

# Selenium 与 PhantomJS

## 动态 HTML 介绍

### 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。

### 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:

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。

## 解决办法

用 Python 解决这个问题只有两种途径:

- 直接从 JavaScript 代码里采集内容 (费时费力)
- 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。

## 动态页面爬取环境配置

### Selenium 简介

Selenium 是一个 Web 的自动化测试工具, 类型像我们玩游戏用的按键精灵, 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

Selenium 库里有个叫 WebDriver 的 API。WebDriver 可以加载网站也可以查找页面元素，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。

**安装方式一：**PyPI 网站下载安装 <https://pypi.python.org/simple/selenium>

**安装方式二：**pip install selenium

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

## PhantomJS 简介

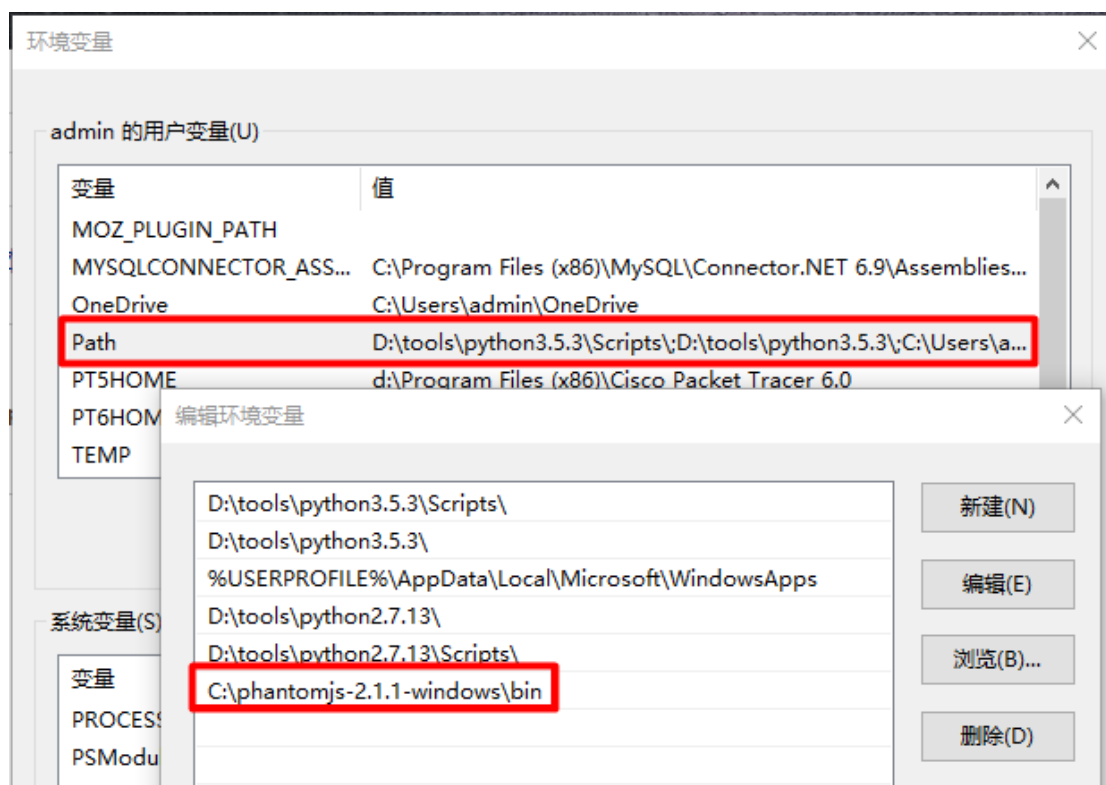
[PhantomJS](#) 是一个基于 Webkit 的“无界面” (headless)浏览器非 Python 库，它会把网站加载到内存并执行页面上的 JavaScript，不会展示图形界面。

