**requests**



Requests库的get（）方法



Requests库的Response对象：

Response对象包含服务器返回的所有信息，也包含请求的Request信息。



修改headers字段、cookie、代理

cookies={}

#自动保持  
s = requests.session()

#打开cookie文件  
*with* open('E:\example\豆瓣读书爬虫\cookie.txt')*as* file:  
 raw\_cookies=file.read()  
 *for* line *in* raw\_cookies.split(';'):  
 key,value=line.split('=',1)  
 cookies[key]=value  
proxie = {'http': 'http://14.118.254.31:6666'}

headers={'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:58.0) Gecko/20100101 Firefox/58.0'}  
res=s.get(url=*url*,cookies=cookies,headers=headers,proxies=proxie)

修改params参数提交关键词

百度的关键词接口：http://www.baidu.com/s?wd=keyword

360的关键词接口：<http://www.so.com/s?q=keyword>

url="http://www.baidu.com/s"   
kv={'wd':'Python'}   
r=requests.get(url,params=kv)

获取cookie

response = requests.get("https://www.baidu.com")  
print(response.cookies)

发送cookie

url = 'http://httpbin.org/cookies'  
cookies = {'testCookies\_1': 'Hello\_Python3', 'testCookies\_2': 'Hello\_Requests'}  
# 在Cookie Version 0中规定空格、方括号、圆括号、等于号、逗号、双引号、斜杠、问号、@，冒号，分号等特殊符号都不能作为Cookie的内容。  
r = requests.get(url, cookies=cookies)

证书验证

response = requests.get('https://www.12306.cn', verify=*False*)

超时设置

*from* requests.exceptions *import* ReadTimeout  
*try*:  
 response = requests.get("http://httpbin.org/get", timeout = 0.5)  
 print(response.status\_code)  
*except* ReadTimeout:  
 print('Timeout')

**BeautifulSoup**

创建Beautiful Soup对象

Beautiful Soup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:

* Tag
* NavigableString
* BeautifulSoup
* Comment

**Beautiful Soup获取 Tags**

它查找的是在所有内容中的第一个符合要求的标签

Tag：soup.标签名

Tag有两个重要的属性：name和attrs。

soup.标签名.name/attrs

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 标签的所有属性打印输出，得到的类型是一个字典。

print soup.p.attrs

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

如果需要单独获得某个属性可以使用字典的打印方法或者get(‘key’)

print soup.p['class']

print soup.p.get('class')

可以对这些属性和内容等等进行修改 soup.p['class']="newClass"

对这个属性进行删除: del soup.p['class']

NavigableString：获取标签内部的文字 soup.标签名.string

BeautifulSoup：表示的是一个文档的全部内容

Comment：特殊类型的NavigableString对象，首先判断的类型是否为 Comment 类型，然后再进行其他操作，如打印输出

if type(soup.li.string) == element.Comment:

print(soup.li.string)

**遍历文档树**

1. 直接子节点(不包含孙节点)
2. .contents

tag的content属性可以将tag的子节点以列表的方式输出：soup.标签名.contents

可以用列表索引来获取它的某一个元素: soup.标签名.contents[1]

1. .children

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

for child in soup. 标签名.children:

print(child)

1. 所有子孙节点

.descendants 属性

.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

1. 节点内容

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。）

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

1. 多个内容

.strings .stripped\_strings 属性

for string in soup.stripped\_strings:

print(repr(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

1. 父节点
2. .parent 属性

content = soup.head.title.string

print content.parent.name

1. .parents 属性

递归得到元素的所有父辈节点

content = soup.head.title.string

for parent in content.parents:

print parent.name

1. 兄弟节点
2. .next\_sibling .previous\_sibling 属性

.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

1. .next\_siblings .previous\_siblings 属性

对当前节点的兄弟节点迭代输出

for sibling in soup.a.next\_siblings:

print(repr(sibling))

