Beautiful Soup

和 lxml 一样，Beautiful Soup 也是一个 HTML/XML 的解析器，主要的功能也是如何解析和提取 HTML/XML 数据。

lxml 只会局部遍历，而 Beautiful Soup 是基于 HTML DOM 的，会载入整个文档，解析整个 DOM 树，因此时间和内存开销都会大很多，所以性能要低于 lxml。

BeautifulSoup 用来解析 HTML 比较简单，API 非常人性化，支持 CSS 选择器、Python 标准库中的 HTML 解析器，也支持 lxml 的 XML 解析器。

Beautiful Soup 3 目前已经停止开发，推荐现在的项目使用 Beautiful Soup 4。使用 pip 安装即可：pip install beautifulsoup4

官方文档：http://beautifulsoup.readthedocs.io/zh_CN/v4.4.0

快速使用

通过下面的一个例子，对 bs4 有个简单的了解，以及看一下它的强大之处：

```
from bs4 import BeautifulSoup

html = '''
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><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 Beautiful Soup 对象
soup = BeautifulSoup(html,'lxml')

#格式化输出 soup 对象的内容
print(soup.prettify())

print(soup.title)
print(soup.title.name)
print(soup.title.string)
print(soup.title.parent.name)
print(soup.p)
```

```
print(soup.p["class"])
print(soup.a)
print(soup.find_all('a'))
print(soup.find(id='link3'))
```

```
<html>
 <head>
  <title>
   The Dormouse's story
  </title>
 </head>
 <body>
  <p class="title">
   <b>
    The Dormouse's story
   </b>
  </p>
  <p class="story">
   Once upon a time there were three little sisters; and their names we
re
   <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>
<title>The Dormouse's story</title>
title
The Dormouse's story
head
<p class="title"><b>The Dormouse's story</b></p>
['title']
<a class="sister" href="http://example.com/elsie" id="link1">Elsie</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>]
<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

使用 BeautifulSoup 解析这段代码,能够得到一个 BeautifulSoup 的对象,并能按照标准的缩进格式的结构输出。

同时我们通过下面代码可以分别获取所有的链接，以及文字内容：

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

print(soup.get_text())
```

解析器

Beautiful Soup 支持 Python 标准库中的 HTML 解析器,还支持一些第三方的解析器，如果我们不安装它，则 Python 会使用 Python 默认的解析器，lxml 解析器更加强大，速度更快，推荐安装。

下面是常见解析器：

解析器	使用方法	优势	劣势
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 2.7.3 or 3.2.2) 前的版本中文档容错能力差
lxml HTML 解析器	`BeautifulSoup(markup, "lxml")`	• 速度快 • 文档容错能力强	• 需要安装C语言库
lxml XML 解析器	`BeautifulSoup(markup, ["lxml", "xml"])` `BeautifulSoup(markup, "xml")`	• 速度快 • 唯一支持XML的解析器	• 需要安装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	• 速度慢 • 不依赖外部扩展

推荐使用 lxml 作为解析器,因为效率更高. 在 Python2.7.3 之前的版本和 Python3 中 3.2.2 之前的版本,必须安装 lxml 或 html5lib, 因为那些 Python 版本的标准库中内置的 HTML 解析方法不够稳定.

基本使用

标签选择器

```
<html><head><title>The story</title></head>
<body>
<p class="title"><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>
```

上面的 title head a p 等等都是 HTML 标签

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

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

```
<title>The Dormouse's story</title>
<head><title>The Dormouse's story</title></head>
<p class="title"><b>The Dormouse's story</b></p>

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

通过这种我们使用 soup.标签名就可以获得这个标签的内容

这里有个问题需要注意，通过这种方式获取标签，如果文档中有多个这样的标签，返回的结果是第一个标签的内容，如上面我们通过 soup.p 获取 p 标签，而文档中有多个 p 标签，但是只返回了第一个 p 标签内容

获取名称 name

当我们通过 soup.title.name 的时候就可以获得该 title 标签的名称，即 title

获取属性 attrs

```
# 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。
print(soup.p.attrs)

# 以下这两种方式都能获取 p 标签的 class 属性值
print(soup.p.attrs['class'])
print(soup.p['class'])
```

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

获取内容 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

```
print(soup.p.string)