把 Selenium 和 PhantomJS 结合在一起,通过 Selenium 调用 PhantomJS 来直接使用,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 JavaScript、Cookie、headers, 以及任何我们真实用户需要做的事情。

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

**官网下载安装** <http://phantomjs.org/download.html>

- 1、解压放到：C:\phantomjs-2.1.1-windows
- 2、需设置环境变量，Path 添加 C:\phantomjs-2.1.1-windows\bin



3、win+R, 输入 cmd 打开控制台, 输入 phantomjs -v, 若输出了版本号, 则证明安装成功

## Chrome driver 的安装

爬虫开发过程中使用 selenium + webdriver 打开 chrome,

### chrome driver 的安装

以 chrome 版本 65.0.3325.181

1、打开如下页面:

<https://sites.google.com/a/chromium.org/chromedriver/downloads>

对照下载说明, 找到对应的 ChromeDriver 版本 2.38

2、打开如下网页:

<http://chromedriver.storage.googleapis.com/index.html>

选取 2.38 文件夹, 下载对应文件

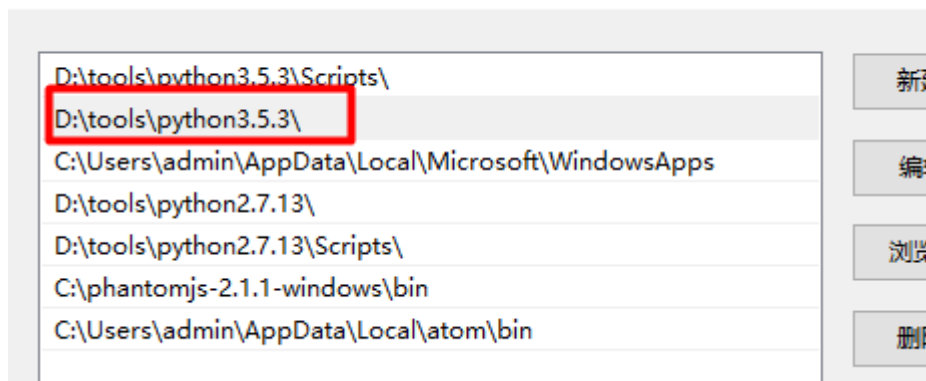
3、把 exe 文件放置到如下位置之一:

- (1) chrome 的安装目录下 (eg: C:\Program Files (x86)\Google\Chrome\Application)
- (2) Python 的安装目录下 (eg: D:\software\Python35)

|                     |                  |        |          |
|---------------------|------------------|--------|----------|
| include             | 2017/6/13 17:10  | 文件夹    |          |
| Lib                 | 2017/6/13 17:10  | 文件夹    |          |
| libs                | 2017/6/13 17:10  | 文件夹    |          |
| Scripts             | 2018/6/12 7:30   | 文件夹    |          |
| share               | 2017/12/21 11:14 | 文件夹    |          |
| tcl                 | 2017/6/13 17:10  | 文件夹    |          |
| Tools               | 2017/6/13 17:10  | 文件夹    |          |
| chromedriver.exe    | 2018/4/20 17:01  | 应用程序   | 6,256 KB |
| LICENSE.txt         | 2017/1/16 16:06  | 文本文档   | 30 KB    |
| NEWS.txt            | 2017/1/16 15:37  | 文本文档   | 359 KB   |
| python.exe          | 2017/1/16 16:05  | 应用程序   | 42 KB    |
| python3.dll         | 2017/1/16 16:03  | 应用程序扩展 | 57 KB    |
| python35.dll        | 2017/1/16 16:02  | 应用程序扩展 | 3,851 KB |
| pythoncom35.dll     | 2016/1/12 6:19   | 应用程序扩展 | 540 KB   |
| pythonw.exe         | 2017/1/16 16:05  | 应用程序   | 42 KB    |
| pywin32-wininst.log | 2018/6/12 7:30   | 文本文档   | 135 KB   |

