



# 第五章 网络爬虫与信息提取

## 5.2 信息解析

同济大学 计算机基础教研室

# 三大解析方法

- 使用Requests获取到网页的 HTML 代码信息后，怎样分析得到我们想要的信息？

方法	效率	难度
BeautifulSoup	慢	最简单
正则表达式	最快	困难
Xpath	快	正常

- Beautiful Soup是一个HTML或XML解析库，用于格式化和组织复杂的网页信息，是解析、遍历、维护“标签树”的功能库。

# 引例：豆瓣电影名称提取

浏览器地址栏显示：<https://movie.douban.com/cinema/nowplaying/shanghai/>

页面标题：电影票 - 上海 [切换城市]

正在上映

页面展示了正在上映的电影海报，其中《大侦探皮卡丘》（Detective Pikachu）被红色框选中。

该电影的豆瓣评分为 6.7 分。

右侧显示了该电影的 HTML 结构，其中 `<li id="26835471" class="list-item">` 被红色框选中，其属性包括：

- `data-title="大侦探皮卡丘"`
- `data-score="6.7"`
- `data-star="35"`
- `data-release="2019"`
- `data-duration="104分钟"`
- `data-region="美国 日本"`
- `data-director="罗伯·莱特曼"`
- `data-actors="瑞恩·雷诺兹 / 贾斯蒂斯·史密斯 / 凯瑟琳·纽顿"`
- `data-category="nowplaying"`
- `data-enough="True"`
- `data-showed="True"`
- `data-votecount="90244"`
- `data-subject="26835471"`

下方显示了该电影的选座购票按钮。

<https://movie.douban.com/cinema/nowplaying/shanghai/>

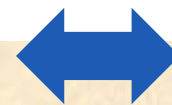
# 代码

```
import requests
from bs4 import BeautifulSoup
url = "https://movie.douban.com/cinema/nowplaying/shanghai/"
headers = {'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.108 Safari/537.36"}
r = requests.get(url, headers=headers) # 获取页面信息
soup = BeautifulSoup(r.text, 'html.parser') # 分析页面
# 找到所有的电影信息对应的li标签
movie_list = soup.find_all('li', class_='list-item')
# 遍历所有li标签，分别获取电影名称
for item in movie_list:
    print(item['data-title'])
```

html源代码由很多尖括号构成的标签组织而成

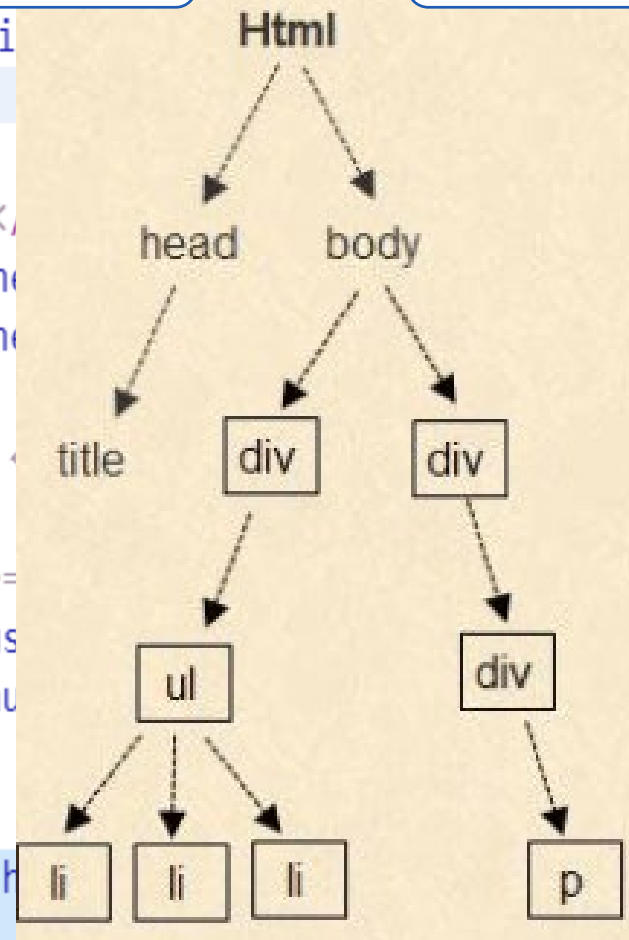
标签之间存在上下游关系

文件



标签树

```
<!doctype html>
<html xmlns="http://www.w3.org/1999/xhtml" class="sui">
  <head>...</head>
  <body class="s-manhattan-index" style>
    <div id="s_is_index_css" style="display:none;">...</div>
    <textarea id="s_is_result_css" style="display:none;">...</textarea>
    <textarea id="s_index_off_css" style="display:none;">...</textarea>
    <div id="wrapper">
      <div class="s-skin-container s-isindex-wrap">
        <div id="head" class>
          <div id="s_top_wrap" class="s-top-wrap" style>...</div>
          <div id="s_upfunc_menus" class="s-upfunc-menus">...</div>
          <div id="u_sp" class="s-isindex-wrap s-sp-menu">...</div>
          <style>...</style>
          <div class="clear"></div>
          <div id="head_wrapper" class="s-isindex-wrap">...</div>
          <img ">...</div> == $0
          <div id="s_wrap" class="s-isindex-wrap">...</div>
```



<p class="title">.....</p>



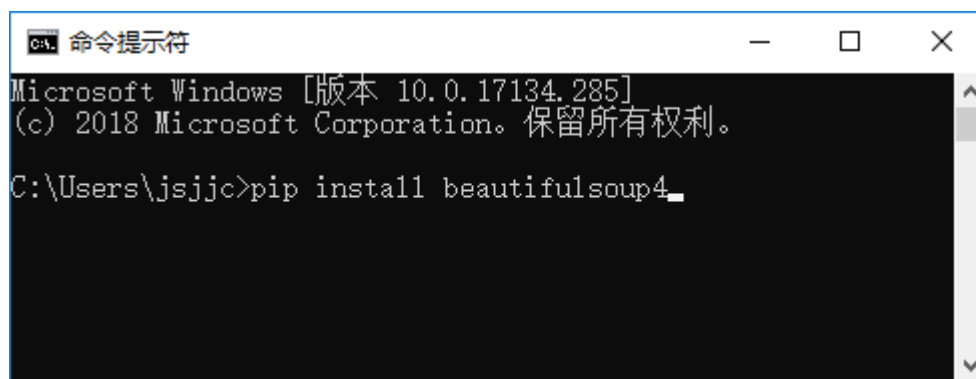
# BeautifulSoup库

- BeautifulSoup是解析、遍历、维护“标签树”的功能库
- BeautifulSoup库支持多种解析器，可方便地提取HTML或XML标签中的内容。
- Beautiful Soup提供一些简单的、python式的函数用来处理导航、搜索、修改分析树等功能。
- Beautiful Soup使用编码自动检测库识别输入文档编码并转换为Unicode编码，输出文档均转换为utf-8编码。