# 嵌套获取
print(soup.head.title.string)
```

结果就可以获取第一个 p 标签的内容：

```
The Dormouse's story

The story
```

子节点和子孙节点

1) contents

```
html = """
<html>
    <head>
        <title>The Dormouse's story</title>
    </head>
    <body>
        <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
">
                <span>Elsie</span>
            </a>
            <a href="http://example.com/lacie" class="sister" id="link2
">Lacie</a>
            and
            <a href="http://example.com/tillie" class="sister" id="link
3">Tillie</a>
            and they lived at the bottom of a well.
        </p>
        <p class="story">...</p>
"""

from bs4 import BeautifulSoup

soup = BeautifulSoup(html,'lxml')
print(soup.p.contents)
```

结果是将 p 标签下的所有子节点存入到了一个列表中

2) children

使用 children 也可以获取 p 标签下的所有子节点内容，和通过 contents 获取的结果是一样的，但是不同的地方是 soup.p.children 是一个迭代对象，而不是列表，只能通过循环的方式获取素有的信息

```
print(soup.p.children)
for i,child in enumerate(soup.p.children):
    print(i,child)
```

通过 contents 以及 children 都是获取子节点，如果想要获取子孙节点可以通过 descendants
print(soup.descendants)同时这种获取的结果也是一个迭代器

3) 父节点和祖先节点

通过 soup.a.parent 就可以获取父节点的信息

通过 list(enumerate(soup.a.parents))可以获取祖先节点，这个方法返回的结果是一个列表，会分别将 a 标签的父节点的信息存放到列表中，以及父节点的父节点也放到列表中，并且最后还会讲整个文档放到列表中，所有列表的最后一个元素以及倒数第二个元素都是存的整个文档的信息

4) 兄弟节点

- soup.a.next_siblings 获取后面的兄弟节点
- soup.a.previous_siblings 获取前面的兄弟节点
- soup.a.next_sibling 获取下一个兄弟标签
- souo.a.previous_sinbling 获取上一个兄弟标签

find_all

find_all(name,attrs,recursive,text,**kwargs) 可以根据标签名，属性，内容查找文档

1) name

```
html='''
<div class="panel">
    <div class="panel-heading">
        <h4>Hello</h4>
    </div>
    <div class="panel-body">
        <ul class="list" id="list-1">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
            <li class="element">Jay</li>
        </ul>
        <ul class="list list-small" id="list-2">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
        </ul>
    </div>
</div>
'''
from bs4 import BeautifulSoup
import re

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

```
# 传字符串.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('ul'))

# 传正则表达式
print(soup.find_all(re.compile("^l")))

print(type(soup.find_all('ul')[0]))
```

结果返回的是一个列表的方式

```
[<ul class="list" id="list-1">
<li class="element">Foo</li>
<li class="element">Bar</li>
<li class="element">Jay</li>
</ul>, <ul class="list list-small" id="list-2">
<li class="element">Foo</li>
<li class="element">Bar</li>
</ul>]

[<li class="element">Foo</li>, <li class="element">Bar</li>, <li class="element">Jay</li>, <li class="element">Foo</li>, <li class="element">Bar</li>]

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

同时我们是可以针对结果再次 find_all,从而获取所有的 li 标签信息

```
for ul in soup.find_all('ul'):
    print(ul.find_all('li'))
```

2) attrs

```
html='''
<div class="panel">
    <div class="panel-heading">
        <h4>Hello</h4>
    </div>
    <div class="panel-body">
        <ul class="list" id="list-1" name="elements">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
            <li class="element">Jay</li>
        </ul>
        <ul class="list list-small" id="list-2">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
        </ul>
    </div>
</div>
```

```
'''
from bs4 import BeautifulSoup

soup = BeautifulSoup(html, 'lxml')
print(soup.find_all(attrs={'id': 'list-1'}))
print(soup.find_all(attrs={'name': 'elements'}))
```

```
[<ul class="list" id="list-1" name="elements">
<li class="element">Foo</li>
<li class="element">Bar</li>
<li class="element">Jay</li>
</ul>]
[<ul class="list" id="list-1" name="elements">
<li class="element">Foo</li>
<li class="element">Bar</li>
<li class="element">Jay</li>
</ul>]
```

attrs 可以传入字典的方式来查找标签，但是这里有个特殊的就是 class,因为 class 在 python 中是特殊的字段，所以如果想要查找 class 相关的可以更改 attrs={'class_':'element'}或者 soup.find_all('',{"class":"element})，特殊的标签属性可以不写 attrs，例如 id

3) text

