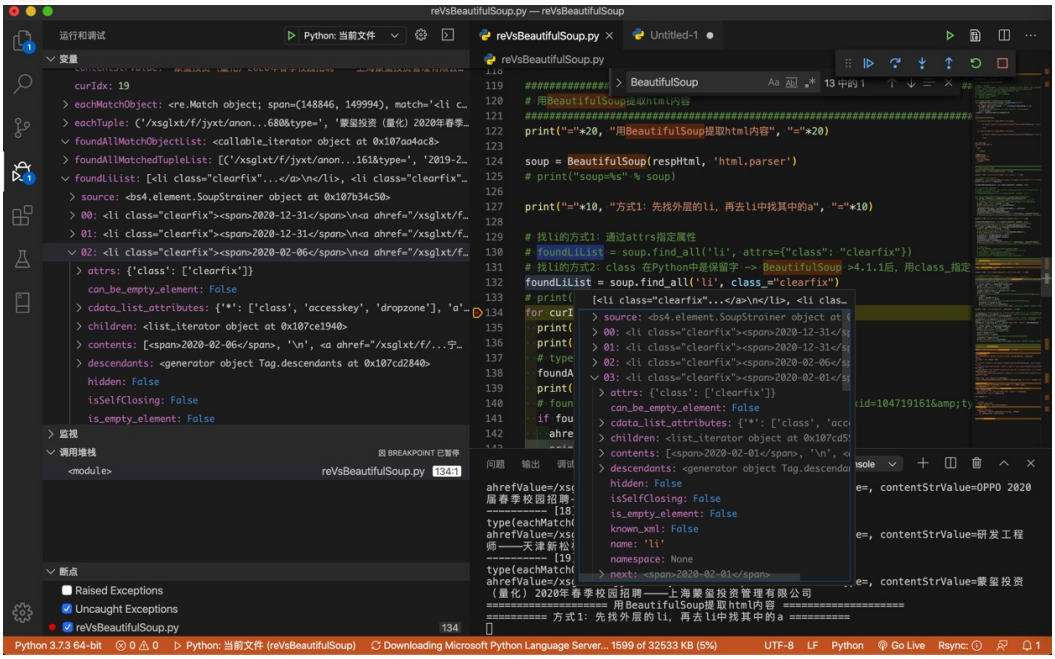
bs 和 re 函数返回变量类型

此处调试期间，可以看到对应变量的类型：

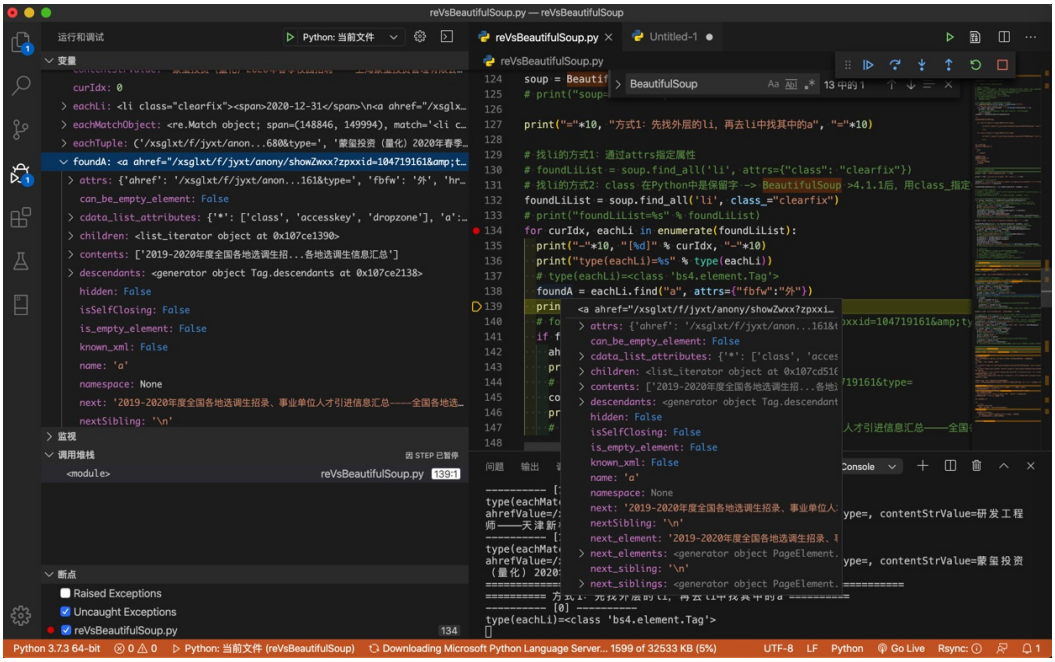
- 正则re
  - `re.findall` 返回的是匹配的元祖 `tuple` 的列表：<class 'tuple'>的 `list`
    - 
  - `re.finditer` 返回的是匹配对象 `Match Object` 的列表：<class 're.Match'>的 `list`
    - 
- BeautifulSoup
  - `BeautifulSoup` 返回的 `soup` 变量的详情：