BeautifulSoup可以对任何标签类型的文件进行页面内容解析

# 安装和引用

- 在Windows平台以管理员身份运行cmd，运行下列语句进行库的安装：**pip install beautifulsoup4**



```
命令提示符
Microsoft Windows [版本 10.0.17134.285]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\jsjjc>pip install beautifulsoup4_
```

- 在Anaconda Prompt下安装：

**conda install beautifulsoup4**

- 引用方式

**from bs4 import BeautifulSoup**

**import bs4**

超文本标记语言

HyperText Markup Language

使用标签描述文档结构和表现形式

# HTML 网页结构

整个  
HTML文档

**<标签>内容</标签>**

`<html>`

`<head>`

`<title>页面标题</title>`

`</head>`

HTML文档  
的主体

`<body>`

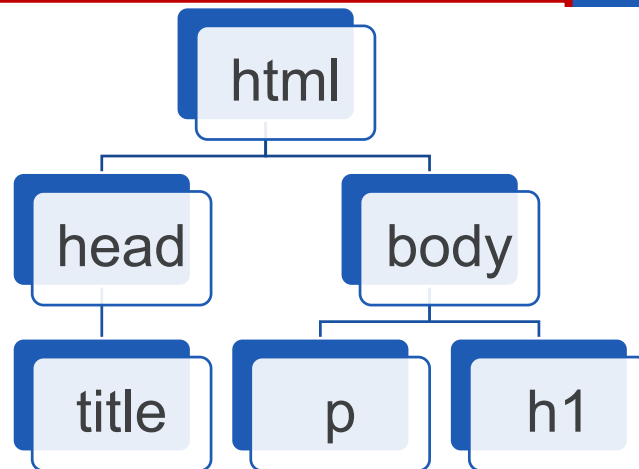
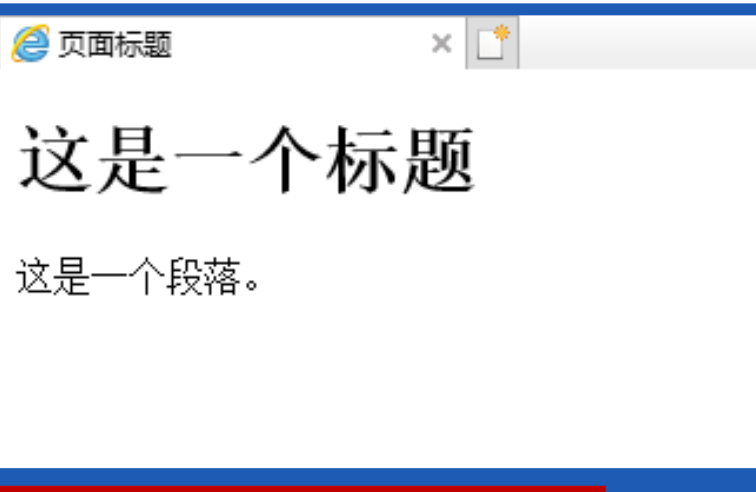
HTML文档中  
的一个段落

`<h1>这是一个标题</h1>`

`<p>这是一个段落。</p>`

`</body>`

`</html>`



嵌套

HTML元素以开始标签起始、以结束标签终止，  
元素内容是开始标签与结束标签之间的内容



# 创建 BeautifulSoup 对象

```
soup = BeautifulSoup(html, "html.parser")
```



汤



汤料



食谱

```
import requests
from bs4 import BeautifulSoup #导入 bs4 库

r = requests.get('http://news.baidu.com/')
soup = BeautifulSoup(r.text, "html.parser") # html 解析
print(soup.prettify()) # 格式化输出，增加换行符，分行显示
```

➤ 在浏览器中，右击“检查”，比较几种结果

# BeautifulSoup解析器

- BeautifulSoup可以解析多种格式的文档

```
soup = BeautifulSoup('<html>data</html>','html.parser') # html或xml文本
```

```
soup = BeautifulSoup(r.text,"html.parser") # response对象的text
```

```
soup = BeautifulSoup(open('a.html'),'html.parser') # 网页对象
```

解析器	使用方法	使用前提	特点
bs4的 html解析器	<b>html.parser</b>	pip install beautifulsoup4	执行速度适中；文档容错力强
lxml的 html解析器	<b>lxml</b>	pip install lxml	速度快； 文档容错能力强
lxml的 xml解析器	xml	pip install xml	速度快；唯一支持XML 的解析器
html5lib的解析器	html5lib	pip install html5lib	更好的容错性；以浏览器 的方式解析文档

<http://www.weather.com.cn/weather/101020100.shtml>



预报

## 实例—上海当日天气

顾村二手房出售	掉头发是缺啥	保姆一月多少钱	上海养老院	上海二手		
韩式三点是好吗	石斛有什么效果	哈尔滨到漠河	哈尔滨三日游	上海的整形医院		
福州房价	佛山房价	切下眼袋多少钱	千股千评个股	种牙的利弊	日本服务器	上

全国 > 上海 > 城区

11:30更新 | 数据来源 中央气象台

今天	7天	8-15天	40天 <small>hot</small>	雷达图
----	----	-------	------------------------	-----

2日 (今天)	3日 (明天)	4日 (后天)	5日 (周六)	6日 (周日)	7日 (周一)	8日 (周二)
多云	多云	晴转多云	多云	多云	多云	小雨转多云
13/8°C	11/6°C	11/5°C	11/7°C	14/9°C	15/10°C	15/10°C
3-4级	4-5级转<3级	<3级	<3级	<3级	<3级	<3级

## 实例—上海当日天气

```
import requests
from bs4 import BeautifulSoup

url="http://www.weather.com.cn/weather/101020100.shtml"
r=requests.get(url)
r.encoding=r.apparent_encoding
soup=BeautifulSoup(r.text,"html.parser")
name=soup.find_all(class_="sky skyid lv3 on")
for u in name:
    day=u.h1.string
    wea=u.find(class_="wea").text
    tem = u.find(class_="tem").get_text()
    win = u.find(attrs={"class":"win"}).get_text()
    content ="日期:"+day+ "天气:" + wea + " 温度:" + tem +"风力:"+win
    content = content.replace("\n","")
    print(content)
```