4、Path 进行编辑，在变量值后面加入 chrome 或 Python 的安装目录

编辑环境变量



5、用 Chrome 浏览器来测试

```
from selenium import webdriver
browser = webdriver.Chrome()
browser.get('http://www.baidu.com/')
```

## 动态页面的请求

## 网页抓取

```
# 导入 webdriver
```

```
from selenium import webdriver

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

# 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
time.sleep(2)
driver.get("http://www.baidu.com/")

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

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

# 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()
```

## 说明：

- 1、要使用 selenium 首先导入 webdriver
- 2、生成浏览器对象 `driver = webdriver.PhantomJS()`
- 3、`driver.get(url)` 方法：加载指定页面，阻塞方式直到加载完成
- 4、`driver.find_element_by_id(elemid)` 方法：获取页面指定 id 的标签
- 5、`driver.title`：页面标题
- 6、`driver.page_source`：页面源码
- 7、`driver.get_cookies()`：获取页面 cookies
- 8、`driver.save_screenshot(filename)`：保存当前页面快照

- 9、`driver.find_element_by_id("kw").send_keys("奇酷信息")`：指定元素设置 value，通常用于 input field
- 10、`driver.find_element_by_id("su").click()` 模拟点击，通常用于按钮
- 11、`driver.find_element_by_id("kw").clear()`：清除输入框内容
- 12、`driver.quit()`：关闭浏览器

## 数据解析提取

## 定位 UI 元素 (WebElements)

### 单个元素查找

```
find_element_by_name
find_element_by_id
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
```

### 多个元素查找

```
find_elements_by_name
find_elements_by_id
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
```

按 id, name, xpath, link\_text, partial\_link\_text, tag\_name, class\_name, css\_selector

定位 UI 元素

## By ID

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

### 实现

```
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")
```

## By Tag Name

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

### 实现

```
frame = driver.find_element_by_tag_name("iframe")
```

## By Name

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

### 实现

```
cheese = driver.find_element_by_name("cheese")
```

## By Link Text

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

### 实现



```
cheese = driver.find_element_by_link_text("cheese")
```

### By Partial Link Text (部分链接文本)

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

#### 实现

```
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>
```

#### 实现

```
cheese = driver.find_element_by_css_selector("#food span.dairy.aged")
```

### By XPath

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

#### 实现

```
inputs = driver.find_elements_by_xpath("//input")
```

## 获取元素属性

### get\_attribute('class')

```
from selenium import webdriver
```

```
browser = webdriver.Chrome()
url = 'https://www.zhihu.com/explore'
browser.get(url)
logo = browser.find_element_by_id('zh-top-link-logo')
print(logo)
print(logo.get_attribute('class'))
```

## 获取文本值

可见元素文本：**text**

Selenium WebDriver 只会与可见元素交互，所以获取隐藏元素的文本总是会返回空字符串。

要获取隐藏元素的文本。这些内容可以使用

**element.get\_attribute('innerHTML')**，会返回元素的内部 HTML，包含所有的 HTML 标签。

**element.get\_attribute('textContent')**，只会得到文本内容，而不会包含 HTML 标签。是

W3C 兼容的文字内容属性，但是 IE 不支持

**element.get\_attribute('innerText')**，只会得到文本内容，而不会包含 HTML 标签。不是

W3C DOM 的指定内容，Firefox 不支持

```
from selenium import webdriver
```

```
browser = webdriver.Chrome()
```

```
url = 'https://www.zhihu.com/explore'
```

```
browser.get(url)
```

```
input = browser.find_element_by_class_name('zu-top-add-question')
```

```
print(input.text)
```

## 获取 ID，位置，标签名

id

location

tag\_name

size

```
from selenium import webdriver
```

```
browser = webdriver.Chrome()
```

```
url = 'https://www.zhihu.com/explore'
```

```
browser.get(url)
```

```
input = browser.find_element_by_class_name('zu-top-add-question')
```

```
print(input.id)
print(input.location)
print(input.tag_name)
print(input.size)
```