```
html='''
<div class="panel">
    <div class="panel-heading">
        <h4>Hello</h4>
    </div>
    <div class="panel-body">
        <ul class="list" id="list-1">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
            <li class="element">Jay</li>
        </ul>
        <ul class="list list-small" id="list-2">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
        </ul>
    </div>
</div>
'''
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.find_all(text='Foo'))
```

结果返回的是查到的所有的 text='Foo'的文本

```
['Foo', 'Foo']
```

其他一些类似的用法：

- `find_parents()`返回所有祖先节点，`find_parent()`返回直接父节点。
- `find_next_siblings()`返回后面所有兄弟节点，`find_next_sibling()`返回后面第一个兄弟节点。
- `find_previous_siblings()`返回前面所有兄弟节点，`find_previous_sibling()`返回前面第一个兄弟节点。
- `find_all_next()`返回节点后所有符合条件的节点, `find_next()`返回第一个符合条件的节点
- `find_all_previous()`返回节点后所有符合条件的节点, `find_previous()`返回第一个符合条件的节点

find

find 的用法与 find*all 一样，区别在于 find 返回 第一个符合匹配结果，find*all 则返回 所有匹配结果的列表。

CSS 选择器

通过 select()直接传入 CSS 选择器就可以完成选择,熟悉前端的人对 CSS 可能更加了解，其实用法也是一样的

- .表示 class, #表示 id 标签 1，标签 2 找到所有的标签 1 和标签 2 标签 1 标签 2 找到标签 1 内部的所有的标签 2
- [attr] 可以通过这种方法找到具有某个属性的所有标签
- [atrr=value] 例子[target=_blank]表示查找所有 target=_blank 的标签
- 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

```
html='''
<div class="panel">
    <div class="panel-heading">
        <h4>Hello</h4>
    </div>
    <div class="panel-body">
        <ul class="list" id="list-1">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
            <li class="element">Jay</li>
        </ul>
```

```
        <ul class="list list-small" id="list-2">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
        </ul>
    </div>
</div>
'''
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')

# 通过类名查找
print(soup.select('.panel .panel-heading'))

# 通过标签名查找
print(soup.select('ul li'))

# 通过 id 名查找
print(soup.select('#list-2'))

# 组合查找,查找 id=list-2 中 class=element 的标签，二者需要用空格分开
print(soup.select('#list-2 .element'))

# 属性查找，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('ul[id="list-2"]'))
```

```
[<div class="panel-heading">
<h4>Hello</h4>
</div>]

[<li class="element">Foo</li>, <li class="element">Bar</li>, <li class=
"element">Jay</li>, <li class="element">Foo</li>, <li class="element">B
ar</li>]

[<ul class="list list-small" id="list-2">
<li class="element">Foo</li>
<li class="element">Bar</li>
</ul>]

[<li class="element">Foo</li>, <li class="element">Bar</li>]

[<ul class="list list-small" id="list-2">
<li class="element">Foo</li>
<li class="element">Bar</li>
</ul>]
```

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，然后通过 get_text()就可以获取文本内容

```
html='''
<div class="panel">
    <div class="panel-heading">
        <h4>Hello</h4>
    </div>
    <div class="panel-body">
        <ul class="list" id="list-1">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
            <li class="element">Jay</li>
        </ul>
        <ul class="list list-small" id="list-2">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
        </ul>
    </div>
</div>
'''
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
for li in soup.select('li'):
    print(li.get_text())
```

```
Foo
Bar
Jay
Foo
Bar
```

总结

- 推荐使用 lxml 解析库，必要时使用 html.parser
- 标签选择筛选功能弱但是速度快
- 建议使用 find()、find_all() 查询匹配单个结果或者多个结果
- 如果对 CSS 选择器熟悉建议使用 select()
- 记住常用的获取属性和文本值的方法

Beautiful Soup 爬虫案例

抓取新浪国内新闻（http://news.sina.com.cn/china/），该列表中的标题名称、时间、链接。

```
from bs4 import BeautifulSoup
import requests

url = 'http://news.sina.com.cn/china/'
web_data = requests.get(url)

# charset=utf-8 表示当前内容的字符集是采用utf-8 编码格式，我们需要用encodin
g 来解锁下，否则的话会很多有乱码
web_data.encoding = 'utf-8'
soup = BeautifulSoup(web_data.text,'lxml')