日期:3日 (今天) 天气:多云 温度:11/5℃ 风力:4-5级转<3级

- 上海7日天气预报？  
引入正则
- 任意城市7日天气预报？  
根据城市编码生成任意城市链接
- 任意城市周边地区当日天气？  
多个周边地区
- 任意城市周边地区7日天气预报？  
根据每个周边地区链接

## 实例—上海7日天气

```
import requests
from bs4 import BeautifulSoup
```

```
url="http://www.weather.com.cn/weather/101020100.shtml"
r=requests.get(url)
r.encoding=r.apparent_encoding
soup=BeautifulSoup(r.text,"html.parser")
name=soup.find_all(class_=re.compile("^sky skyid lv"))
```

```
for u in name:
    day=u.h1.string
    wea=u.find(class_="wea").text #text属性
    tem = u.find(class_="tem").get_text() #get_text方法
    win = u.find(attrs={"class":"win"}).get_text() #设置attrs
    content = "日期:"+day+"天气:" + wea + " 温度:" + tem + " 风力:"+win
    content = content.replace("\n","")
    print(content)
```

15日天气预报?



# 任意城市7日天气预报

- 搜索天气网城市编码
- 保存到txt文档，将“=>”替换成“:”
- 手动拷贝赋值给字典city

```
cName=input("城市名称：")  
print("{}7日天气预报".format(cName))  
code=city[cName]  
url="http://www.weather.com.cn/weather/"+code+".shtml"  
#调用自定义函数get7DaysWeather(url)  
get7DaysWeather(url)
```

```
def get7DaysWeather(url):  
    r=requests.get(url)  
    r.encoding=r.apparent_encoding  
    soup=BeautifulSoup(r.text,"html.parser")  
    name=soup.find_all(class_=re.compile("^sky skyid lv"))  
    for u in name:  
        day=u.h1.string  
        wea=u.find(class_="wea").text #text属性  
        tem = u.find(class_="tem").get_text() #get_text方法  
        win = u.find(attrs={"class":"win"}).get_text() #设置attrs  
        content = "日期:"+day+"天气:" + wea + " 温度:" + tem + " 风  
力:"+win  
        content = content.replace("\n","")
```

# 任意城市周边地区当日天气

- 函数getLink(cName)，根据城市名称得到城市链接

```
def getLink(cName):  
    fp=open(r"城市编码.txt","r")  
    r="{ "+fp.read()+"}"  
    cityCode=eval(r)  
    cityCode[cName]  
    code=city[cName]  
    url="http://www.weather.com.cn/weather/"+code+".shtml"  
    return url
```

# 任意城市周边地区当日天气

```
def getRegionInfo(url):  
    r=requests.get(url)  
    r.encoding=r.apparent_encoding  
    soup=BeautifulSoup(r.text,"html.parser")  
    info=soup.find('ul',class_="clearfix city")  
    info=info.find_all("li")  
    regionInfo={}  
    #获取每个周边地区的名称及其链接  
    for i in range(len(info)):  
        name=info[i].find('span').text  
        tem=info[i].find('i').text  
        #link=info[i].find('a').get('href').replace("#around2","")  
        regionInfo[name]=tem  
    return regionInfo
```

# 主函数

```
def main():  
    cName=input("城市名称：")  
    url=getLink(cName)  
    regionInfo=getRegionInfo(url)  
    for name,tem in regionInfo.items():  
        print("{}:气温{}".format(name,tem))  
main()
```

任意城市周边地区的  
7日天气预报？

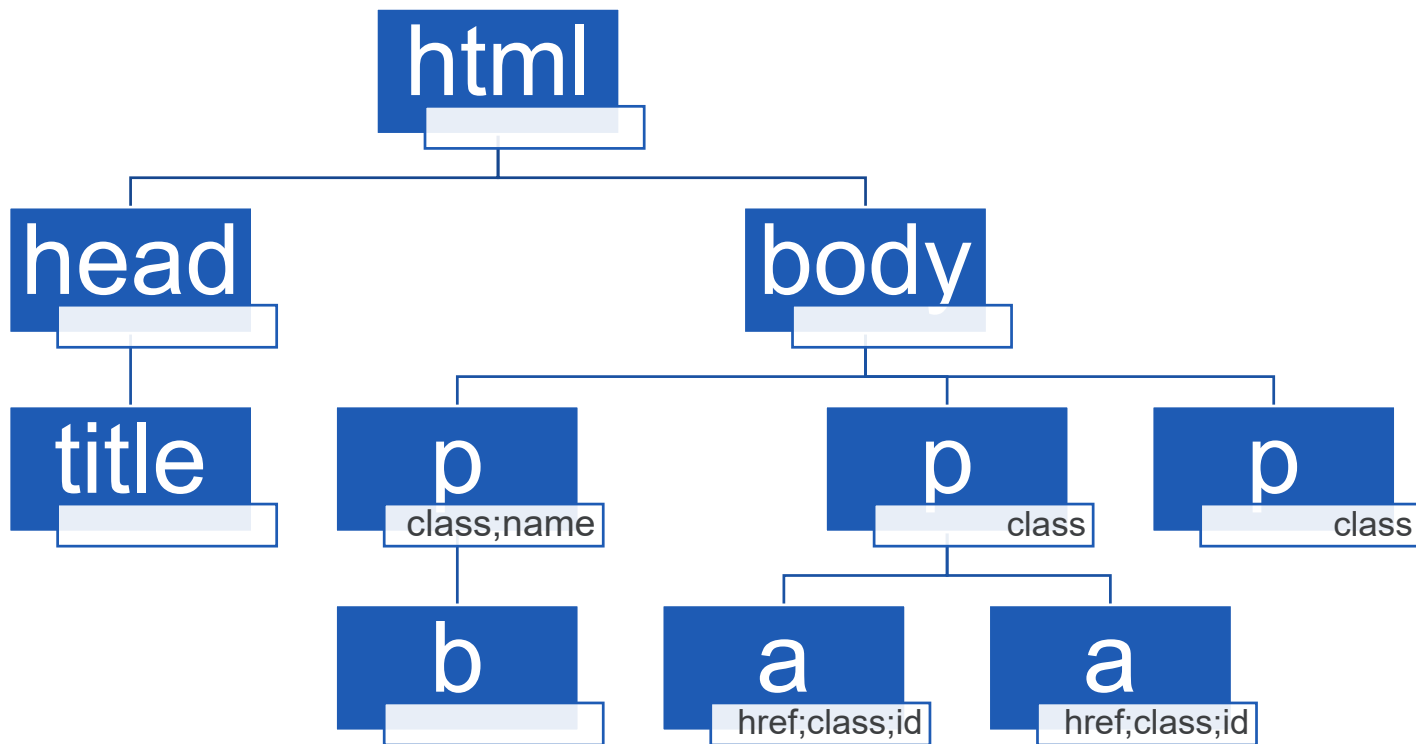
## 案例文档

```
html = """
<html><head><title>网络爬虫之学习</title></head>
<body>
<p class="title" name="spider"><b>爬虫之信息解析</b></p>
<p class="method">下面介绍了两种解析方法
<a href="http://example.com/bs" class="method"
id="link1">BeautifulSoup</a>,
<a href="http://example.com/re" class="method" id="link2">
正则表达式</a>,
两种方法各有特点</p>
<p class="result">...</p>
</body>
</html>"""
```



```
<html>
  <head>
    <title>
  </title>
  <body>
    <p>
      <b>
    </b>
    <p>
      <a>
    </a>
    <a>
  </a>
    <p>
  </p>
</html>
```

```
<html>
  <head>
    <title>网络爬虫之学习</title>
  </head>
  <body>
    <p class="title" name="spider">
      <b>爬虫之信息解析</b>
    </p>
    <p class="method">下面介绍了两种解析方法
      <a href="http://example.com/bs" class="method" id="link1">BeautifulSoup</a>,
      <a href="http://example.com/re" class="method" id="link2">正则表达式</a>,
      两种方法各有特点
    </p>
    <p class="result">...</p>
  </body>
</html>
```