## 执行 JavaScript

这是一个非常有用的方法，这里就可以直接调用 js 方法来实现一些操作，

下面的例子是通过登录知乎然后通过 js 翻到页面底部，并弹框提示

```
from selenium import webdriver
browser = webdriver.Chrome()
browser.get("http://www.zhihu.com/explore")
print(browser.page_source)
browser.execute_script('window.scrollTo(0, document.body.scrollHeight)')
browser.execute_script('alert("To Bottom")')
```

## Cookies

```
get_cookies()
delete_all_cookies()
add_cookie()
```

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:

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

```
# all
driver.delete_all_cookies()
```

## 范例-获取 cookie

```
from selenium import webdriver
driver = webdriver.Chrome()
# driver.get("http://member.rltxttest.xyz/login/login.html")
driver.get("http://www.youdao.com")

# 获取 cookie 信息
cookies = driver.get_cookies()
# 打印获取的 cookies 信息
print(cookies)
driver.quit()
```

## 范例-添加 cookie

```
from selenium import webdriver

driver = webdriver.Chrome()
# driver.get("http://member.rltxttest.xyz/login/login.html")
driver.get("http://www.youdao.com")
# 向 cookie 中 name 和 value 中添加回话信息,
driver.add_cookie({'name': 'key-aaaaaaa', 'value': 'value-bbbbbb'})
# 遍历 cookie 中 name 和 value 信息并打印对应的信息，并包括添加对应的信息
for cookie in driver.get_cookies():
    print("%s->%s" % (cookie['name'], cookie['value']))
driver.quit()
```

## 截屏

```
driver = webdriver.Chrome()
driver.maximize_window()
driver.get("https://blog.csdn.net/Kwoky/article/details/80285201")
driver.save_screenshot("./images/app2.png")
```

## 超长截屏

```
driver = webdriver.PhantomJS()
driver.maximize_window()
```

```
driver.get("https://blog.csdn.net/Kwoky/article/details/80285201")
driver.save_screenshot("./images/app1.png")
```

## 自动化交互

### 鼠标动作链

在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，可以通过导入 ActionChains 类实现

### ActionChains 执行原理

当调用 ActionChains 的方法时，不会立即执行，而是会将所有的操作按顺序存放在一个队列里，当你调用 perform() 方法时，队列中的时间会依次执行。

有两种写法本质是一样的，ActionChains 都会按照顺序执行所有的操作。

#### 链式写法

```
menu = driver.find_element_by_css_selector(".nav")
hidden_submenu = driver.find_element_by_css_selector(".nav #submenu1")
ActionChains(driver).move_to_element(menu).click(hidden_submenu).perform()
```

#### 分步写法

```
menu = driver.find_element_by_css_selector(".nav")
hidden_submenu = driver.find_element_by_css_selector(".nav #submenu1")

actions = ActionChains(driver)
actions.move_to_element(menu)
actions.click(hidden_submenu)
actions.perform()
```

### ActionChains 方法列表

```
click(on_element=None) ——单击鼠标左键
click_and_hold(on_element=None) ——点击鼠标左键，不松开
context_click(on_element=None) ——点击鼠标右键
```

**double\_click**(*on\_element=None*) —— 双击鼠标左键  
**drag\_and\_drop**(*source, target*) —— 拖拽到某个元素然后松开  
**drag\_and\_drop\_by\_offset**(*source, xoffset, yoffset*) —— 拖拽到某个坐标然后松开  
**key\_down**(*value, element=None*) —— 按下某个键盘上的键  
**key\_up**(*value, element=None*) —— 松开某个键  
**move\_by\_offset**(*xoffset, yoffset*) —— 鼠标从当前位置移动到某个坐标  
**move\_to\_element**(*to\_element*) —— 鼠标移动到某个元素  
**move\_to\_element\_with\_offset**(*to\_element, xoffset, yoffset*) —— 移动到距某个元素（左上角坐标）多少距离的位置  
**perform**() —— 执行链中的所有动作  
**release**(*on\_element=None*) —— 在某个元素位置松开鼠标左键  
**send\_keys**(*\*keys\_to\_send*) —— 发送某个键到当前焦点的元素  
**send\_keys\_to\_element**(*element, \*keys\_to\_send*) —— 发送某个键到指定元素