# 找出所有class 为news-item 的元素，class 名前面需要加点（.）
for news in soup.select('.news-item'):

    # 通过观察代码，可以看到标题被储存在标签h2 中，如果h2 的长度大于0，这里
是为了去除为空的标题数据
    if(len(news.select('h2')) > 0):

        # 将news.select('h2')[0].text 存储在变量h2 中
        h2 = news.select('h2')[0].text

        # time 是class 类型，前面加点来表示，同上，将其数据存储在变量time 中

        time = news.select('.time')[0].text

        # 我们要抓取的链接存放在a 标签中，链接已经不是text 了，后面用href，
将链接数据存储在变量a 中
        a = news.select('a')[0]['href']

        print(h2,time,a)
```

先导入需要的库

```
from bs4 import BeautifulSoup
import requests
from datetime import datetime
import json
import re


news_url = 'http://news.sina.com.cn/c/nd/2017-05-08/doc-ifyeycfp9368908.
shtml'
web_data = requests.get(news_url)
web_data.encoding = 'utf-8'
soup = BeautifulSoup(web_data.text,'lxml')
title = soup.select('#artibodyTitle')[0].text
print(title)
```

```
# 抓取时间，并将原有日期格式转化为标准格式
time = soup.select('.time-source')[0].contents[0].strip()
dt = datetime.strptime(time,'%Y 年%m 月%d 日%H:%M')
print(dt)

source = soup.select('.time-source span span a')[0].text
print(source)

print('\n'.join([p.text.strip() for p in soup.select('#artibody p')[:-1]]))

editor = soup.select('.article-editor')[0].text.lstrip('责任编辑：')
print(editor)

comments = requests.get('http://comment5.news.sina.com.cn/page/info?version=1&format=js&channel=gn&newsid=comos-fyeycfp9368908&group=&compress=0&ie=utf-8&oe=utf-8&page=1&page_size=20')
comments_total = json.loads(comments.text.strip('var data='))
print(comments_total['result']['count']['total'])

news_id = re.search('doc-i(.+).shtml',news_url)
print(news_id.group(1))
```

动态 HTML

JavaScript

JavaScript 是网络上最常用也是支持者最多的客户端脚本语言。它可以收集 用户的跟踪数据,不需要重载页面直接提交表单，在页面嵌入多媒体文件，甚至运行网页游戏。

我们可以在网页源代码的标签里看到，比如：

```
<script type="text/javascript" src="https://statics.huxiu.com/w/mini/static_2015/js/sea.js?v=201601150944"></script>
```

jQuery

jQuery 是一个十分常见的库,70% 最流行的网站(约 200 万)和约 30% 的其他网站(约 2 亿)都在使用。一个网站使用 jQuery 的特征,就是源代码里包含了 jQuery 入口,比如:

```
<script type="text/javascript" src="https://statics.huxiu.com/w/mini/static_2015/js/jquery-1.11.1.min.js?v=201512181512"></script>
```

如果你在一个网站上看到了 jQuery，那么采集这个网站数据的时候要格外小心。jQuery 可 以动态地创建 HTML 内容,只有在 JavaScript 代码执行之后才会显示。如

果你用传统的方 法采集页面内容,就只能获得 JavaScript 代码执行之前页面上的内容。

Ajax

我们与网站服务器通信的唯一方式，就是发出 HTTP 请求获取新页面。如果提交表单之后，或从服务器获取信息之后，网站的页面不需要重新刷新，那么你访问的网站就在用 Ajax 技术。

Ajax 其实并不是一门语言,而是用来完成网络任务(可以认为 它与网络数据采集差不多)的一系列技术。Ajax 全称是 Asynchronous JavaScript and XML(异步 JavaScript 和 XML)，网站不需要使用单独的页面请求就可以和网络服务器进行交互 (收发信息)。

DHTML

Ajax 一样，动态 HTML(Dynamic HTML, DHTML)也是一系列用于解决网络问题的技术集合。DHTML 是用客户端语言改变页面的 HTML 元素(HTML、CSS，或者二者皆 被改变)。比如页面上的按钮只有当用户移动鼠标之后才出现,背景色可能每次点击都会改变，或者用一个 Ajax 请求触发页面加载一段新内容，网页是否属于 DHTML，关键要看有没有用 JavaScript 控制 HTML 和 CSS 元素。

如何解决？

那些使用了 Ajax 或 DHTML 技术改变 / 加载内容的页面，可能有一些采集手段。但是用 Python 解决这个问题只有两种途径:

1. 直接从 JavaScript 代码里采集内容（费时费力）
2. 用 Python 的 第三方库运行 JavaScript，直接采集你在浏览器里看到的页面（简单轻松）。

Selenium+PhantomJS

Selenium

Selenium 是一个 Web 的自动化测试工具，最初是为网站自动化测试而开发的，类型像我们玩游戏用的按键精灵，可以按指定的命令自动操作，不同是 Selenium 可以直接运行在浏览器上，它支持所有主流的浏览器（包括 PhantomJS 这些无界面的浏览器）。

Selenium 可以根据我们的指令，让浏览器自动加载页面，获取需要的数据，甚至页面截屏，或者判断网站上某些动作是否发生。

Selenium 自己不带浏览器，不支持浏览器的功能，它需要与第三方浏览器结合在一起才能使用。但是我们有时候需要让它内嵌在代码中运行，所以我们可以用一个叫 PhantomJS 的工具代替真实的浏览器。

可以从 PyPI 网站下载 Selenium 库 https://pypi.python.org/simple/selenium ，

也可以用 第三方管理器 pip 用命令安装：sudo pip install selenium

Selenium 官方参考文档：http://selenium-python.readthedocs.io/index.html

PhantomJS

PhantomJS 是一个基于 Webkit 的“无界面”(headless)浏览器，它会把网站加载到内存并执行页面上的 JavaScript，因为不会展示图形界面，所以运行起来比完整的浏览器要高效。

如果我们把 Selenium 和 PhantomJS 结合在一起，就可以运行一个非常强大的网络爬虫了，这个爬虫可以处理 JavaScrip、Cookie、headers，以及任何我们真实用户需要做的事情。

PhantomJS 是一个功能完善(虽然无界面)的浏览器而非一个 Python 库，所以它不需要像 Python 的其他库一样安装，但我们可以通过 Selenium 调用 PhantomJS 来直接使用。

PhantomJS 官方参考文档：http://phantomjs.org/documentation

快速入门

Selenium 库里有个叫 WebDriver 的 API。WebDriver 有点儿像可以加载网站的浏览器，但是它也可以像 BeautifulSoup 或者其他 Selector 对象一样用来查找页面元素，与页面上的元素进行交互 (发送文本、点击等)，以及执行其他动作来运行网络爬虫。