1. 前后节点
2. .next\_element .previous\_element 属性

所有节点，不分层次

比如 head 节点为

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

print soup.head.next\_element

那么它的下一个节点便是 title，它是不分层次关系的

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

1. .next\_elements .previous\_elements 属性

可以向前或向后访问文档的解析内容

for element in last\_a\_tag.next\_elements:

print(repr(element))

**搜索文档树**

1. **find\_all(name, attrs, recursive, text, limit, \*\*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

查找所有名字为name的tag,字符串对象会被自动忽略掉

1. 传递字符：soup.find\_all('b')

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

1. 传递正则表达式：通过正则表达式的 match() 来匹配内容

for tag in soup.find\_all(re.compile("^b")):

print(tag.name)

# body

# b

1. 传递列表：将与列表中任一元素匹配的内容返回

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>]

1. 传递True：可以匹配任何值，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

1. 传递方法：可以定义一个方法,方法只接受一个元素参数,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

def has\_class\_but\_no\_id(tag): 包含 class 属性却不包含 id 属性

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

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>]

1. attrs参数：定义一个字典参数来搜索包含特殊属性的tag

soup.find\_all (attrs={"data-foo": "value"})

1. 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("title", recursive=False)

1. text参数：搜索文档中的字符串，接受字符串，正则表达式 ，列表，True

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"]

1. limit参数：find\_all() 方法返回全部的搜索结构，使用 limit 参数限制返回结果的数量

soup.find\_all("a", limit=2)

1. kwargs参数：kwargs 接收字符串，正则表达式

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

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')

soup.find\_all("a", class\_="sister") 使用class 过滤，加个下划线

1. **find( name , attrs , recursive , text , \*\*kwargs )**

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

1. **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

1. **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

1. **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

1. **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

1. **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

**css选择器 soup.select()，返回类型是 list**

1. 通过标签名查找 soup.select('title')
2. 通过类名查找 soup.select('.sister')
3. 通过 id 名查找 soup.select('#link1')
4. 组合查找 soup.select('p #link1') p 标签中，id 等于 link1的内容

空格分开

直接子标签查找 soup.select("head > title")

1. 属性查找 soup.select('a[class="sister"]')

属性和标签属于同一节点，中间不能加空格

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

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

soup = BeautifulSoup(html, 'lxml')

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

print soup.select('title')[0].get\_text()

for title in soup.select('title'):

print title.get\_text()

**selenium**

自动化测试工具。它支持各种浏览器。如果用Selenium来驱动浏览器加载网页的话，就可以直接拿到JavaScript渲染的结果了

1. **声明浏览器对象**

的初始化并将其赋值为browser对象

from selenium import webdriver

browser = webdriver.Chrome()

browser = webdriver.Firefox()

browser = webdriver.Edge()

browser = webdriver.PhantomJS()

browser = webdriver.Safari()

1. **访问页面**

get()方法来请求网页，参数传入链接URL即可。

browser.get('https://www.taobao.com')

browser.page\_source page\_source输出网页源码

1. **查找节点**
2. 获取单个节点

find\_element\_by\_id

find\_element\_by\_name

find\_element\_by\_xpath

find\_element\_by\_link\_text

find\_element\_by\_partial\_link\_text

find\_element\_by\_tag\_name

find\_element\_by\_class\_name

find\_element\_by\_css\_selector CSS选择器

通用方法find\_element()，它需要传入两个参数：查找方式By和值

find\_element\_by\_id(id)就等价于find\_element(By.ID, id)

1. 获取多个节点

如果有多个节点，用find\_element()方法查找，就只能得到第一个节点

find\_elements\_by\_id

find\_elements\_by\_name

find\_elements\_by\_xpath

find\_elements\_by\_link\_text

find\_elements\_by\_partial\_link\_text

find\_elements\_by\_tag\_name

find\_elements\_by\_class\_name

find\_elements\_by\_css\_selector

1. **节点交互**

输入文字时用send\_keys()方法，清空文字时用clear()方法，点击按钮时用click()方法。

1. **动作链**

一些交互动作都是针对某个节点执行的。比如，对于输入框，我们就调用它的输入文字和清空文字方法。

一些操作没有特定的执行对象，比如鼠标拖曳、键盘按键等，这些动作用动作链来执行

*from* selenium *import* webdriver  
*from* selenium.webdriver *import* ActionChains  
  
