webdriver 实用指南 python 版本

启动浏览器 8
场景 8
代码 8
关闭浏览器 8
场景 8
代码 9
浏览器最大化 9
场景 9
代码 10
设置浏览器大小 10
场景 10
代码 11
访问链接 12
情景 12
代码 12
打印当前页面的 title 及 url 12
情景 12
代码 13

前进和后退 14
场景 14
代码 14
简单的对象定位 15
场景 15
代码 16
html 代码 form.html 16
python 代码 simple_locate.rb 18
讨论 20
定位一组对象 20
场景 20
代码 21
checkbox.html 21
find_element.rb 22
讨论 23
层级定位 24
场景 24
代码 24
level_locate.html 24
level_locate.py 26
讨论 27
操作测试对象 27

场景 28
代码 28
operate_element.html 28
operate_element.py 30
send keys 模拟按键输入 31
场景 31
代码 31
send_keys.html 32
send_keys.py 33
处理 button group 34
场景 34
代码 34
button_group.html 34
button_group.py 36
讨论 37
处理 button dropdown 37
场景 37
代码 37
button_dropdown.html 37
button_dropdown.py 39
处理 navs 40
场景 40

代码 40
navs.html 40
navs.py 41
处理面包屑 42
场景 42
代码 43
breadcrumb.html 43
breadcrumb.rb 44
处理对话框 45
场景 45
代码 45
modal.html 46
modal.py 47
获取测试对象的属性及内容 49
场景 49
代码 50
attribute.html 50
attribute.py 51
获取测试对象的 css 属性 52
场景 52
代码 52
css.html 52

css.py 53
获取测试对象的状态 54
场景 54
代码 55
status.html 55
status.py 56
form 的操作 57
场景 57
代码 58
form.html 58
form.rb 60
执行 js 61
场景 61
代码 62
js.html 62
js.python 63
处理 alert/confirm/prompt 64
场景 64
代码 64
alert.html 64
alert.py 66
wait 66

场景 67
代码 67
wait.html 68
wait.py 69
定位 frame 中的元素 70
场景 70
代码 71
frame.html 71
inner.html 72
frame.py 73
讨论 74
action 74
场景 75
代码 75
讨论 76
上传文件 76
场景 76
代码 76
upload_file.html 76
upload_file.py 77
下载 78
场景 78

代码 78
超时设置 79
场景 79
代码 79
Remote Webdriver 79
安装 80
启动 driver 80
使用 watir-webdriver 启动 driver 81
java 版本 81
python 版本 82
cookie 82
场景 82
代码 83
cookie.py 83

启动浏览器

场景

在使用 webdriver 进行测试时启动浏览器无疑是必须的前置工作。

代码

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
public class StartBrowser {
    public static void main(String[] args) {
        WebDriver dr = new ChromeDriver();
    }
}
```

关闭浏览器

场景

在脚本运行完毕或者测试代码结束的时候关闭浏览器是非常自然的事情，就像在吃完饭后就把餐桌收拾干净一样。

关闭浏览器有两种方式：

close 方法

quit 方法

close 方法关闭当前的浏览器窗口，quit 方法不仅关闭窗口，还会彻底的退出 webdriver，释放与 driver server 之间的连接。所以简单来说 quit 是更加彻底的 close，quit 会更好的释放资源，适合强迫症和完美主义者。

代码

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;


public class CloseBrowser {
    public static void main(String[] args) {
        WebDriver dr = new ChromeDriver();
        System.out.println("browser will be closed");
        dr.quit();
        System.out.println("browser is closed");
    }
}
```

浏览器最大化

场景

当我们在测试中使用一些基于图像和坐标的辅助测试工具时，我们就会需要使浏览器在每次测试时保存最大化，以便在同一分辨率下进行图像比对和坐标点选。

举例来说，如果在 webdriver 测试中使用了 sikuli 来对 flash 插件进行操作的话，把浏览器最大化无疑是一个比较简单的保证分辨率统一的解决方案。

代码

```
   import org.openqa.selenium.WebDriver;
    import org.openqa.selenium.chrome.ChromeDriver;
    public class Maximize {
        public static void main(String[] args) throws
InterruptedException {
            WebDriver dr = new ChromeDriver();
            Thread.sleep(2000);
            System.out.println("maximize browser");
            dr.manage().window().maximize();
            Thread.sleep(2000);

            System.out.println("browser will be close");
            dr.quit();
     }
  }
```

设置浏览器大小

场景

设置浏览器窗口的大小有下面两个比较常见的用途：

在统一的浏览器大小下运行用例，可以比较容易的跟一些基于图像比对的工具进行结合，提升测试的灵活性及普遍适用性。比如可以跟 sikuli 结合，使用 sikuli 操作 flash；

在不同的浏览器大小下访问测试站点，对测试页面截图并保存，然后观察或使用图像比对工具对被测页面的前端样式进行评测。比如可以将浏览器设置成移动端大小(320x480)，然后访问移动站点，对其样式进行评估；

代码

将浏览器调整成移动端大小，然后访问移动站点，对移动站点的样式进行评估。

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
public class CloseBrowser {
    public static void main(String[] args) {
        WebDriver dr = new ChromeDriver();
        System.out.println("browser will be closed");
        dr.quit();
        System.out.println("browser is closed");
    }
}
```

讨论

webdriver 提供了很多调整浏览器窗口的接口，比如 move_to(移动窗口)，position(设置或获取浏览器的位置)。在一般情况下这些功能并不常用。

访问链接

情景

web UI 测试里最简单也是最基本的事情就是访问 1 个链接了。

在 python 的 webdrive 中，访问 url 时应该使用 get 方法。

代码