```
# 导入 webdriver
from selenium import webdriver

# 调用键盘按键操作时需要引入的 Keys 包
from selenium.webdriver.common.keys import Keys


# 调用环境变量指定的 PhantomJS 浏览器创建浏览器对象
driver = webdriver.PhantomJS()

# 如果没有在环境变量指定 PhantomJS 位置
# driver = webdriver.PhantomJS(executable_path="/usr/local/bin/phantomj
s"))

# get 方法会一直等到页面被完全加载，然后才会继续程序，通常测试会在这里选择 tim
```

```
e.sleep(2)
driver.get("http://www.baidu.com/")

# 获取页面名为 wrapper 的id 标签的文本内容
data = driver.find_element_by_id("wrapper").text

# 打印数据内容
print(data)

# 打印页面标题 "百度一下，你就知道"
print(driver.title)

# 生成当前页面快照并保存
driver.save_screenshot("baidu.png")

# id="kw"是百度搜索输入框，输入字符串"长城"
driver.find_element_by_id("kw").send_keys(u"长城")

# id="su"是百度搜索按钮，click() 是模拟点击
driver.find_element_by_id("su").click()

# 获取新的页面快照
driver.save_screenshot("长城.png")

# 打印网页渲染后的源代码
print(driver.page_source)

# 获取当前页面Cookie
print(driver.get_cookies())


# ctrl+a 全选输入框内容
driver.find_element_by_id("kw").send_keys(Keys.CONTROL,'a')

# ctrl+x 剪切输入框内容
driver.find_element_by_id("kw").send_keys(Keys.CONTROL,'x')

# 输入框重新输入内容
driver.find_element_by_id("kw").send_keys("itcast")

# 模拟Enter 回车键
driver.find_element_by_id("su").send_keys(Keys.RETURN)

# 清除输入框内容
driver.find_element_by_id("kw").clear()

# 生成新的页面快照
```

```
driver.save_screenshot("itcast.png")

# 获取当前 url
print(driver.current_url)

# 关闭当前页面，如果只有一个页面，会关闭浏览器
# driver.close()

# 关闭浏览器
driver.quit()
```

页面操作

Selenium 的 WebDriver 提供了各种方法来寻找元素，假设下面有一个表单输入框：

```
<input type="text" name="user-name" id="passwd-id" />
```

那么：

```
# 获取 id 标签值
element = driver.find_element_by_id("passwd-id")