browser = webdriver.Chrome()  
url = 'http://www.runoob.com/try/try.php?filename=jqueryui-api-droppable'  
browser.get(url)  
browser.switch\_to.frame('iframeResult')  
source = browser.find\_element\_by\_css\_selector('#draggable')  
target = browser.find\_element\_by\_css\_selector('#droppable')  
actions = ActionChains(browser)  
actions.drag\_and\_drop(source, target)  
actions.perform()

首先，打开网页中的一个拖曳实例，然后依次选中要拖曳的节点和拖曳到的目标节点，接着声明ActionChains对象并将其赋值为actions变量，然后通过调用actions变量的drag\_and\_drop()方法，再调用perform()方法执行动作，此时就完成了拖曳操作

1. **执行JavaScript**

browser = webdriver.Chrome()  
browser.get('https://www.zhihu.com/explore')  
browser.execute\_script('window.scrollTo(0,document.body.scrollHeight)')  
browser.execute\_script('alert("To Bottom")')

execute\_script()方法将进度条下拉到最底部，然后弹出alert提示框

1. **获取节点信息**

通过page\_source属性可以获取网页的源代码，接着就可以使用解析库（如正则表达式、Beautiful Soup、pyquery等）来提取信息

Selenium返回的是WebElement类型，也有相关的方法和属性来直接提取节点信息

1. 获取属性

选中这个节点，**get\_attribute()**方法来获取节点的属性，通过get\_attribute()方法，传入想要获取的属性名，就可以得到它的值了。

logo = browser.find\_element\_by\_id('zh-top-link-logo')

print(logo.get\_attribute('class'))

1. 获取文本值

每个WebElement节点都有text属性，直接调用这个属性就可以得到节点内部的文本信息，这相当于Beautiful Soup的get\_text()方法、pyquery的text()方法

browser.get(url)

input = browser.find\_element\_by\_class\_name('zu-top-add-question')

print(input.text)

1. 获取id、位置、标签名和大小

id属性可以获取节点id，location属性可以获取该节点在页面中的相对位置，tag\_name属性可以获取标签名称，size属性可以获取节点的大小，也就是宽高

input = browser.find\_element\_by\_class\_name('zu-top-add-question')

print(input.id)

print(input.location)

print(input.tag\_name)

print(input.size)

1. 切换Frame

网页中有一种节点叫作iframe，也就是子Frame，相当于页面的子页面，它的结构和外部网页的结构完全一致

Selenium打开页面后，它默认是在父级Frame里面操作，而此时如果页面中还有子Frame，它是不能获取到子Frame里面的节点的。使用switch\_to.frame()方法来切换Frame

browser.get(url)

browser.switch\_to.frame('iframeResult')

1. 延时等待

get()方法会在网页框架加载结束后结束执行，此时如果获取page\_source，可能并不是浏览器完全加载完成的页面

1. 隐式等待 implicitly\_wait()

当查找节点而节点并没有立即出现的时候，隐式等待将等待一段时间再查找DOM，默认的时间是0，超出设定时间后，则抛出找不到节点的异常

browser = webdriver.Chrome()  
browser.implicitly\_wait(10)  
browser.get('https://www.zhihu.com/explore')  
input = browser.find\_element\_by\_class\_name('zu-top-add-question')

1. 显式等待

它指定要查找的节点，然后指定一个最长等待时间。如果在规定时间内加载出来了这个节点，就返回查找的节点；如果到了规定时间依然没有加载出该节点，则抛出超时异常

*from* selenium *import* webdriver  
*from* selenium.webdriver.common.by *import* By  
*from* selenium.webdriver.support.ui *import* WebDriverWait  
*from* selenium.webdriver.support *import* expected\_conditions *as* EC  
browser = webdriver.Chrome()  
browser.get('https://www.taobao.com/')  
wait = WebDriverWait(browser, 10)  
input = wait.until(EC.presence\_of\_element\_located((By.ID, 'q')))  
button = wait.until(EC.element\_to\_be\_clickable((By.CSS\_SELECTOR, '.btn-search')))  
print(input, button)