```
from selenium import webdriver
import time
dr = webdriver.Chrome()
url = 'http://www.baidu.com'
print "now access %s" %(url)
dr.get(url)
time.sleep(3)
dr.quit()
```

打印当前页面的 title 及 url

情景

测试中，访问 1 个页面然后判断其 title 是否符合预期是很常见的 1 个用例，所谓用例不够，title 来凑就是这个道理。更具体一点，假设 1 个页面的 title 应该是'hello world'，那么可以写这样的一个用例：访问该页面，获取该页面的 title，判断获取的值是否等于'hello world'。

获取当前页面的 url 也是非常重要的一个操作。在某些情况下，你访问一个 url，这时系统会自动对这个 url 进行跳转，这就是所谓的'重定向'。一般测试重定向的方法是访问这个 url，然后等待页面重定向完毕之后，获取当前页面的 url，判断该 url 是否符合预期。另外的一个常见的测试场景是提交了一个表单，如果表单内容通过了验证，那么则会跳转到一个新页面，如果未通过验证，则会停留在当前页面，此时获取当前页面的 url 则可以帮助我们判断表单提交的跳转是否符合预期。更具体一点，假如你在测试一个登陆页面，输入正确的登陆信息后，会跳转到系统首页。获取跳转后的 url 然后判断其是否与系统首页的 url 相符将是一个很不错的用例。

代码

```
# -*- coding: utf-8 -*-

from selenium import webdriver

from time import sleep

import os
```

```
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']
dr = webdriver.Chrome()
url = 'http://www.baidu.com'
dr.get(url)
print "title of current page is %s" %(dr.title)
print "url of current page is %s" %(dr.current_url)
sleep(1)
dr.quit()
```

前进和后退

场景

说实话，这两个功能一般不太常用。所能想到的场景大概也就是在几个页面间来回跳转，省去每次都 get url。

代码

```
from selenium import webdriver
from time import sleep
import os
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']

dr = webdriver.Chrome()

first_url = 'http://www.baidu.com'
print "now access %s" %(first_url)

dr.get(first_url)
```

```
sleep(1)

second_url = 'http://www.news.baidu.com'

print "now access %s" %(second_url)

dr.get(second_url)

sleep(1)


print "back to %s" %(first_url)

dr.back()

sleep(1)

print "forward to %s" %(second_url)

dr.forward()

sleep(1)

dr.quit()
```

简单的对象定位

场景

测试对象的定位和操作是 webdriver 的核心内容，其中操作又是建立在定位的基础之上,因此对象定位就越发显得重要了。

定位对象的目的一般有下面几种

操作对象

获得对象的属性，如获得测试对象的 class 属性，name 属性等等

获得对象的 text

获得对象的数量

webdriver 提供了一系列的对象定位方法，常用的有以下几种

id

name

class name

link text

partial link text

tag name

xpath

css selector

代码

html 代码 form.html

```
<html>
  <head>
    <meta http-equiv="content-type"
content="text/html;charset=utf-8" />
    <title>Form</title>
```

```
    <script type="text/javascript" async=""
src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.j
s"></script>

    <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />

    <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>

  </head>

  <body>

    <h3>simple login form</h3>

    <form class="form-horizontal">

      <div class="control-group">

        <label class="control-label" for="inputEmail">Email</label>

        <div class="controls">

          <input type="text" id="inputEmail" placeholder="Email"
name="email">

        </div>

      </div>

      <div class="control-group">

        <label class="control-label"
for="inputPassword">Password</label>

        <div class="controls">

          <input type="password" id="inputPassword"
placeholder="Password" name="password">

        </div>

      </div>

      <div class="control-group">

        <div class="controls">

          <label class="checkbox">
```

```
            <input type="checkbox"> Remember me
          </label>
          <button type="submit" class="btn">Sign in</button>
          <a href="#">register</a>
        </div>
      </div>
    </form>
  </body>
</html>
```

python 代码 simple_locate.rb

```
from selenium import webdriver
from time import sleep
import os
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()
file_path = 'file:///' + os.path.abspath('form.html')
print file_path


dr.get(file_path)


# by id
dr.find_element_by_id('inputEmail').click()


# by name
dr.find_element_by_name('password').click()
```

```
# by tagname
print dr.find_element_by_tag_name('form').get_attribute('class')


# by class_name
e = dr.find_element_by_class_name('controls')
dr.execute_script('$(arguments[0]).fadeOut().fadeIn()', e)
sleep(1)


# by link text
link = dr.find_element_by_link_text('register')
dr.execute_script('$(arguments[0]).fadeOut().fadeIn()', link)
sleep(1)


# by partial link text
link = dr.find_element_by_partial_link_text('reg')
dr.execute_script('$(arguments[0]).fadeOut().fadeIn()', link)
sleep(1)


# by css selector
div = dr.find_element_by_css_selector('.controls')
dr.execute_script('$(arguments[0]).fadeOut().fadeIn()', div)
sleep(1)


# by xpath
dr.find_element_by_xpath('/html/body/form/div[3]/div/label/input').c
lick()


sleep(2)
```

```
dr.quit()
```

讨论

上面例子里由于 html 文件中引用了 jquery，所以在执行 js 时可以使用 jquery 的$()及 fadeIn()等方法。如果你测试的页面没用包含 jquery 的话，这些方法是无效的。

定位一组对象

场景

从上一节的例子中可以看出，webdriver 可以很方便的使用 find_element 方法来定位某个特定的对象，不过有时候我们却需要定位一组对象，这时候就需要使用 find_elements 方法。

定位一组对象一般用于以下场景：

批量操作对象，比如将页面上所有的 checkbox 都勾上

先获取一组对象，再在这组对象中过滤出需要具体定位的一些对象。比如定位出页面上所有的 checkbox，然后选择最后一个

代码

checkbox.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>Checkbox</title>
        <script type="text/javascript" async="" src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
        <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
    </head>
    <body>
        <h3>checkbox</h3>
        <div class="well">
            <form class="form-horizontal">
                <div class="control-group">
                    <label class="control-label" for="c1">checkbox1</label>
                    <div class="controls">
                        <input type="checkbox" id="c1" />
                    </div>
                </div>
                <div class="control-group">
```

```
                        <label class="control-label"
for="c2">checkbox2</label>
                        <div class="controls">
                            <input type="checkbox" id="c2" />
                        </div>
                    </div>
                    <div class="control-group">
                        <label class="control-label"
for="c3">checkbox3</label>
                        <div class="controls">
                            <input type="checkbox" id="c3" />
                        </div>
                    </div>
                    <div class="control-group">
                        <label class="control-label"
for="r">radio</label>
                        <div class="controls">
                            <input type="radio" id="r" />
                        </div>
                    </div>
                </form>
            </div>
        </body>
    </html>
```