- 而 `BeautifulSoup` 的 `find_all` 返回的是标签元素的列表：<class 'bs4.element.Tag'> 的 `list`



- 而 `BeautifulSoup` 的 `find` 返回的是单个标签元素：<class 'bs4.element.Tag'>



BeautifulSoup vs 正则re

最后总结各自的优缺点：

- re
  - 性能：好
  - 支持html程度：有限
    - 仅限于不是很复杂的，比较规整的html
  - 常用函数
    - `re.search`
    - `re.findall`
    - `re.finditer`
- BeautifulSoup
  - 性能：中等
  - 支持html程度：很好
    - 不仅支持复杂的html
      - 还支持html网页源码内部元素和位置变化时，往往 `bs` 的代码也无需改动
    - 对于不规范的html也有很好的支持
  - 常用函数
    - `soup.find`
    - `soup.findall`

注意事项和心得

`BeautifulSoup v3` 升级到 `BeautifulSoup v4` 后的函数变化

官网已解释, BeautifulSoup从 `v3` 升级到 `v4` 后，很多函数名变化了：

- `renderContents` -> `encode_contents`
- `replaceWith` -> `replace_with`
- `replaceWithChildren` -> `unwrap`
- `findAll` -> `find_all`
- `findAllNext` -> `find_all_next`
- `findAllPrevious` -> `find_all_previous`
- `findNext` -> `find_next`
- `findNextSibling` -> `find_next_sibling`
- `findNextSiblings` -> `find_next_siblings`
- `findParent` -> `find_parent`
- `findParents` -> `find_parents`
- `findPrevious` -> `find_previous`
- `findPreviousSibling` -> `find_previous_sibling`
- `findPreviousSiblings` -> `find_previous_siblings`
- `nextSibling` -> `next_sibling`
- `previousSibling` -> `previous_sibling`

其他细节变化：

- Beautiful Soup构造方法的参数部分也有名字变化
  - `BeautifulSoup(parseOnlyThese=...)` -> `BeautifulSoup(parse_only=...)`
  - `BeautifulSoup(fromEncoding=...)` -> `BeautifulSoup(from_encoding=...)`
- 为了适配Python3,修改了一个方法名
  - `Tag.has_key()` -> `Tag.has_attr()`
- 修改了一个属性名,让它看起来更专业点
  - `Tag.isSelfClosing` -> `Tag.is_empty_element`
- 修改了下面3个属性的名字,以免与Python保留字冲突.这些变动不是向下兼容的,如果在BS3中使用了这些属性,那么在BS4中这些代码无法执行.
  - `UnicodeDammit.Unicode` -> `UnicodeDammit.Unicode_markup`
  - `Tag.next` -> `Tag.next_element`
  - `Tag.previous` -> `Tag.previous_element`

注意事项

BeautifulSoup的Tag的属性

BeautifulSoup处理这种html源码：

```
<a href="http://creativecommons.org/licenses/by/3.0/deed.zh" target="_blank">版权声明</a>
```

后，是可以通过 `soup.a['href']` 去获得对应的href属性值的。

但是，想要去获得当前的某个未知的BeautifulSoup.Tag中，一共存在多少个属性，以及每个属性的值的时候，却不知道如何下手了。

比如对于如下html源码：

```
<p class="cc-lisence" style="line-height:180%;">......</p>
```

想要得知，一共有两个属性，分别是class和style，然后就可以通过上面的方法，去获得对应的属性值了。

对此问题，虽然看到了官网的解释：Tags的属性中的 `你可以将Tag看成字典来访问标签的属性`

但是还是无法通过：

```
if("class" in soup)
```

的方式去判断soup中是否存在class属性，因为此时soup是Beautifulsoup.Tag类型变量，而不是dict变量。

并且，如果去强制将soup转换成为dict变量：

```
soupDict = dict(soup)
```

会报错的。

最后，还是无意间发现，原来 `Beautifulsoup.Tag` 是有个 `attrs` 属性的，其可以获得对应的元组列表，每一个元组是对应属性名和属性值：

```
attrsList = soup.attrs
print("attrsList=%s" % attrsList())

attrsList= [('class', 'cc-lisence'), ('style', 'line-height:180%;')]
```