# BeautifulSoup类的基本元素

基本元素	说 明
Tag	<b>标签</b> ，最基本的信息组织单元。用<>和</>标明开头和结尾
Name	标签的 <b>名字</b> ，比如 <a href = "xxx">，a就是标签名。格式： <b>&lt;tag&gt;.name</b>
Attributes	标签的 <b>属性</b> ，字典形式组织，格式： <b>&lt;tag&gt;.attrs</b>
NavigableString	标签内 <b>非属性字符串</b> ，<a></a>中间部分的字符串。格式： <b>&lt;tag&gt;.string</b>
Comment	标签内字符串的 <b>注释</b> 部分，<!--...-->中的部分字符串。

<a href="http://www.tongji.edu.cn" class="University" >TongJi<!--Uni--></a>

# Tag: 返回标签内容

## ➤ 创建 BeautifulSoup 对象

```
soup = BeautifulSoup(html)
```

直接用html文本

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

将本地 index.html 文件打开，用之创建 soup 对象

```
print(soup.prettify())
```

格式化输出 soup 对象的内容

## ➤ 获取标签

**soup.标签名**

```
soup.title
```

利用 soup 加标签名轻松地获取标签的内容

```
soup.head
```

```
soup.a
```

注：它查找的是在所有内容中的**第一个**符合要求的标签

```
soup.p
```

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

验证对象的类型：

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

```
<html>
```

```
<head>
```

```
<title>
```

```
    网络爬虫之学习
```

```
</title>
```

```
</head>
```

```
<body>
```

```
<p class="title" name="spider">
```

```
<b>
```

```
    爬虫之信息解析
```

```
</b>
```

```
</p>
```

```
<p class="method">
```

```
    下面介绍了两种解析方法
```

```
<a class="method" href="http://example.com/bs" id="link1">
```

```
    BeautifulSoup
```

```
</a>
```

soup.head

soup.title

注：查找的是在所有内容中的  
第一个符合要求的标签

soup.p

soup.a

type(soup.a)

name

tag

attrs

# Name: 标签名字 (字符串类型)

➤ 获取标签名字

**<tag>.name**

soup.head.name

head

内部标签，输出的值即为标签本身的名称

soup.p.name

p

soup.p.parent.name

body

soup.name

[document]

soup 对象本身比较特殊，它的 name 即为 [document]

<html>

<head>

<title>

soup.head.name

soup.title.name

网络爬虫之学习

</title>

soup.p.parent.name

</head>

<body>

<p class="title" name="spider">

<b>

soup.p.name

爬虫之信息解析

</b>

</p>

<p class="method">

soup.name

[document]

下面介绍了两种解析方法

<a class="method" href="http://example.com/bs"  
id="link1">

BeautifulSoup



# 属性 (Attributes) : 说明标签特点的区域

tag

name

attrs

➤ 获取标签属性

**<tag>.attrs**

p标签的所有属性；  
类型：字典

soup.p.**attrs**

{'class': ['title'], 'name': 'spider'}

soup.p['**class**']

['title']

单独获取class属性

soup.p.**get**('class')

['title']

利用get方法传入属性的名称

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

修改属性

**del** soup.p['class']

删除属性

soup.attrs 空字典

```
<html>
<head>
<title>
```

网络爬虫学习

删除属性

```
del soup.p['class']
```

p标签的所有属性

```
soup.p.attrs
```

单独获取class属性

```
{'class': ['title'], 'name': 'spider'}
```

```
</title>
</head>
<body>
```

```
<p class="newC" name="spider">
```

```
<b>
```

爬虫信息解析

```
soup.p['class']
```

```
</b>
```

修改属性

```
< soup.p['class']="newC"
```

```
soup.p.get('class')
```

利用get方法传入属性的名称

```
<p class="method">
```

下面介绍了两种解析方法

```
<a class="method" href="http://example.com/bs"
id="link1">
```

BeautifulSoup

# 内容 (NavigableString) : 标签内部字符串

**<tag>.string**

## ➤ 获取标签的内容

soup.b.**string**

type(soup.b.string)

soup.a.string

soup.p.string

爬虫之信息解析

<class 'bs4.element.NavigableString'>

BeautifulSoup

爬虫之信息解析

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。

soup.html.string

None

tag包含了多个子节点,tag无法确定

```
<head>
```

```
<title>
```

```
    网络爬虫之学习
```

```
</title>
```

```
</head>
```

```
<body>
```

```
<p class="title" name="spider">
```

```
<b>
```

```
    爬虫之信息解析
```

```
</b>
```

```
</p>
```

```
<p class="method">
```

```
    下面介绍了两种解析方法
```

```
<a class="method" href="http://example.com/bs"
id="link1">
```

```
    BeautifulSoup
```

```
</a>
```