find_element.rb

```
#encoding: utf-8
require 'selenium-webdriver'
```

```
    dr = Selenium::WebDriver.for :chrome
    file_path = 'file:///' + File.expand_path(File.join('.',
'checkbox.html'))


    dr.get file_path


    # 选择所有的 checkbox 并全部勾上
    dr.find_elements(:css, 'input[type=checkbox]').each {|c| c.click}
    dr.navigate.refresh()
    sleep 1


    # 打印当前页面上有多少个 checkbox
    puts dr.find_elements(:css, 'input[type=checkbox]').size


    # 选择页面上所有的 input，然后从中过滤出所有的 checkbox 并勾选之
    dr.find_elements(:tag_name, 'input').each do |input|
        input.click if input.attribute(:type) == 'checkbox'
    end
    sleep 1


    # 把页面上最后 1 个 checkbox 的勾给去掉
    dr.find_elements(:css, 'input[type=checkbox]').last.click


    sleep 2
    dr.quit
```

讨论

checkbox.html 必须与 find_elments.rb 在同一级目录下

层级定位

场景

在实际的项目测试中，经常会有这样的需求：页面上有很多个属性基本相同的元素，现在需要具体定位到其中的一个。由于属性基本相当，所以在定位的时候会有些麻烦，这时候就需要用到层级定位。先定位父元素，然后再通过父元素定位子孙元素。

代码

下面的代码演示了如何通过层级定位来定位下拉菜单中的某一项。由于两个下拉菜单中每个选项的 link text 都相同，href 也一样，所以在这里就需要使用层级定位了。

具体思路是：先点击显示出 1 个下拉菜单，然后再定位到该下拉菜单所在的 ul，再定位这个 ul 下的某个具体的 link。在这里，我们定位第 1 个下拉菜单中的 Another action 这个选项。

level_locate.html

```
    <html>
        <head>
            <meta http-equiv="content-type"
content="text/html;charset=utf-8" />
            <title>Level Locate</title>
```

```
            <script type="text/javascript" async=""
src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.j
s"></script>

            <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />

        </head>

        <body>

            <h3>Level locate</h3>

            <div class="span3">

                <div class="well">

                    <div class="dropdown">

                        <a class="dropdown-toggle"
data-toggle="dropdown" href="#">Link1</a>

                        <ul class="dropdown-menu" role="menu"
aria-labelledby="dLabel" id="dropdown1" >

                            <li><a tabindex="-1"
href="#">Action</a></li>

                            <li><a tabindex="-1" href="#">Another
action</a></li>

                            <li><a tabindex="-1" href="#">Something else
here</a></li>

                            <li class="divider"></li>

                            <li><a tabindex="-1" href="#">Separated
link</a></li>

                        </ul>

                    </div>

                </div>

            </div>

            <div class="span3">

                <div class="well">

                    <div class="dropdown">
```

```
                    <a class="dropdown-toggle"
data-toggle="dropdown" href="#">Link2</a>

                    <ul class="dropdown-menu" role="menu"
aria-labelledby="dLabel" >

                        <li><a tabindex="-1"
href="#">Action</a></li>

                        <li><a tabindex="-1" href="#">Another
action</a></li>

                        <li><a tabindex="-1" href="#">Something else
here</a></li>

                        <li class="divider"></li>

                        <li><a tabindex="-1" href="#">Separated
link</a></li>

                    </ul>

                </div>

            </div>

        </div>

    </body>

    <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>

</html>
```

level_locate.py

```
# -*- coding: utf-8 -*-

from selenium import webdriver

from selenium.webdriver.support.ui import WebDriverWait

import time

import os
```

```
dr = webdriver.Chrome()

file_path =  'file:///' + os.path.abspath('level_locate.html')

dr.get(file_path)


dr.find_element_by_link_text('Link1').click()


WebDriverWait(dr, 10).until(lambda the_driver:
the_driver.find_element_by_id('dropdown1').is_displayed())

menu =
dr.find_element_by_id('dropdown1').find_element_by_link_text('Anothe
r action')


webdriver.ActionChains(dr).move_to_element(menu).perform()


time.sleep(2)


dr.quit()
```