# 获取 name 标签值
element = driver.find_element_by_name("user-name")

# 获取标签名值
element = driver.find_elements_by_tag_name("input")

# 也可以通过 XPath 来匹配
element = driver.find_element_by_xpath("//input[@id='passwd-id']")
```

定位 UI 元素 (WebElements)

关于元素的选取，有如下的 API 单个元素选取

```
find_element_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
```

By ID

```
<div id="coolestWidgetEvah">...</div>

element = driver.find_element_by_id("coolestWidgetEvah")
---------------------- or ------------------------
```

```
from selenium.webdriver.common.by import By
element = driver.find_element(by=By.ID, value="coolestWidgetEvah")
```

By Class Name

```
<div class="cheese"><span>Cheddar</span></div><div class="cheese"><span>
Gouda</span></div>
```

实现

```
cheeses = driver.find_elements_by_class_name("cheese")
----------------------- or -------------------------
from selenium.webdriver.common.by import By
cheeses = driver.find_elements(By.CLASS_NAME, "cheese")
```

By Tag Name

```
<iframe src="..."></iframe>
```

```
frame = driver.find_element_by_tag_name("iframe")
----------------------- or -------------------------
from selenium.webdriver.common.by import By
frame = driver.find_element(By.TAG_NAME, "iframe")
```

By Name

```
<input name="cheese" type="text"/>
```

```
cheese = driver.find_element_by_name("cheese")
----------------------- or -------------------------
from selenium.webdriver.common.by import By
cheese = driver.find_element(By.NAME, "cheese")
```

By Link Text

```
<a href="http://www.google.com/search?q=cheese">cheese</a>
```

```
cheese = driver.find_element_by_link_text("cheese")
----------------------- or -------------------------
from selenium.webdriver.common.by import By
cheese = driver.find_element(By.LINK_TEXT, "cheese")
```

By Partial Link Text

```
<a href="http://www.google.com/search?q=cheese">search for cheese</a>>
```

```
cheese = driver.find_element_by_partial_link_text("cheese")
----------------------- or -------------------------
from selenium.webdriver.common.by import By
cheese = driver.find_element(By.PARTIAL_LINK_TEXT, "cheese")
```

By CSS

```
<div id="food"><span class="dairy">milk</span><span class="dairy aged">
cheese</span></div>
```

```
cheese = driver.find_element_by_css_selector("#food span.dairy.aged")
----------------------- or -------------------------
from selenium.webdriver.common.by import By
cheese = driver.find_element(By.CSS_SELECTOR, "#food span.dairy.aged")
```

By XPath

```
<input type="text" name="example" />
<INPUT type="text" name="other" />

inputs = driver.find_elements_by_xpath("//input")
----------------------- or -------------------------
from selenium.webdriver.common.by import By
inputs = driver.find_elements(By.XPATH, "//input")
```

鼠标动作链

有些时候，我们需要再页面上模拟一些鼠标操作，比如双击、右击、拖拽甚至按住不动等，我们可以通过导入 ActionChains 类来做到：

示例：

```
#导入 ActionChains 类
from selenium.webdriver import ActionChains

# 鼠标移动到 ac 位置
ac = driver.find_element_by_xpath('element')
ActionChains(driver).move_to_element(ac).perform()


# 在 ac 位置单击
ac = driver.find_element_by_xpath("elementA")
ActionChains(driver).move_to_element(ac).click(ac).perform()

# 在 ac 位置双击
ac = driver.find_element_by_xpath("elementB")
ActionChains(driver).move_to_element(ac).double_click(ac).perform()

# 在 ac 位置右击
ac = driver.find_element_by_xpath("elementC")
ActionChains(driver).move_to_element(ac).context_click(ac).perform()

# 在 ac 位置左键单击hold 住
ac = driver.find_element_by_xpath('elementF')
ActionChains(driver).move_to_element(ac).click_and_hold(ac).perform()

# 将 ac1 拖拽到 ac2 位置
ac1 = driver.find_element_by_xpath('elementD')
```

```
ac2 = driver.find_element_by_xpath('elementE')
ActionChains(driver).drag_and_drop(ac1, ac2).perform()
```

弹窗处理

当你触发了某个事件之后，页面出现了弹窗提示，处理这个提示或者获取提示信息方法如下：

```
alert = driver.switch_to_alert()
```

页面切换

一个浏览器肯定会有很多窗口，所以我们肯定要有方法来实现窗口的切换。切换窗口的方法如下：

```
driver.switch_to.window("this is window name")
```

也可以使用 window_handles 方法来获取每个窗口的操作对象。例如：