soup.p.string

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。

soup.b.string

soup.a.string

soup.html.string

None

# Comment: 标签内字符串的注释部分

➤ 获取注释 <!--注释-->

```
soup.a.string
```

BeautifulSoup

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

<class 'bs4.element.Comment'>

注释是特殊的NavigableString对象

可利用 .string 来输出它的内容  
需通过类型区分两者。

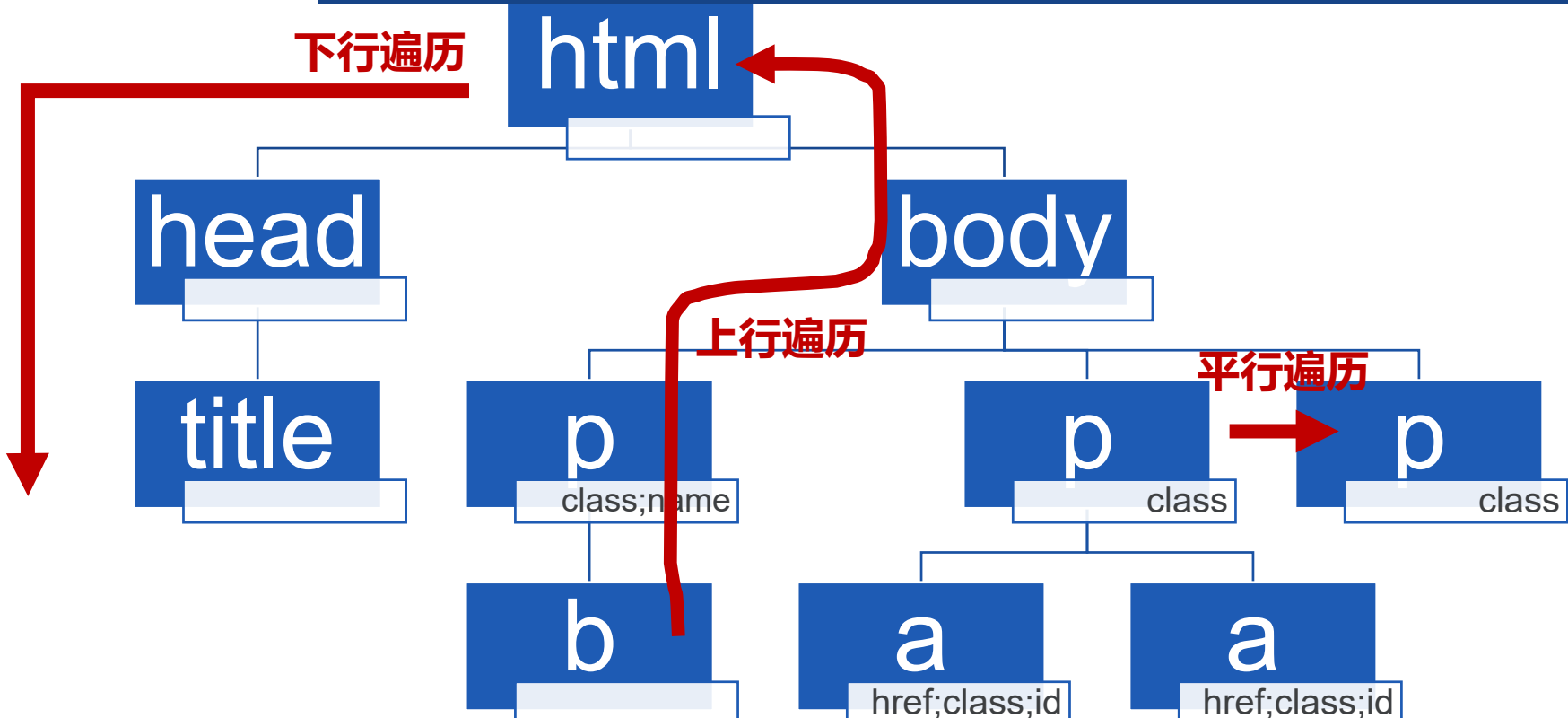
```
if type(soup.a.string)==bs4.element.Comment:
```

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

可以在使用前做一下判断

```
<html>
<head>
  <title>
</head>
<body>
  <p>
    <b>
  </p>
  <p>
    <a>
    <a>
  </p>
</body>
</html>
```

```
<html>
<head>
  <title>网络爬虫之学习</title>
</head>
<body>
  <p class="title" name="spider">
    <b>爬虫之信息解析</b>
  </p>
  <p class="method">下面介绍了两种解析方法
    <a href="http://example.com/bs" class="method" id="link1">BeautifulSoup</a>,
    <a href="http://example.com/re" class="method" id="link2">正则表达式</a>,
    两种方法各有特点
  </p>
  <p class="result">...</p>
</body>
</html>
```





# 基于bs4库的html内容遍历

遍历属性	说明
<b>.contents</b>	子节点的列表，将所有儿子节点存入列表
<b>.children</b>	子节点的迭代类型，与.contents类似，用于循环遍历儿子节点
<b>.descendants</b>	子孙节点的迭代类型，包含所有子孙节点，用于循环遍历
<b>.parent</b>	节点的父亲标签
<b>.parents</b>	节点的父辈标签，包含父亲，爷爷及以上
<b>.next_sibling</b>	返回按照html文本顺序的下一个平行节点标签
<b>.previous_sibling</b>	返回按照html文本顺序的上一个平行节点标签
<b>.next_siblings</b>	迭代类型，返回按照html文本顺序的后续所有平行节点标签
<b>.previous_siblings</b>	返回按照html文本顺序的前序平行节点标签

# 直接子节点 `.contents` `.children` 属性

- contents属性将tag的子节点以列表的方式输出

```
soup.head.contents
```

```
soup.body.contents
```

```
[<title>网络爬虫之学习</title>]
```

- 输出方式为列表，可以用列表索引来获取它的某一个元素

```
soup.body.contents[1]
```

- children返回的不是一个 list，是一个 list 生成器对象，可以通过遍历获取所有子节点

```
soup.head.children
```

```
list(soup.body.children)
```

```
for child in soup.body.children:
```

```
    print(child)
```

```
<list_iterator at 0xe713310>
```

```
<html>
```

```
<head>
```

```
<title>
```

```
    网络爬虫之学习
```

```
</title>
```

```
</head>
```

```
<body>
```

```
<p class="title" name="spider">
```

```
<b>
```

```
    爬虫之信息解析
```

```
</b>
```

```
</p>
```

```
<p class="method">
```

```
    下面介绍了两种解析方法
```

```
<a class="method" href="http://example.com/bs"
```

```
id="link1">
```

```
    BeautifulSoup
```

soup.head.contents

soup.head.contents[0]

soup.body.contents

soup.head.children

soup.body.contents[1]

# 所有子孙节点 .descendants 属性

- .descendants 属性可以对tag的所有子孙节点进行递归循环

```
for child in soup.body.descendants:  
    print(child.name)
```

注意：而.contents 和 .children 属性仅包含tag的直接子节点

## **.strings** **stripped\_strings** 属性

- 和string属性相比，**strings**获取多层内容，可遍历获取

```
for string in soup.strings:  
    print(string)
```