讨论

move_to 方法实际上是模拟把鼠标移动到某个具体的测试对象上。

操作测试对象

场景

定位到具体的对象后，我们就可以对这个对象进行具体的操作，比如先前已经看到过的点击操作(click)。一般来说，webdriver 中比较常用的操作对象的方法有下面几个

click 点击对象

send_keys 在对象上模拟按键输入

clear 清除对象的内容，如果可以的话

代码

下面的代码演示了如何点击元素，如何往文本框中输入文字以及如何清空文字。

operate_element.html

```
    <html>
        <head>
            <meta http-equiv="content-type"
content="text/html;charset=utf-8" />
            <title>Level Locate</title>
            <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>
            <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />
        </head>
```

```
        <body>
            <h3>Level locate</h3>
            <div class="span3">
                <div class="well">
                    <div class="dropdown">
                        <a class="dropdown-toggle"
data-toggle="dropdown" href="#">Link1</a>
                        <ul class="dropdown-menu" role="menu"
aria-labelledby="dLabel" id="dropdown1" >
                            <li><a tabindex="-1"
href="#">Action</a></li>
                            <li><a tabindex="-1" href="#">Another
action</a></li>
                            <li><a tabindex="-1" href="#">Something else
here</a></li>
                            <li class="divider"></li>
                            <li><a tabindex="-1" href="#">Separated
link</a></li>
                        </ul>
                    </div>
                </div>
            </div>
            <div class="span3">
                <div class="well">
                    <div class="dropdown">
                        <a class="dropdown-toggle"
data-toggle="dropdown" href="#">Link2</a>
                        <ul class="dropdown-menu" role="menu"
aria-labelledby="dLabel" >
                            <li><a tabindex="-1"
href="#">Action</a></li>
```

```
                        <li><a tabindex="-1" href="#">Another
action</a></li>
                        <li><a tabindex="-1" href="#">Something else
here</a></li>
                        <li class="divider"></li>
                        <li><a tabindex="-1" href="#">Separated
link</a></li>
                    </ul>
                </div>
            </div>
        </div>
    </body>
    <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>
</html>
```

operate_element.py

```
# -*- coding: utf-8 -*-
from selenium import webdriver
import time
import os


dr = webdriver.Chrome()
file_path =  'file:///' + os.path.abspath('operate_element.html')
dr.get(file_path)


# click
dr.find_element_by_link_text('Link1').click()
time.sleep(1)
```

```
dr.find_element_by_link_text('Link1').click()


# send_keys
element = dr.find_element_by_name('q')
element.send_keys('something')
time.sleep(1)


# clear
element.clear()
time.sleep(1)


dr.quit()
```

send keys 模拟按键输入

场景

send_keys 方法可以模拟一些组合键操作，比如 ctrl+a 等。另外有时候我们需要在测试时使用 tab 键将焦点转移到下一个元素，这时候也需要 send_keys。在某些更复杂的情况下，还会出现使用 send_keys 来模拟上下键来操作下拉列表的情况。

代码

下面的代码演示了如何将 A 多行文本框中的内容清空并复制到 B 文本框中。

send_keys.html

```
   <html>

        <head>

            <meta http-equiv="content-type"
content="text/html;charset=utf-8" />

            <title>send keys</title>

            <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>

            <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />

        </head>

        <body>

            <h3>send keys</h3>

            <div class="row-fluid">

            <div class="span3">

                <div class="well">

                    <label>A</label>

                    <textarea rows="10", cols="10" id="A">I think
watir-webdriver is better than selenium-webdriver</textarea>

                </div>

            </div>

            <div class="span3">

                <div class="well">

                    <label>B</label>

                    <textarea rows="10", cols="10" id="B"></textarea>

                </div>

            </div>
```

```
            </div>
        </body>
        <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>
    </html>
```

send_keys.py

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from time import sleep
import os
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()
file_path = 'file:///' + os.path.abspath('send_keys.html')


dr.get(file_path)


# copy content of A
dr.find_element_by_id('A').send_keys((Keys.CONTROL, 'a'))
dr.find_element_by_id('A').send_keys((Keys.CONTROL, 'x'))
sleep(1)


# paste to B
dr.find_element_by_id('B').send_keys((Keys.CONTROL, 'v'))
sleep(1)


# # send keys to A
```

```
dr.find_element_by_id('A').send_keys('watir', '-', 'webdriver',
Keys.SPACE, 'is', Keys.SPACE, 'better')

sleep(2)



dr.quit()
```

处理 button group

场景

button group 就是按钮组，将一组按钮排列在一起。处理这种对象的思路一般是先找到 button group 的包裹(wrapper)div，然后通过层级定位，用 index 或属性去定位更具体的按钮。

代码

下面的代码演示了如何找到 second 这个按钮。其处理方法是先找到 button group 的父 div，class 为 btn-group 的 div，然后再找到下面所有的 div(也就是 button)，返回 text 是 second 的 div。

button_group.html

```
   <html>

       <head>

           <meta http-equiv="content-type"
content="text/html;charset=utf-8" />

           <title>button group</title>
```

```
            <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>
            <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />
            <script type="text/javascript">
                $(document).ready(function(){
                    $('.btn').click(function(){
                        alert($(this).text());
                    });
                });
            </script>
        </head>
        <body>
            <h3>button group</h3>
            <div class="row-fluid">
                <div class="span3">
                    <div class="well">
                        <div class="btn-toolbar">
                            <div class="btn-group">
                                <div class="btn">first</div>
                                <div class="btn">second</div>
                                <div class="btn">third</div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
```

```
        </body>

        <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>

    </html>
```

button_group.py

```
# -*- coding: utf-8 -*-

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

from time import sleep

import os

if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()

file_path = 'file:///' + os.path.abspath('button_group.html')


dr.get(file_path)


sleep(1)


# 定位 text 是 second 的按钮

buttons =
dr.find_element_by_class_name('btn-group').find_elements_by_class_na
me('btn')

for btn in buttons:

    if btn.text == 'second': print 'find second button'


sleep(1)
```

```
dr.quit()
```

讨论

自己查资料搞清楚 detect 方法的作用。

处理 button dropdown

场景

button dropdown 就是把按钮和下拉菜单弄到了一起。处理这种对象的思路一般是先点击这个按钮，等待下拉菜单显示出来，然后使用层级定位方法来获取下拉菜单中的具体项。

代码

下面的代码演示了如何找到 watir-webdriver 这个菜单项。其处理方法是先点击 info 按钮，然后等到下拉菜单出现后定位下拉菜单的 ul 元素，再定位 ul 元素中 link text 为 watir-webdriver 的 link，并点击之。

button_dropdown.html