## 代码示例

### 1. 模拟点击

```

from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from time import sleep

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.maximize_window()
driver.get('http://sahitest.com/demo/clicks.htm')

click_btn = driver.find_element_by_xpath('//input[@value="click me"]') # 单击按钮
doubleclick_btn = driver.find_element_by_xpath('//input[@value="dbl click me"]') # 双击按钮
rightclick_btn = driver.find_element_by_xpath('//input[@value="right click me"]') # 右键单击按钮

ActionChains(driver).click(click_btn).double_click(doubleclick_btn).context_click(rightclick_btn).perform() # 链式用法

print(driver.find_element_by_name('t2').get_attribute('value'))

sleep(2)
driver.quit()
  
```

结果：

```
[CLICK][DOUBLE_CLICK][RIGHT_CLICK]
```

## 2.鼠标移动

```
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from time import sleep

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.maximize_window()
driver.get('http://sahitest.com/demo/mouseover.htm')

write = driver.find_element_by_xpath('//input[@value="Write on hover"]') # 鼠标移动到此元素，
在下面的 input 框中会显示 “Mouse moved”
blank = driver.find_element_by_xpath('//input[@value="Blank on hover"]') # 鼠标移动到此元素，
会清空下面 input 框中的内容

result = driver.find_element_by_name('t1')

action = ActionChains(driver)
action.move_to_element(write).perform() # 移动到 write，显示 “Mouse moved”
print(result.get_attribute('value'))

# action.move_to_element(blank).perform()
action.move_by_offset(10, 50).perform() # 移动到距离当前位置(10,50)的点，与上句效果相同，
移动到 blank 上，清空
print(result.get_attribute('value'))

action.move_to_element_with_offset(blank, 10, -40).perform() # 移动到距离 blank 元素(10,-40)的
点，可移动到 write 上
print(result.get_attribute('value'))

sleep(2)
driver.quit()
```

## 结果

Mouse moved

Mouse moved

### 3.拖拽

```
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from time import sleep

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.maximize_window()
driver.get('http://sahitest.com/demo/dragDropMooTools.htm')

dragger = driver.find_element_by_id('dragger') # 被拖拽元素
item1 = driver.find_element_by_xpath('//div[text()='Item 1']') # 目标元素 1
item2 = driver.find_element_by_xpath('//div[text()='Item 2']') # 目标 2
item3 = driver.find_element_by_xpath('//div[text()='Item 3']') # 目标 3
item4 = driver.find_element_by_xpath('//div[text()='Item 4']') # 目标 4

action = ActionChains(driver)
action.drag_and_drop(dragger, item1).perform() # 1.移动 dragger 到目标 1
sleep(2)
action.click_and_hold(dragger).release(item2).perform() # 2.效果与上句相同，也能起到移动效果
sleep(2)
action.click_and_hold(dragger).move_to_element(item3).release().perform() # 3.效果与上两句相同，也能起到移动的效果
sleep(2)
# action.drag_and_drop_by_offset(dragger, 400, 150).perform() # 4.移动到指定坐标
action.click_and_hold(dragger).move_by_offset(400, 150).release().perform() # 5.与上一句相同，移动到指定坐标
sleep(2)
driver.quit()
```

一般用坐标定位很少，用上例中的方法 1 足够了，如果看源码，会发现方法 2 其实就是方法 1 中的 drag\_and\_drop()的实现。**注意：拖拽使用时注意加等待时间，有时会因为速度太快而失败。**