```
for handle in driver.window_handles:
    driver.switch_to_window(handle)
```

页面前进和后退

操作页面的前进和后退功能：

```
driver.forward()      #前进
driver.back()         # 后退
```

Cookies

获取页面每个 Cookies 值，用法如下

```
for cookie in driver.get_cookies():
    print "%s=%s;" % (cookie['name'], cookie['value'])
```

删除 Cookies，用法如下

```
# By name
driver.delete_cookie("BAIDUID")

# all
driver.delete_all_cookies()
```

页面等待

现在的网页越来越多采用了 Ajax 技术，这样程序便不能确定何时某个元素完全加载出来了。如果实际页面等待时间过长导致某个 dom 元素还没出来，但是你的代码直接使用了这个 WebElement，那么就会抛出 NullPointer 的异常。

为了避免这种元素定位困难而且会提高产生 ElementNotVisibleException 的概率。所以 Selenium 提供了两种等待方式，一种是隐式等待，一种是显式等待。

隐式等待是等待特定的时间，显式等待是指定某一条件直到这个条件成立时继续执行。

显式等待

显式等待指定某个条件，然后设置最长等待时间。如果在这个时间还没有找到元素，那么便会抛出异常了。

```
from selenium import webdriver
from selenium.webdriver.common.by import By
# WebDriverWait 库，负责循环等待
from selenium.webdriver.support.ui import WebDriverWait
# expected_conditions 类，负责条件出发
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.PhantomJS()
driver.get("http://www.xxxxx.com/loading")
try:
    # 每隔10秒查找页面元素 id="myDynamicElement"，直到出现则返回
    element = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "myDynamicElement"))
    )
finally:
    driver.quit()
```

如果不写参数，程序默认会 0.5s 调用一次来查看元素是否已经生成，如果本来元素就是存在的，那么会立即返回。

下面是一些内置的等待条件，你可以直接调用这些条件，而不用自己写某些等待条件了。

```
title_is
title_contains
presence_of_element_located
visibility_of_element_located
visibility_of
presence_of_all_elements_located
text_to_be_present_in_element
text_to_be_present_in_element_value
frame_to_be_available_and_switch_to_it
invisibility_of_element_located
element_to_be_clickable - it is Displayed and Enabled.
staleness_of
element_to_be_selected
element_located_to_be_selected
element_selection_state_to_be
```

```
element_located_selection_state_to_be
alert_is_present
```

隐式等待

隐式等待比较简单，就是简单地设置一个等待时间，单位为秒。

```
from selenium import webdriver

driver = webdriver.PhantomJS()
driver.implicitly_wait(10) # seconds
driver.get("http://www.xxxxx.com/loading")
myDynamicElement = driver.find_element_by_id("myDynamicElement")
```

当然如果不设置，默认等待时间为 0。

网站模拟登录

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

driver = webdriver.PhantomJS()
driver.get("http://www.douban.com")

# 输入账号密码
driver.find_element_by_name("form_email").send_keys("xxxxx@xxxx.com")
driver.find_element_by_name("form_password").send_keys("xxxxxxxx")

# 模拟点击登录
driver.find_element_by_xpath("//input[@class='bn-submit']").click()

# 等待 3 秒
time.sleep(3)

# 生成登陆后快照
driver.save_screenshot("douban.png")

# 保存源码
with open("douban.html", "w") as file:
    file.write(driver.page_source)

driver.quit()
```

动态页面模拟点击

爬取斗鱼直播平台的所有房间信息：

```
import unittest
from selenium import webdriver
from bs4 import BeautifulSoup


class douyuSelenium(unittest.TestCase):
    # 初始化方法
    def setUp(self):
        self.driver = webdriver.PhantomJS()

    #具体的测试用例方法，一定要以 test 开头
    def testDouyu(self):
        self.driver.get('http://www.douyu.com/directory/all')
        while True:
            # 指定 xml 解析
            soup = BeautifulSoup(driver.page_source, 'xml')
            # 返回当前页面所有房间标题列表 和 观众人数列表
            titles = soup.find_all('h3', {'class': 'ellipsis'})
            nums = soup.find_all('span', {'class': 'dy-num fr'})

            # 使用 zip()函数来可以把列表合并，并创建一个元组对的列表[(1,2),
(3,4)]
            for title, num in zip(nums, titles):
                print u"观众人数:" + num.get_text().strip(), u"\t 房间标
题: " + title.get_text().strip()
            # 指定元素找到则返回 非-1，表示到达最后一页，退出循环
            if driver.page_source.find('shark-pager-disable-next') != -
1:
                break
            # 模拟下一页点击
            self.driver.find_element_by_class_name('shark-pager-next').
click()

    # 退出时的清理方法
    def tearDown(self):
        print '加载完成...'
        self.driver.quit()

if __name__ == "__main__":
    unittest.main()
```