```
   <html>

       <head>

           <meta http-equiv="content-type"
content="text/html;charset=utf-8" />

           <title>button group</title>
```

```
            <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>
            <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />
            <script type="text/javascript">
                $(document).ready(function(){
                    $('.btn').click(function(){
                        alert($(this).text());
                    });
                });
            </script>
        </head>
        <body>
            <h3>button group</h3>
            <div class="row-fluid">
                <div class="span3">
                    <div class="well">
                        <div class="btn-toolbar">
                            <div class="btn-group">
                                <div class="btn">first</div>
                                <div class="btn">second</div>
                                <div class="btn">third</div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
```

```
        </body>

        <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>

    </html>
```

button_dropdown.py

```
# -*- coding: utf-8 -*-

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

from time import sleep

import os

if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()

file_path = 'file:///' + os.path.abspath('button_group.html')


dr.get(file_path)


sleep(1)


# 定位text是second的按钮

buttons =
dr.find_element_by_class_name('btn-group').find_elements_by_class_na
me('btn')

for btn in buttons:

    if btn.text == 'second': print 'find second button'


sleep(1)
```

```
dr.quit()
```

处理 navs

场景

navs 可以看作是简单的类似于 tab 的导航栏。一般来说导航栏都是 ul+li。先定位 ul 再去层级定位 li 中的 link 基本就能解决问题。

代码

navs.html

```
    <html>

        <head>

            <meta http-equiv="content-type"
content="text/html;charset=utf-8" />

            <title>Navs</title>

            <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>

            <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />

            <script type="text/javascript">

                $(document).ready(

                    function(){

                        $('.nav').find('li').click(function() {


$(this).parent().find('li').removeClass('active');
```

```
                        $(this).addClass('active');
                    });
                }
            );
        </script>
    </head>
    <body>
        <h3>Navs</h3>
        <div class="row-fluid">
            <div class="span3">
                <ul class="nav nav-pills">
                    <li class="active">
                        <a href="#">Home</a>
                    </li>
                    <li><a href="#">Content</a></li>
                    <li><a href="#">About</a></li>
                </ul>
            </div>
        </div>
    </body>
    <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>
</html>
```

navs.py

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
```

```
from time import sleep

import os

if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()

file_path = 'file:///' + os.path.abspath('navs.html')


dr.get(file_path)


sleep(1)


# 方法1：层级定位，先定位ul 再定位li

dr.find_element_by_class_name('nav').find_element_by_link_text('Abou
t').click()

sleep(1)


# 方法2: 直接定位link

dr.find_element_by_link_text('Home').click()

sleep(1)


dr.quit()
```

处理面包屑

场景

在实际的测试脚本中，有可能需要处理面包屑。处理面包屑主要是获取其层级关系，以及获得当前的层级。一般来说当

前层级都不会是链接，而父层级则基本是以链接，所以处理面包屑的思路就很明显了。找到面包屑所在的 div 或 ul，然后再通过该 div 或 ul 找到下面的所有链接，这些链接就是父层级。最后不是链接的部分就应该是当前层级了。

代码

breadcrumb.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>breadcrumb</title>
        <script type="text/javascript" async="" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
        <script type="text/javascript">
            $(document).ready(
                function(){


                }
            );
        </script>
    </head>
    <body>
        <h3>breadcrumb</h3>
```

```
            <div class="row-fluid">
                <div class="span3">
                    <ul class="breadcrumb">
                        <li><a href="#">Home</a> <span
class="divider">/</span></li>
                        <li><a href="#">Library</a> <span
class="divider">/</span></li>
                        <li class="active">Data</li>
                    </ul>
                </div>
            </div>
        </body>
        <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>
    </html>
```

breadcrumb.rb

```
    #encoding: utf-8
    require 'selenium-webdriver'


    dr = Selenium::WebDriver.for :chrome
    file_path = 'file:///' + File.expand_path(File.join('.',
'breadcrumb.html'))
    dr.get file_path


    # 获得其父层级
    anstors = dr.find_element(:class,
'breadcrumb').find_elements(:tag_name, 'a').map { |link| link.text }
    p anstors
```

```
    sleep(1)


    # 获取当前层级
    # 由于页面上可能有很多 class 为 active 的元素
    # 所以使用层级定位最为保险
    puts dr.find_element(:class, 'breadcrumb').find_element(:class,
'active').text


    dr.quit()
```

处理对话框

场景

页面上弹出的对话框是自动化测试经常会遇到的一个问题。很多情况下这个弹出的对话框是一个 iframe，处理起来有点麻烦，需要进行 switch_to 操作。但现在很多前端框架的对话框都是 div 形式的，这就让我们的处理变得十分简单了。

处理对话框一般会做下面的一些事情

- 打开对话框
- 关闭对话框
- 操作对话框中的元素

代码

下面的代码演示了如何打开、关闭以及点击对话框中的链接

modal.html

```
   <html>

       <head>

           <meta http-equiv="content-type"
content="text/html;charset=utf-8" />

           <title>modal</title>

           <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>

           <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />

           <script type="text/javascript">

               $(document).ready(function(){

                   $('#click').click(function(){

                       $(this).parent().find('p').text('try
watir-webdriver right now!');

                   });

               });

           </script>

       </head>


       <body>

           <h3>modal</h3>

           <div class="row-fluid">

               <div class="span6">

                   <!-- Button to trigger modal -->

                   <a href="#myModal" role="button" class="btn
btn-primary" data-toggle="modal" id="show_modal">Click</a>
```