这样自己就可以从元组列表中去转换，获得属性的列表或字典变量了，就可以接着按照自己意愿去处理了。

另外，此处也通过 `soup.name` 获得了该tag的名字

而想要获得整个soup变量所有的属性和方法的话，可以用经典的dir去打印出来：

```
print("dir(soup)=%s" % dir(soup))
```

此处的打印输出为：

```
dir(soup) = ['BARE_AMPERSAND_OR_BRACKET', 'XML_ENTITIES_TO_SPECIAL_CHARS', 'XML_SPECIAL_CHA
RS_TO_ENTITIES', '__call__', '__contains__', '__delitem__', '__doc__', '__eq__', '__getatt
r__', '__getitem__', '__init__', '__iter__', '__len__', '__module__', '__ne__', '__nonzero
__', '__repr__', '__setitem__', '__str__', '__unicode__', '_convertEntities', '_findAll',
'_findOne', '_getAttrMap', '_invert', '_lastRecursiveChild', '_sub_entity', 'append', 'att
rMap', 'attrs', 'childGenerator', 'containsSubstitutions', 'contents', 'convertHTMLEntitie
s', 'convertXMLEntities', 'decompose', 'escapeUnrecognizedEntities', 'extract', 'fetch', '
fetchNextSiblings', 'fetchParents', 'fetchPrevious', 'fetchPreviousSiblings', 'fetchText',
'find', 'findAll', 'findAllNext', 'findAllPrevious', 'findChild', 'findChildren', 'findNex
t', 'findNextSibling', 'findNextSiblings', 'findParent', 'findParents', 'findPrevious', 'f
indPreviousSibling', 'findPreviousSiblings', 'first', 'firstText', 'get', 'has_key', 'hidd
en', 'insert', 'isSelfClosing', 'name', 'next', 'nextGenerator', 'nextSibling', 'nextSibli
ngGenerator', 'parent', 'parentGenerator', 'parserClass', 'prettify', 'previous', 'previou
sGenerator', 'previousSibling', 'previousSiblingGenerator', 'recursiveChildGenerator', 're
nderContents', 'replaceWith', 'setup', 'substituteEncoding', 'toEncoding']
```

有需要的话，可以对这些属性和方法，都尝试一下，以便更加清楚其含义。

刚写完上面这句话呢，然后自己随便测试了一下 `attrMap` ：

```
attrMap = soup.attrMap
print("attrMap=%s" % attrMap)
```

然后就很惊喜的发现，原来此处的attrMap，就是我程序中所需要的属性的dict变量啊：

```
attrMap = {'style': 'line-height:180%;', 'class': 'cc-lisence'}
```

这样，就又省去了我程序中将attrs转换为dict变量的操作了，更加提高了效率。

在次感谢Beautifulsoup的开发者。

BeautifulSoup有时候会遇到非法的，不支持的html源码而导致无法解析或无法正常解析html

在使用Beautifulsoup过程中，对于大多数html源码，通过指定正确的编码，或者本身是默认UTF-8编码而无需指定编码类型，其都可以正确解析html源码，得到对应的soup变量。

然后就接着去利用soup实现你所想要的功能了。

但是有时候会发现，有些html解析后，有些标签等内容丢失了，即所得到的soup不是所期望的完整的html的内容。

这时候，很可能遇到了非法的html，即其中可能包含了一些不合法的html标签等内容，导致Beautifulsoup虽然可以解析，没有报错，但是实际上得到的soup变量，内容缺失了一部分了。

比如我就遇到过不少这样的例子：

部分Blogbus的帖子的html中非html5和html5的代码混合导致Beautifulsoup解析错误

之前在为BlogsToWordPress添加Blogbus支持过程中去解析Blogbus的帖子的时候，遇到一个特殊的帖子：

http://ronghuihou.blogbus.com/logs/89099700.html

其中一堆的非html5的代码中，包含了这样一段html5的代码的写法，即标签属性值不加括号的：

```
<SCRIPT language=JavaScript>
document.oncontextmenu=new Function("event.returnValue=false;"); //禁止右键功能,单击右键将无任何反应
document.onselectstart=new Function( "event.returnValue=false;"); //禁止先择,也就是无法复制
</SCRIPT language=JavaScript>
```