执行 JavaScript 语句

- 隐藏百度图片

```
from selenium import webdriver

driver = webdriver.PhantomJS()
driver.get("https://www.baidu.com/")
```

```
# 给搜索输入框标红的javascript 脚本
js = "var q=document.getElementById(\"kw\");q.style.border=\"2px solid
red\";"

# 调用给搜索输入框标红js 脚本
driver.execute_script(js)

#查看页面快照
driver.save_screenshot("redbaidu.png")

#js 隐藏元素，将获取的图片元素隐藏
img = driver.find_element_by_xpath("//*[@id='lg']/img")
driver.execute_script('$(arguments[0]).fadeOut()',img)

# 向下滚动到页面底部
driver.execute_script("$('.scroll_top').click(function(){$('html,body').
animate({scrollTop: '0px'}, 800);});")

#查看页面快照
driver.save_screenshot("nullbaidu.png")

driver.quit()
```

- 模拟滚动条滚动到底部

```
from selenium import webdriver
import time

driver = webdriver.PhantomJS()
driver.get("https://movie.douban.com/typerank?type_name=剧情&type=11&in
terval_id=100:90&action=")

# 向下滚动10000 像素
js = "document.body.scrollTop=10000"
#js="var q=document.documentElement.scrollTop=10000"
time.sleep(3)

#查看页面快照
driver.save_screenshot("douban.png")

# 执行JS 语句
driver.execute_script(js)
time.sleep(10)

#查看页面快照
driver.save_screenshot("newdouban.png")
```

```
driver.quit()
```

selenium 案例

```
import pymongo
from selenium import webdriver
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.wait import WebDriverWait
from pyquery import PyQuery as pq
from config import *
from urllib.parse import quote

# browser = webdriver.Chrome()
# browser = webdriver.PhantomJS(service_args=SERVICE_ARGS)

chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--headless')
browser = webdriver.Chrome(chrome_options=chrome_options)

wait = WebDriverWait(browser, 10)
client = pymongo.MongoClient(MONGO_URL)
db = client[MONGO_DB]


def index_page(page):
    """
    抓取索引页
    :param page: 页码
    """
    print('正在爬取第', page, '页')
    try:
        url = 'https://s.taobao.com/search?q=' + quote(KEYWORD)
        browser.get(url)
        if page > 1:
            input = wait.until(
                EC.presence_of_element_located((By.CSS_SELECTOR, '#main
srp-pager div.form > input')))
            submit = wait.until(
                EC.element_to_be_clickable((By.CSS_SELECTOR, '#mainsrp-
pager div.form > span.btn.J_Submit')))
            input.clear()
            input.send_keys(page)
            submit.click()
        wait.until(
            EC.text_to_be_present_in_element((By.CSS_SELECTOR, '#mainsr
p-pager li.item.active > span'), str(page)))
        wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, '.m
```

```
-itemlist .items .item')))
        get_products()
    except TimeoutException:
        index_page(page)


def get_products():
    """
    提取商品数据
    """
    html = browser.page_source
    doc = pq(html)
    items = doc('#mainsrp-itemlist .items .item').items()
    for item in items:
        product = {
            'image': item.find('.pic .img').attr('data-src'),
            'price': item.find('.price').text(),
            'deal': item.find('.deal-cnt').text(),
            'title': item.find('.title').text(),
            'shop': item.find('.shop').text(),
            'location': item.find('.location').text()
        }
        print(product)
        save_to_mongo(product)


def save_to_mongo(result):
    """
    保存至MongoDB
    :param result: 结果
    """
    MONGO_URL = 'localhost'
    MONGO_DB = 'taobao'
    MONGO_COLLECTION = 'products'
    KEYWORD = 'ipad'
    MAX_PAGE = 100
    SERVICE_ARGS = ['--load-images=false', '--disk-cache=true']
    try:
        if db[MONGO_COLLECTION].insert(result):
            print('存储到 MongoDB 成功')
    except Exception:
        print('存储到 MongoDB 失败')


def main():
    """
    遍历每一页
    """
    for i in range(1, MAX_PAGE + 1):
```

```
        index_page(i)
    browser.close()


if __name__ == '__main__':
    main()
```