## 4.按键

模拟按键有多种方法，能用 win32api 来实现，能用 SendKeys 来实现，也可以用 selenium 的 WebElement 对象的 send\_keys()方法来实现，这里 ActionChains 类也提供了几个模拟按键的方法。

```
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
# 要想调用键盘按键操作需要引入 keys 包
from selenium.webdriver.common.keys import Keys
from time import sleep

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.maximize_window()
driver.get('http://sahitest.com/demo/keypress.htm')

key_up_radio = driver.find_element_by_id('r1') # 监测按键升起
key_down_radio = driver.find_element_by_id('r2') # 监测按键按下
key_press_radio = driver.find_element_by_id('r3') # 监测按键按下升起

enter = driver.find_elements_by_xpath('//*[@name="f1"]/input')[1] # 输入框
result = driver.find_elements_by_xpath('//*[@name="f1"]/input')[0] # 监测结果

# 监测 key_down
key_down_radio.click()
ActionChains(driver).key_down(Keys.CONTROL, enter).key_up(Keys.CONTROL).perform()
print(result.get_attribute('value'))

# 监测 key_up
key_up_radio.click()
enter.click()
ActionChains(driver).key_down(Keys.SHIFT).key_up(Keys.SHIFT).perform()
print(result.get_attribute('value'))

# 监测 key_press
key_press_radio.click()
enter.click()
ActionChains(driver).send_keys('a').perform()
```

```
print(result.get_attribute('value'))
driver.quit()
```

## 结果:

```
key downed charCode=[0] keyCode=[17] CTRL
key upped charCode=[0] keyCode=[16] NONE
key pressed charCode=[97] keyCode=[0] NONE
```

## 案例：百度自动化搜索

```
# 导入 webdriver
from selenium import webdriver

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

# 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
time.sleep(2)
driver.get("http://www.baidu.com/")

# 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("奇酷信息")

# 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("奇酷信息.png")

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

# 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()
```

## html 元素操作

### 窗口最大化

```
driver.maximize_window()
```

### 下拉框的处理

有时候我们会碰到<select> </select>标签的下拉框。直接点击下拉框中的选项不一定可行。

Selenium 专门提供了 Select 类来处理下拉框完成这些事情

- Select 提供了三种选择方法：

```
select_by_index(index) ——通过选项的顺序，第一个为 0  
select_by_value(value) ——通过 value 属性  
select_by_visible_text(text) ——通过选项可见文本
```

- 全部取消选择:

```
select.deselect_all()
```

- Select 提供了相应属性，选择了选项之后，查看所选项

```
options ——提供所有的选项的列表，其中都是选项的 WebElement 元素  
all_selected_options ——提供所有被选中的选项的列表，其中也均为选项的 WebElement 元素  
first_selected_option ——提供第一个被选中的选项，也是下拉框的默认值
```

### 范例

```
from selenium import webdriver  
from selenium.webdriver.support.ui import Select  
  
driver = webdriver.Chrome()  
driver.get('http://sahitest.com/demo/selectTest.htm')  
  
s1 = Select(driver.find_element_by_id('s1Id')) # 实例化 Select  
  
s1.select_by_index(1) # 选择第二项选项: o1  
print(s1.first_selected_option.text)  
s1.select_by_value("o2") # 选择 value="o2"的项
```

```
print(s1.first_selected_option.text)
s1.select_by_visible_text("o3") # 选择 text="o3"的值，即在下拉时我们可以看到的文本
print(s1.first_selected_option.text)
driver.quit()
```

## 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()
```

用 `switch_to_alert` 先定位到弹窗，然后使用一系列方法来操作：

`accept` - 点击【确认】按钮

`dismiss` - 点击【取消】按钮（如有按钮）

`send_keys` - 输入内容（如有输入框）