- 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(string)
```

# 父节点 **.parent** 属性

soup.p.parent.name

body

soup.title.parent.name

head

soup.title.string.parent.name

title

```
<html>
```

```
<head>
```

```
<title>
```

网络爬虫之学习

```
</title>
```

```
</head>
```

```
<body>
```

```
<p class="title" name="spider">
```

```
<b>
```

爬虫之信息解析

```
</b>
```

```
</p>
```

```
<p class="method">
```

下面介绍了两种解析方法

```
<a class="method" href="http://example.com/bs"
id="link1">
```

`soup.title.parent.name`

`soup.title.string.parent.name`

`soup.p.parent.name`

# 全部父节点 .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

```
for parent in soup.a.parents:
```

```
    print(parent.name)
```

```
for parent in soup.p.string.parents:
```

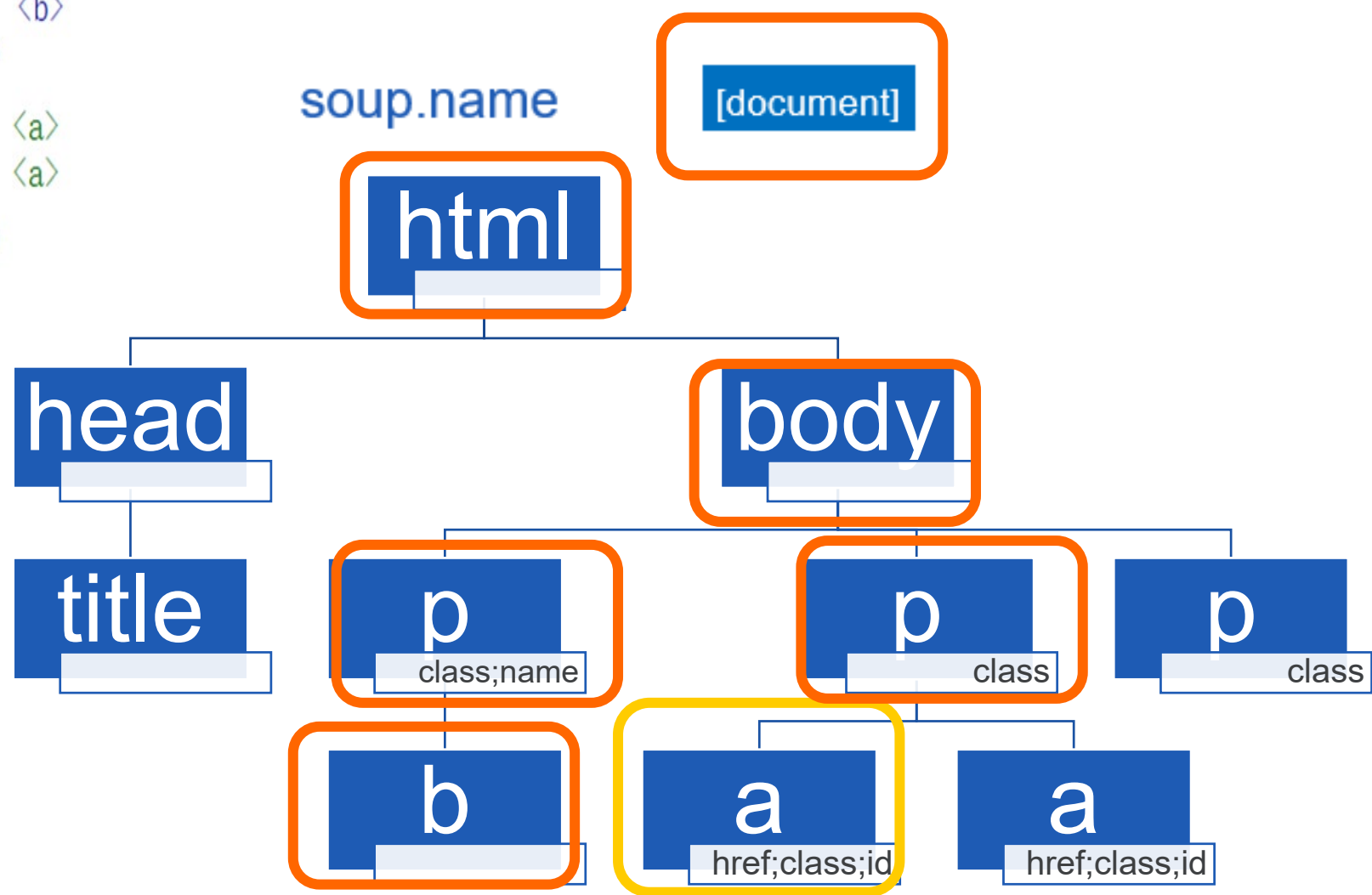
```
    print(parent.name)
```

```
p
body
html
[document]
```

```
b
p
body
html
[document]
```



```
<html>
<head>
  <title>
<body>
  <p>
    <b>
  <p>
    <a>
    <a>
  <p>
</html>
```



soup.p.string.parents

soup.a.parents

# 兄弟节点 `.next_sibling` `.previous_sibling` 属性

- `.next_sibling` 属性获取节点的下一个兄弟节点(同一级节点), `.previous_sibling` 则获取上一个兄弟节点。
- 注意: 实际文档中的`tag.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。

```
print(soup.p.next_sibling) #实际该处为空白
```

```
print(soup.p.next_sibling.next_sibling)
```

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

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

# **.next\_siblings** **.previous\_siblings** 属性

全部兄弟节点

```
for sibling in soup.a.next_siblings:  
    print(sibling.name)
```

None

a

None

a

None

(所有)前后节点

**.next\_element(s)**  
**.previous\_element(s)** 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

```
soup.p.next_element
```

- `.next_elements` 和 `.previous_elements` 的迭代器可以向前或向后访问文档的解析内容

```
for element in soup.p.next_elements:  
    print(element.name)
```

# find\_all()方法：搜索文档树

```
find_all(name, attrs, recursive, string, **kwargs)
```

可以根据标签，属性，内容查找文档。

find\_all()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

- **name 参数：**查找所有名字为 name 的 tag,字符串对象会被自动忽略掉。
- 1.传字符串：最简单的过滤器是字符串。BeautifulSoup会查找与字符串完整匹配的内容，

```
soup.find_all('b')
```

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

查找文档中所有的<b>标签

# Name参数

- **2.传正则表达式：**Beautiful Soup会通过正则表达式的 `match()` 来匹配内容。