引入WebDriverWait这个对象，指定最长等待时间

调用它的until()方法，传入要等待条件expected\_conditions

presence\_of\_element\_located这个条件，代表节点出现的意思，其参数是节点的定位元组，也就是ID为q的节点搜索框

效果就是，在10秒内如果ID为q的节点（即搜索框）成功加载出来，就返回该节点；如果超过10秒还没有加载出来，就抛出TimeoutException异常。

| **等待条件** | **含义** |
| --- | --- |
| title\_is | 标题是某内容 |
| title\_contains | 标题包含某内容 |
| presence\_of\_element\_located | 节点加载出来，传入定位元组，如(By.ID, 'p') |
| visibility\_of\_element\_located | 节点可见，传入定位元组 |
| visibility\_of | 可见，传入节点对象 |
| presence\_of\_all\_elements\_located | 所有节点加载出来 |
| text\_to\_be\_present\_in\_element | 某个节点文本包含某文字 |
| text\_to\_be\_present\_in\_element\_value | 某个节点值包含某文字 |
| frame\_to\_be\_available\_and\_switch\_to\_it | 加载并切换 |
| invisibility\_of\_element\_located | 节点不可见 |
| element\_to\_be\_clickable | 节点可点击 |
| staleness\_of | 判断一个节点是否仍在DOM，可判断页面是否已经刷新 |
| element\_to\_be\_selected | 节点可选择，传节点对象 |
| element\_located\_to\_be\_selected | 节点可选择，传入定位元组 |
| element\_selection\_state\_to\_be | 传入节点对象以及状态，相等返回True，否则返回False |
| element\_located\_selection\_state\_to\_be | 传入定位元组以及状态，相等返回True，否则返回False |
| alert\_is\_present | 是否出现警告 |

1. 前进和后退 back()方法后退， forward()方法前进
2. cookies

browser = webdriver.Chrome()  
browser.get('https://www.zhihu.com/explore')  
browser.get\_cookies()  
browser.add\_cookie({'name': 'name', 'domain': 'www.zhihu.com', 'value': 'germey'})  
browser.delete\_all\_cookies()

加载完成后，浏览器实际上已经生成Cookies了。接着，调用get\_cookies()方法获取所有的Cookies。

添加一个Cookie，这里传入一个字典，有name、domain和value等内容

接下来，再次获取所有的Cookies。可以发现，结果就多了这一项新加的Cookie。

调用delete\_all\_cookies()方法删除所有的Cookies。

1. 选项卡管理

browser = webdriver.Chrome()  
browser.get('https://www.baidu.com')  
browser.execute\_script('window.open()')  
print(browser.window\_handles)  
browser.switch\_to\_window(browser.window\_handles[1])

调用execute\_script()方法，传入window.open()这个JavaScript语句新开启一个选项卡。

window\_handles属性获取当前开启的所有选项卡，返回的是选项卡的代号列表

切换选项卡，调用switch\_to\_window()方法即可，其中参数是选项卡的代号

1. 异常处理

可以使用try except语句来捕获各种异常, 一旦出现错误，就进行异常处理

browser = webdriver.Chrome()  
*try*:  
 browser.get('https://www.baidu.com')  
*except* TimeoutException:  
 print('Time Out')

CREATE TABLE `allbooks` (

`title` char(255) NOT NULL,

`author` char(255) DEFAULT NULL,

`money` char(255) DEFAULT NULL,

`publish` char(255) DEFAULT NULL,

`pages` char(255) DEFAULT NULL,

`years` char(255) DEFAULT NULL,

`ISBN` char(255) DEFAULT NULL,

`people` char(255) DEFAULT NULL,

`score` char(255) DEFAULT NULL,

) ENGINE=InnoDB DEFAULT CHARSET=utf8;