```
from selenium import webdriver
import time
```

```
driver = webdriver.Chrome()
driver.get("http://sahitest.com/demo/alertTest.htm")
driver.find_element_by_name("b1").click()
time.sleep(1)
al = driver.switch_to_alert()
time.sleep(1)
al.accept()
```

`switch_to_alert()`是旧写法，新写法应该是 `switch_to.alert()`，但是会报错，猜测是版本问题。

## 页面切换

一个浏览器肯定会有很多窗口，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

from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.implicitly_wait(3) # seconds
first_url = 'http://www.baidu.com/'
driver.get(first_url)
print(driver.title) # 第一个页面
driver.find_element_by_link_text("新闻").click()
driver.switch_to_window(driver.window_handles[0])
print(driver.title) # 第一个页面
#browser.switch_to_window(browser.window_handles[1])
#browser.title # 最后一个页面
time.sleep(2)
driver.back() # 从百度新闻后退到百度首页
print(driver.title) # 第一个页面
time.sleep(2)
driver.forward() # 百度首页前进到百度新闻
print(driver.title) # 第一个页面
time.sleep(2)
driver.quit()
```

## 运行结果

百度一下，你就知道  
百度新闻——全球最大的中文新闻平台  
百度一下，你就知道  
百度新闻——全球最大的中文新闻平台

## 选项卡管理

通过执行 js 命令实现新开选项卡 window.open()

不同的选项卡是存在列表里 browser.window\_handles

通过 browser.window\_handles[0]就可以操作第一个选项卡

```
import time
from selenium import webdriver

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])
browser.get('https://www.taobao.com')
time.sleep(1)
browser.switch_to_window(browser.window_handles[0])
browser.get('https://python.org')
```

## iframe 定位

```
from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.maximize_window()
# 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 time.sleep(2)
driver.get("http://sahitest.com/demo/iframesTest.htm")
time.sleep(2)
print(driver.current_url)
print(driver.page_source)
frames = driver.find_elements_by_tag_name("iframe")
driver.switch_to_frame(frames[0])
print('=====')
print(driver.page_source)
```

## 页面等待

注意：这是非常重要的一部分!!

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 NullPointerException 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, 负责条件触发
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Chrome()
driver.get("http://www.xxxxx.com/loading")
try:
    # 页面一直循环, 直到 id="myDynamicElement" 出现
    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 标题是某内容
title_contains 标题包含某内容
```

presence\_of\_element\_located 元素加载出，传入定位元组，如 (By.ID, 'p')

visibility\_of\_element\_located 元素可见，传入定位元组

visibility\_of 可见，传入元素对象

presence\_of\_all\_elements\_located 所有元素加载出

text\_to\_be\_present\_in\_element 某个元素文本包含某文字

text\_to\_be\_present\_in\_element\_value 某个元素值包含某文字

frame\_to\_be\_available\_and\_switch\_to\_it frame 加载并切换

invisibility\_of\_element\_located 元素不可见

element\_to\_be\_clickable 元素可点击

staleness\_of 判断一个元素是否仍在 DOM，可判断页面是否已经刷新

element\_to\_be\_selected 元素可选择，传元素对象

element\_located\_to\_be\_selected 元素可选择，传入定位元组

element\_selection\_state\_to\_be 传入元素对象以及状态，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 Alert

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

当然如果不设置，默认等待时间为 0。

## 其它设置

## 无界面浏览器

```
opt = webdriver.ChromeOptions() # 创建 chrome 参数对象
opt.set_headless() # 把 chrome 设置成无头模式，不论 windows 还是 linux 都可以，自动适配对应参数
driver = webdriver.Chrome(options=opt) # 不制定 options 选项则是普通有头浏览器
```



## 代理 ip 的使用

```
ip = '123.157.67.30:34942'
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--proxy-server=http://' + ip)

url = 'http://www.gsxt.gov.cn/index.html'
driver = webdriver.Chrome(chrome_options=chrome_options)
driver.get(url)
time.sleep(2)
print(driver.page_source)
driver.close()
```