```
                <!-- Modal -->
                <div id="myModal" class="modal hide fade"
tabindex="-1" role="dialog" aria-labelledby="myModalLabel"
aria-hidden="true">
                  <div class="modal-header">
                    <button type="button" class="close"
data-dismiss="modal" aria-hidden="true">×</button>
                    <h3 id="myModalLabel">Modal header</h3>
                  </div>
                  <div class="modal-body">
                    <p>watir-webdriver is better than
slenium-webdriver</p>
                    <a href="#" id="click">click me</a>
                  </div>
                  <div class="modal-footer">
                    <button class="btn" data-dismiss="modal"
aria-hidden="true">Close</button>
                    <button class="btn btn-primary">Save
changes</button>
                  </div>
                </div>
              </div>
            </div>
        </body>
        <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>
    </html>
```

modal.py

```
# -*- coding: utf-8 -*-
```

```
from selenium import webdriver

from time import sleep

import os

import selenium.webdriver.support.ui as ui

if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()

file_path = 'file:///' + os.path.abspath('modal.html')


dr.get(file_path)


# 打开对话框

dr.find_element_by_id('show_modal').click()


wait = ui.WebDriverWait(dr, 10)

wait.until(lambda dr:
dr.find_element_by_id('myModal').is_displayed())


# 点击对话框中的链接

# 由于对话框中的元素被蒙板所遮挡，直接点击会报 Element is not clickable 的错误

# 所以使用 js 来模拟 click

# 在 watir-webdriver 中只需要 fire_event(:click) 就可以了

link = dr.find_element_by_id('myModal').find_element_by_id('click')

dr.execute_script('$(arguments[0]).click()', link)

sleep(2)


# 关闭对话框
```

```
buttons =
dr.find_element_by_class_name('modal-footer').find_elements_by_tag_n
ame('button')

buttons[0].click()



dr.quit()
```

获取测试对象的属性及内容

场景

获取测试对象的内容是前端自动化测试里一定会使用到的技术。比如我们要判断页面上是否显示了一个提示，那么我们就需要找到这个提示对象，然后获取其中的文字，再跟我们的预期进行比较。在 webdriver 中使用 element.attribute()方法可以获取 dom 元素(测试对象)的属性。

获取测试对象的属性能够帮我们更好的进行对象的定位。比如页面上有很多 class 都是'btn'的 div，而我们需要定位其中 1 个有具有 title 属性的 div。由于 selenium-webdriver 是不支持直接使用 title 来定位对象的，所以我们只能先把所有 class 是 btn 的 div 都找到，然后遍历这些 div，获取这些 div 的 title 属性，一旦发现具体 title 属性的 div，那么返回这个 div 既可。在 webdriver 中，使用 element.text()方法可以返回 dom 节点的内容(text)。

代码

下面的代码演示了如何获取测试对象的 title 属性和该对象的文字内容

attribute.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>attribute</title>
        <script type="text/javascript" async="" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
        <script type="text/javascript">
            $(document).ready(function(){
                $('#tooltip').tooltip({"placement": "right"});
            });
        </script>
    </head>

    <body>
        <h3>attribute</h3>
        <div class="row-fluid">
            <div class="span6">
```

```
                    <a id="tooltip" href="#" data-toggle="tooltip"
title="watir-webdriver better than selenium-webdriver">hover to see
tooltip</a>

                </div>

            </div>

        </body>

        <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>

    </html>
```

attribute.py

```
# -*- coding: utf-8 -*-

from selenium import webdriver

from time import sleep

import os

if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()

file_path = 'file:///' + os.path.abspath('attribute.html')


dr.get(file_path)


link = dr.find_element_by_id('tooltip')


sleep(1)

# 获得tooltip的内容

print link.get_attribute('data-original-title')
```

```
# 获取该链接的text
print link.text


dr.quit()
```

获取测试对象的 css 属性

场景

当你的测试用例纠结细枝末节的时候，你就需要通过判断元素的 css 属性来验证你的操作是否达到了预期的效果。比如你可以通过判断页面上的标题字号以字体来验证页面的显示是否符合预期。当然，这个是强烈不推荐的。因为页面上最不稳定的就是 css 了，css 变动频繁，而且通过属性也不能直观的判断页面的显示效果，还不如让人为的去看一眼，大问题一望即知。

代码

下面的代码演示了如何获取测试对象的 css 属性。

css.html

```
   <html>

       <head>

           <meta http-equiv="content-type"
content="text/html;charset=utf-8" />

           <title>attribute</title>
```

```
            <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>

            <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />

            <script type="text/javascript">

                $(document).ready(function(){

                    $('#tooltip').tooltip({"placement": "right"});

                });

            </script>

        </head>


        <body>

            <h3>attribute</h3>

            <div class="row-fluid">

                <div class="span6">

                    <a id="tooltip" href="#" data-toggle="tooltip"
title="watir-webdriver better than selenium-webdriver">hover to see
tooltip</a>

                </div>

            </div>

        </body>

        <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>

    </html>
```

css.py

```
# -*- coding: utf-8 -*-

from selenium import webdriver
```

```
from time import sleep
import os
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()
file_path = 'file:///' + os.path.abspath('css.html')


dr.get(file_path)


link = dr.find_element_by_id('tooltip')
print link.value_of_css_property('color')


print dr.find_element_by_tag_name('h3').value_of_css_property('font')


dr.quit()
```

获取测试对象的状态

场景

在 web 自动化测试中，我们需要获取测试对象的四种状态

- 是否显示。使用 element.is_displayed()方法；
- 是否存在。使用 find_element_by_xxx 方法，捕获其抛出的异常，如果存在异常的话则可以确定该元素不存在；
- 是否被选中。一般是判断表单元素，比如 radio 或 checkbox 是否被选中。使用 element.is_selected()方法；