```
import re
```

```
for tag in soup.find_all(re.compile("^b")):  
    print(tag.name)
```

找出所有以b开头的标签

body  
b

- **3.传列表：**Beautiful Soup会将与列表中任一元素匹配的内容返回

```
soup.find_all(["a", "b"])
```

找出文档中所有<a>标签和<b>标签

- **4.传True：**True 可以匹配任何值。

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

找出当前tag下所有的tag,但不返回字符串节点

# Name参数

- **5.传方法：**该方法只接受一个元素参数。当前元素匹配并且被找到则该方法返回 True 表示,反之则返回 False
- 例如：定义校验了当前元素,如果包含 class 属性且包含 id 属性,那么将返回 True:
  - ✓ 

```
def has_class_and_id(tag):  
    return tag.has_attr('class') and tag.has_attr('id')
```
- 将这个方法作为参数传入 find\_all() 方法,将得到所有符合条件的标签
  - ✓ 

```
soup.find_all(has_class_and_id)
```

# Attrs参数——keyword

- 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tag的指定名字的属性来搜索

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

搜索每个tag的” id”属性以进行匹配

- 如传入 href 参数,BeautifulSoup会搜索每个tag的” href”属性

```
soup.find_all(href=re.compile("example"))
```

- 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("exa"), id='link2')
```

- 如果想用 class 过滤, 由于 class 是 python 的关键词, 加个下划线

```
soup.find_all("a", class_="method")
```

- 还有某些属性不能搜索 (如data-\*), 可以定义字典参数进行搜索

```
soup2.find_all(attrs={"data-foo": "value"})
```



# Attrs参数——text

- 通过 text 参数可以搜索文档中的字符串内容

```
soup.find_all(text="正则表达式")
```

```
soup.find_all(text=["Beautiful Soup", "正则表达式"])
```

```
soup.find_all(text=re.compile("爬虫"))
```

## Attrs参数——limit

- limit 参数限制返回结果的数量，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

```
soup.find_all("a", limit=2)
```

# 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`

```
soup.html.find_all("a")
```

所有子孙

```
soup.html.find_all("a",recursive=False)
```

直接子孙

# find方法与find\_all的区别

- find\_all()返回的是所有元素列表，find()返回单个元素

```
find(name, attrs, recursive, string, **kwargs)
```

- find\_all()方法将返回文档中符合条件的所有 tag。有时只想得到一个结果，使用find\_all方法并设置limit=1参数不如直接使用find()方法。例如：

```
soup.find_all('title', limit=1)  
# [<title>网络爬虫之学习</title>]
```

```
soup.find('title')  
#<title>网络爬虫之学习</title>
```

- find\_all()方法的返回结果仍然是只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。

# 拓展

- 返回（所有/第一个）当前节点的父辈节点

`find_parents()`                      `find_parent()`

- 返回（所有/第一个）符合条件的后面的兄弟节点

`find_next_siblings()`    `find_next_sibling()`

- 返回（所有/第一个）符合条件的前面的兄弟节点

`find_previous_siblings()`                      `find_previous_sibling()`

- 返回之后（所有/第一个）符合条件的节点

`find_all_next()`                      `find_next()`

- 返回前面（所有/第一个）符合条件的节点

`find_all_previous ()`                      `find_previous ()`

# 豆瓣电影提取

← → ↻ <https://movie.douban.com/cinema/nowplaying/shanghai/> ☆ 人 ⓘ ⋮

影讯&购票 选电影 电视剧 排行榜 分

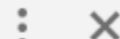


Elements

Console

Sources

>> 6



## 电影票 - 上海 [切换城市]

正在上映



大侦探皮卡丘

★★★★☆ 6.7

选座购票



复仇者联盟4: 终局之战

★★★★★ 8.6

选座购票



驯龙高手3

★★★☆☆

选座购票

▼<ul class="lists">

▼<li id="26835471" class="list-item" data-title="大侦探皮卡丘" data-score="6.7" data-star="35" data-release="2019" data-duration="104分钟" data-region="美国 日本" data-director="罗伯·莱特曼" data-actors="瑞恩·雷诺兹 / 贾斯蒂斯·史密斯 / 凯瑟琳·纽顿" data-category="nowplaying" data-enough="True" data-showed="True" data-votecount="90244" data-subject="26835471">

... div #nowplaying div ul #26835471 ul li a img

Styles

Computed

Event Listeners

DOM Breakpoints



Filter

:hov .cls



<https://movie.douban.com/cinema/nowplaying/shanghai/>

# 改进版

```
import requests
from bs4 import BeautifulSoup
url = "https://movie.douban.com/cinema/nowplaying/shanghai/"

# 获取页面信息
response = requests.get(url)
content = response.text
# 分析页面，获取ID和电影名称
soup = BeautifulSoup(content, 'html.parser')
# 找到所有的电影信息对应的LI标签
movie_list = soup.find_all('li', class_='list-item')
# 储存所有电影信息[{'title': '名称', 'id': 'id号'}]
movies_info = []
```



# 依次遍历每一个li标签，提取所需要的信息

for item in movie\_list:

    now\_movies\_dict = {}

    now\_movies\_dict['标题'] = item['data-title']

    now\_movies\_dict['ID'] = item['id']

    now\_movies\_dict['演员'] = item['data-actors']

    now\_movies\_dict['导演'] = item['data-director']

    movies\_info.append(now\_movies\_dict)

with open('movies.txt', 'w') as f:

    for item in movies\_info:

        f.write(str(item) + '\n')



← → ↻

 博客园  
cnblogs.com

代码

```
<div class="day">
  <div class="dayTitle">
    <a href="https://www.cnblogs.com/developer-huawei/archive/2021/07/23.html">2021年7月23日</a>
  </div>
```

<div class="postTitle">

```
<a class="postTitle2 vertical-middle" href="https://www.cnblogs.com/developer-huawei/p/15048142.html">
  <span>
    玩转AR，让电商营销锦上添花
  </span>
</a>
```