结果导致Beautifulsoup解析错误，得到的soup中，找不到所需要的各种class等属性值。

对应的解决办法就是，把这部分的代码删除掉，然后再解析就可以了：

其中一堆的非html5的代码中，包含了这样一段html5的代码的写法，即标签属性值不加括号的：

```
foundInvliadScript = re.search("<SCRIPT language=JavaScript>.+</SCRIPT language=JavaScript>", html, re.I | re.S )
logging.debug("foundInvliadScript=%s", foundInvliadScript)
if(foundInvliadScript):
    invalidScriptStr = foundInvliadScript.group(0)
    logging.debug("invalidScriptStr=%s", invalidScriptStr)
    html = html.replace(invalidScriptStr, "")
    logging.debug("filter out invalid script OK")

soup = htmlToSoup(html)
```

判断浏览器版本的相关代码，导致Beautifulsoup解析不正常

之前在给BlogsToWordpress添加新浪博客的支持的过程中

遇到很多新浪博客的帖子的html中，包含很多判断浏览器版本的相关代码：

```
<!-[if lte IE 6]>
xxx
xxx
<![endif]->
```

由此导致Beautifulsoup解析html不正常。

font标签嵌套层次太多，导致Beautifulsoup无法解析html

接上面那个解析新浪博客帖子的例子，期间又遇到另外一个问题，对于一些特殊帖子：

http://blog.sina.com.cn/s/blog_5058502a01017j3j.html

其包含特殊的好几十个font标签且是一个个嵌套的代码，导致无法Beautifulsoup无法解析html，后来把对应嵌套的font标签删除掉，才可以正常解析。

相关python代码为：

```
# handle special case for http://blog.sina.com.cn/s/blog_5058502a01017j3j.html
processedHtml = processedHtml.replace('<font COLOR="#6D4F19"><font COLOR="#7AAF5A"><font C
OLOR="#7AAF5A"><font COLOR="#6D4F19"><font COLOR="#7AAF5A"><font COLOR="#7AAF5A">', "")
processedHtml = processedHtml.replace("</FONT></FONT></FONT></FONT></FONT></FONT>", "")
```

遇到其他类似的问题，也可以去删除或替换出错代码，即可解决问题。

不过需要说明的是，很多时候，你未必很容易就找到出错的代码。

想要找到出错的代码，更多的时候，需要你一点点调试，一点点的删除看似可疑的一些html源码，然后最终才能定位到出错的代码，然后删除掉后，才可以正常工作的。

使用心得

crifan的Beautifulsoup函数

之前已把一些常用功能封装成函数，放到自己的库中了，详见：

[crifanLibPython/crifanBeautifulsoup.py at master · crifan/crifanLibPython

`find` 和 `find_all` 中支持正则写法

如之前例子中所提到的：

```
ahrefNonEmptyP = re.compile("\S+") # ahref="/xsglxt/f/jyxt/anony/showZwxx?zpxxid=104719161
&type="
styleColorP = re.compile("color:#[a-zA-Z0-9]+;") # style="color:#ff0000;"
foundAList = soup.find_all('a', attrs={"fbfw":"外", "ahref": ahrefNonEmptyP, "style": style
ColorP})
```

`soup.find` 和 `soup.find_all` 中，都支持正则的写法：

```
# also match a-incontent a-title cs-contentblock-hoverlink
titleP = re.compile("a-incontent a-title(\s+?\w+)?")
foundIncontentTitle = soup.find(attrs={"class":titleP})
```

就可以同时匹配多种情况了：

- `a-incontent a-title`
- `a-incontent a-title cs-contentblock-hoverlink`

- `a-incontent a-title xxx`

附录

下面列出相关参考资料。

参考资料

- 【已解决】BeautifulSoup中如何删除某个节点
-
- 实现博客搬家
- Python专题教程：BeautifulSoup详解
- re — Regular expression operations — Python 3.8.2rc1 documentation
- re --- 正则表达式操作 — Python 3.8.2rc1 文档
- 【教程】Python中第三方的用于解析HTML的库：BeautifulSoup – 在路上
- 【已解决】Python3中，已经安装了bs4（Beautifulsoup 4）了，但是却还是出错：ImportError: No module named BeautifulSoup
- 【教程】抓取网并提取网页中所需要的信息 之 Python版
- 【总结】Python的第三方库BeautifulSoup的使用心得
- 【整理】关于Python中的html处理库函数BeautifulSoup使用注意事项
- 【已解决】BeautifulSoup已经获得了Unicode的Soup但是print出来却是乱码
- 【教程】BeautifulSoup中使用正则表达式去搜索多种可能的关键字
- 【整理】用BeautifulSoup查找属性值未知的标签
- crifanLib/crifanLib.py at master · crifan/crifanLib
- 【已解决】Python中用BeautifulSoup提取class中包含一定规则的节点
- 【已解决】Beautifulsoup 4中搜索html的p的value包含特定值和p中的a的href
-