- 是否 enable，也就是是否是灰化状态。使用 element.is_enabled()方法；

代码

status.html

```
   <html>
       <head>
           <meta http-equiv="content-type"
content="text/html;charset=utf-8" />
           <title>status</title>
           <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>
           <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />
           <script type="text/javascript">
               $(document).ready(function(){
                   $('#tooltip').tooltip({"placement": "right"});
               });
           </script>
       </head>

       <body>
           <h3>status</h3>
           <div class="row-fluid">
               <div class="span3">
                   <input name="user" placeholder="Disabled TextField"
disabled  />
```

```
                </div>
                <div class="span3">
                    <a class="btn disabled">Disabled Button</a>
                </div>
                <div class="span3">
                    <input name="radio" type="radio" />
                </div>
            </div>
        </body>
        <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>
    </html>
```

status.py

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from time import sleep
import os
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()
file_path = 'file:///' + os.path.abspath('status.html')
dr.get(file_path)


text_field = dr.find_element_by_name('user')
print text_field.is_enabled()


# 直接用enabled?方法去判断该button 的话返回的会是true
```

```
# 这是因为button 是使用css 方法去disabled 的，并不是真正的disable
# 这时候需要判断其class 里是否有disabled 这值来判断其是否处于disable 状态
print dr.find_element_by_class_name('btn').is_enabled()


# 隐藏掉text_field
# 判断其是否显示
dr.execute_script('$(arguments[0]).hide()', text_field)
print text_field.is_displayed()


# 使用click 方法选择raido
radio = dr.find_element_by_name('radio')
radio.click()
print radio.is_selected()


# 判断元素是否存在
try:
    dr.find_element_by_id('none')
except:
    print 'element does not exist'


dr.quit()
```

form 的操作

场景

表单对象的操作比较简单，只需要记住下面几点

- 使用 send_keys 方法往多行文本框和单行文本框赋值；

- 使用 click 方法选择 checkbox
- 使用 click 方法选择 radio
- 使用 click 方法点击 button
- 使用 click 方法选择 option，从而达到选中 select 下拉框中某个具体菜单项的效果

代码

form.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>form</title>
        <script type="text/javascript" async="" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
        <script type="text/javascript">
            $(document).ready(function(){
                $('input[type=submit]').click(function(){
                    alert('watir-webdriver is better than selenium webdriver');
                });
            });
        </script>
    </head>
```

```
    <body>
        <h3>form</h3>
        <div class="row-fluid">
            <div class="span6 well">
                <form>
                    <fieldset>
                        <legend>Legend</legend>
                        <label class="checkbox">
                            <input type="checkbox"> Check me out
                        </label>

                        <label class="radio">
                            <input type="radio"> select me
                        </label>

                        <label class="select">
                            <select>
                                <option>0</option>
                                <option>1</option>
                                <option>2</option>
                            </select> select one item
                        </label>

                        <input type="submit" class="btn"
value="submit" />
                    </fieldset>
                </form>
```

```
            </div>
        </div>
    </body>
    <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>
</html>
```

form.rb

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from time import sleep
import os
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()
file_path = 'file:///' + os.path.abspath('form.html')
dr.get(file_path)


# 选中checkbox
dr.find_element_by_css_selector('input[type=checkbox]').click()
sleep(1)


# 选中radio
dr.find_element_by_css_selector('input[type=radio]').click()
sleep(1)


# 选择下拉菜单中的最后一项
```

```
dr.find_element_by_tag_name('select').find_elements_by_tag_name('option')[-1].click()

sleep(1)


# 点击提交按钮

dr.find_element_by_css_selector('input[type=submit]').click()

sleep(10)


alert = dr.switch_to_alert()

print alert.text

alert.accept()


dr.quit()
```

执行 js

场景

如果你熟悉 js 的话，那么使用 webdriver 执行 js 就是一件很高效的事情了。在 webdriver 脚本中直接执行 js 的好处很多，这里就不一一枚举了。

webdriver 提供了 execute_script()接口来帮助我们完成这一工作。在实际的测试脚本中，以下两种场景是经常遇到的

- 在页面直接执行一段 js
- 在某个已经定位的元素的上执行 js

代码

下面的代码演示了如何在页面以及在已经定位的元素上执行 js

js.html

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8" />
    <title>js</title>
    <script type="text/javascript" async="" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
    <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
    <script type="text/javascript">
      $(document).ready(function(){
        $('#tooltip').tooltip({"placement": "right"});
      });
    </script>
  </head>

  <body>
    <h3>js</h3>
    <div class="row-fluid">
      <div class="span6 well">
```

```
        <a id="tooltip" href="#" data-toggle="tooltip"
title="watir-webdriver better than selenium-webdriver">hover to see
tooltip</a>

        <a class="btn">Button</a>

      </div>

    </div>

  </body>

  <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>

 </html>
```

js.python

```
# -*- coding: utf-8 -*-

from selenium import webdriver

from time import sleep

import os

if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()

file_path = 'file:///' + os.path.abspath('js.html')

dr.get(file_path)


# 在页面上直接执行js

dr.execute_script('$("#tooltip").fadeOut();')

sleep(1)


# 在已经定位的元素上执行js

button = dr.find_element_by_class_name('btn')
```

```
dr.execute_script('$(arguments[0]).fadeOut()', button)
sleep(1)


dr.quit()
```

处理 alert/confirm/prompt

场景

webdriver 中处理原生的 js alert confirm 以及 prompt 是很简单的。具体思路是使用 switch_to.alert()方法定位到 alert/confirm/prompt。然后使用 text/accept/dismiss/send_keys 按需进行操做

- text。返回 alert/confirm/prompt 中的文字信息
- accept。点击确认按钮
- dismiss。点击取消按钮，如果有的话
- send_keys。向 prompt 中输入文字