摘要：随着科技的发展，从平面媒体到广播、电视、电脑、智能手机再到现在的虚拟现实设备，科技在时间和空间上不断解放人类。《星球大战》、《阿凡达》、《黑客帝国》、《头号玩家》，从这些众多的科幻电影来看，再造一个世界始终是人类梦想，人们以存在哲学为理论基础，发展虚拟世界的理论、技术和伦理。如今，图形学、多媒体、[阅读全文](#)

posted @ 2021-07-23 11:30 华为开发者论坛 阅读(3) 评论(0) 推荐(0) 编辑

摘要：随着科技的发展，从平面媒体到广播，电视，电脑，智能手机再到现在的虚拟现实设备，科技在时间和空间上不断解放人类。

摘要: 现象描述: 当text组件的内容较多且显示多行的时候, 相邻的div样式会显示异常, 会从正常的圆形变为椭圆。问题代码如下: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 3 [阅读全文](#)

posted @ 2021-07-22 14:48 华为开发者论坛 阅读(1780) 评论(0) 推荐(0) 编辑

摘要：万物互联时代已来！HarmonyOS的全新上线，为应用服务带来哪些新机遇？为开发者带来哪些新体验？2021年7月31日，HDD | HarmonyOS 开发者日将在杭州举办。我们为广大开发者准备了最硬核的技术盛宴，轻松玩转跨终端、轻量化万能卡片功能、获取丰富Code Labs开发样例、了解更多生态场 [阅读全文](#)

posted @ 2021-07-22 14:40 华为开发者论坛 阅读(1890) 评论(0) 推荐(0) 编辑

```
from bs4 import BeautifulSoup
import requests
#请求博客园首页
r=requests.get('https://www.cnblogs.com/developer-huawei/')
#使用html.parser解析html
soup=BeautifulSoup(r.content,'html.parser')
#论坛标题
t=soup.find("a",id="Header1_HeaderTitle").get_text()
print("博客标题：",t)
#文章日期
tags=soup.find_all(class_="dayTitle")
#文章标题
titles=soup.find_all("div",class_="postTitle")
#文章摘要
abstracts=soup.find_all(class_="c_b_p_desc")
#显示获取信息
for i in range(len(tags)):
    print("日期:",tags[i].a.string)
    print("文章:",titles[i].a.text.strip())
    print(abstracts[i].contents[0].strip()+"\n')
```

# 信息提取一般方法

```
for link in soup.find_all("a"):
    print(link.get_text())
```

使用get\_text()方法  
获得标签内容

Lacie  
Tillie  
Beautiful  
正则表达式

```
for link in soup.find_all("a"):
    print (link.get("href"))
```

```
http://example.com/elsie
http://example.com/lacie
http://example.com/tillie
```

使用get()方法获得标  
签的href属性

# 实例一豆瓣图书Top250信息获取

- 豆瓣图书Top250，地址：<https://book.douban.com/top250?start=0>

获取豆瓣读书的robots协议

<https://book.douban.com/robots.txt>

目标信息？

## 豆瓣图书 Top 250

源代码？ Request



追风筝的人

可试读

书名

The Kite Runner

作者出版社等

[美] 卡勒德·胡赛尼 / 李继宏 / 上海人民出版社 / 2006-5 / 29.00元

★★★★★ 8.9

( 349046人评价 )

评分

“ 为你，千千万万遍 ”

简介

如何获取目标数据？

BeautifulSoup



解忧杂货店

ナミヤ雑貨店の奇蹟

[日] 东野圭吾 / 李盈春 / 南海出版公司 / 2014-5 / 39.50元

```
<div class="pl2">
```

```
<a href="https://book.douban.com/subject/1770782/" onclick="" moreurl(this,{i:'0'})"" title="追风筝的人">
```

追风筝的人

书名信息包含在class='pl2' div里面的a标签内，是a标签的title属性

```
</a>
```

```
"
```

```
&nbsp; "

```

```

```

```
<br>
```

```
<span style="font-size:12px;">The Kite Runner</span>
```

```
</div>
```

```
<p class="pl">[美] 卡勒德·胡赛尼 / 李继宏 / 上海人民出版社 / 2006-5 / 29.00元</p>
```

```
<div class="star clearfix">
```

```
<span class="allstar45"></span>
```

```
<span class="rating_nums">8.9</span>
```

```
<span class="pl">(
```

349046人评价


```
)</span>
```

```
::after
```

```
</div>
```

```
<p class="quote" style="margin: 10px 0; color: #666">
```

```
<span class="inq">为你，千千万万遍</span> == $0
```



```
import requests
#获取html信息
resp = requests.get('https://book.douban.com/top250?start=0')
print(resp.text)
from bs4 import BeautifulSoup
#解析信息, 用html.parser或者lxml
soup = BeautifulSoup(resp.text, 'lxml')
#书名
alldiv = soup.find_all('div', class_='pl2')
for a in alldiv:
    names = a.find('a')['title']
    print('find_all():', names)
```



#作者信息

#获取到p标签, 直接用get\_text()方法获得文本内容

```
allp = soup.find_all('p', class_='pl')
```

```
for p in allp:
```

```
    authors = p.get_text()
```

```
    print('find_all():', authors)
```

#评分

```
starspan = soup.find_all('span', class_='rating_nums')
```

```
for s in starspan:
```

```
    scores = s.get_text()
```

```
    print('scores:', scores)
```

#简介

```
sumspan = soup.find_all('span', class_='inq')
```

```
for s2 in sumspan:
```

```
    sums = s2.get_text()
```

```
    print('sums:', sums)
```



#或者：用list形式获得所有的书名、作者、评分、简介

```
names = [a.find('a')['title'] for a in alldiv]
```

```
authors = [p.get_text() for p in allp]
```

```
scores = [s.get_text() for s in starspan]
```

```
sums = [i.get_text() for i in sumspan]
```

#用zip函数，以一个或多个序列为参数，返回元组列表，同时将这些序列中并排的元素配对。

```
for name, author, score, sum in zip(names, authors, scores, sums):
```

```
    name = '书名：' + str(name) + '\n'
```

```
    author = '作者：' + str(author) + '\n'
```

```
    score = '评分：' + str(score) + '\n'
```

```
    sum = '简介：' + str(sum) + '\n'
```

```
    data = name + author + score + sum
```

问题？

完整代码



# CSS选择器

- 在Tag或BeautifulSoup对象的.select()方法中传入字符串参数，可使用CSS选择器的语法找到 tag，返回类型为list

- 1.通过标签名查找（不加任何修饰）

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

```
Print(soup.select('a'))
```

```
print(soup.select('b'))
```

- 2.通过类名查找（类名前加点.）

```
soup.select('.sister')
```

- 3.通过 id 名查找（id名前加 #）

```
soup.select('#link1')
```

➤ 4.组合查找（标签名与.类名、#id名进行组合。用空格分开）

```
soup.select('p #link1')
```

查找 p 标签中，id 等于 link1的内容

```
soup.select('html title')
```

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

直接子标签，用 > 连接，注意两边空格

➤ 5.属性查找（属性用[ ]括起来；属性和标签属于同一节点；不能加空格）

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

```
soup.select('body a[href="http://example.com/re"]')
```

综合属性、组合查找，不在同一节点的空格隔开，同一节点的不加空格