代码

下面代码简单的演示了如何去处理原生的 alert

alert.html

```
 <html>
   <head>
     <meta http-equiv="content-type"
content="text/html;charset=utf-8" />
```

```
    <title>alert</title>

    <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>

    <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />

    <script type="text/javascript">

      $(document).ready(function(){

        $('#tooltip').tooltip({"placement": "right"});

        $('#tooltip').click(function(){

            alert('watir-webdriver better than selenium-webdriver')

        });

      });

    </script>

  </head>


  <body>

    <div class="row-fluid">

      <div class="span6 well">

        <h3>alert</h3>

        <a id="tooltip" href="#" data-toggle="tooltip"
title="watir-webdriver better than selenium-webdriver">hover to see
tooltip</a>

      </div>

    </div>

  </body>

  <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>

</html>
```

alert.py

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from time import sleep
import os
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()
file_path = 'file:///' + os.path.abspath('alert.html')
dr.get(file_path)


# 点击链接弹出alert
dr.find_element_by_id('tooltip').click()


try:
    alert = dr.switch_to_alert()
    alert.accept()
except:
    print 'no alerts display'


sleep(1)
dr.quit()
```

wait

场景

Wait 类的使用场景是在页面上进行某些操作，然后页面上就会出现或隐藏一些元素，此时使用 Wait 类的 until 方法来等待这些效果完成以便进行后续的操作。另外页面加载时有可能会执行一些 ajax，这时候也需要去 WebDriverWait 的 until 的等待 ajax 的请求执行完毕。

具体一点的例子前面也曾出现过，点击一个链接然后会出现一个下拉菜单，此时需要先等待下拉菜单出现方可进行点击菜单项的操作。

这时候就需要用到 selenium.webdriver.support.ui.WebDriverWait 类，实例化该类时可以传入 timeout 的时间，单位是 s。

until 方法会一直等下去，直到

- 代码块中的内容为 true(不为 false 或没有抛出异常)
- 超时,也就是超过了 timeout 设置的时间

代码

下面代码演示了点击按钮后如何等待 label 出现。这个例子其实没有前面的下拉菜单例子实用。

wait.html

```
<html>
  <head>
    <meta http-equiv="content-type"
content="text/html;charset=utf-8" />
    <title>wait</title>
    <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>
    <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />
    <script type="text/javascript">
      $(document).ready(function(){
          $('#btn').click(function(){
            $('<p><span class="label
label-info">waitr-webdriver</span></p>').css('margin-top',
'1em').insertAfter($(this));
            $(this).addClass('disabled').unbind('click');
          });
      });
    </script>
  </head>

  <body>
    <div class="row-fluid">
      <div class="span6 well">
        <h3>wait</h3>
        <button class="btn btn-primary" id="btn" >Click</button>
      </div>
```

```
    </div>

  </body>

  <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>

 </html>
```

wait.py

```
# -*- coding: utf-8 -*-

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

from time import sleep

import os

import selenium.webdriver.support.ui as ui

if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()

file_path = 'file:///' + os.path.abspath('wait.html')


dr.get(file_path)


# 点击按钮

dr.find_element_by_id('btn').click()


wait = ui.WebDriverWait(dr, 10)

wait.until(lambda dr:
dr.find_element_by_class_name('label').is_displayed())


sleep(2)
```

```
dr.quit()
```

定位 frame 中的元素

场景

处理 frame 需要用到 2 个方法，分别是 switch_to_frame(name_or_id_or_frame_element)和 switch_to_default_content()

如何理解这个 switch_to_frame(name_or_id_or_frame_element)方法呢？可以简单记忆一下，如果这个 frame 有 name 和 id 属性那么就用这两个属性就好，如果没有的话可以先用 find_element_by_xxx 方法找到这个 frame 元素，然后把这个元素传进去，这也是可行的。

switch_to_frame 方法把当前定位的主体切换了 frame 里。怎么理解这句话呢？我们可以从 frame 的实质去理解。frame 中实际上是嵌入了另一个页面，而 webdriver 每次只能在一个页面识别，因此才需要用 switch_to_frame 方法去获取 frame 中嵌入的页面，对那个页面里的元素进行定位。

switch_to_default_content 方法的话则是从 frame 中嵌入的页面里跳出，跳回到最外面的原始页面中。

如果页面上只有 1 个 frame 的话那么这一切都是很好理解的，但如果页面上有多个 frame，情况有稍微有点复杂了。

代码

下面的代码中 frame.html 里有个 id 为 f1 的 frame，而 f1 中又嵌入了 id 为 f2 的 frame，该 frame 加载了百度的首页。

frame.html

```
 <html>
   <head>
     <meta http-equiv="content-type"
content="text/html;charset=utf-8" />
     <title>frame</title>
     <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>
     <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />
     <script type="text/javascript">
       $(document).ready(function(){
       });
     </script>
   </head>

   <body>
     <div class="row-fluid">
       <div class="span10 well">
```

```
        <h3>frame</h3>
        <iframe id="f1" src="inner.html" width="800",
height="600"></iframe>
      </div>
    </div>
  </body>
  <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>
</html>
```

inner.html

```
<html>
  <head>
    <meta http-equiv="content-type"
content="text/html;charset=utf-8" />
    <title>inner</title>
  </head>

  <body>
    <div class="row-fluid">
      <div class="span6 well">
        <h3>inner</h3>
        <iframe id="f2" src="http://www.baidu.com" width="700"
height="500"></iframe>
        <a href="javascript:alert('watir-webdriver better than
selenium webdriver;')">click</a>
      </div>
    </div>
  </body>
```

```
</html>
```

frame.py

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from time import sleep
import os
import selenium.webdriver.support.ui as ui
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()
file_path = 'file:///' + os.path.abspath('frame.html')


dr.get(file_path)


# 先到f1 再到f2
dr.switch_to_frame('f1')
dr.switch_to_frame('f2')


# 往f2 中的百度关键字文本框中输入内容
dr.find_element_by_id('kw').send_keys('watir-webdriver')


# 直接跳出所有frame
dr.switch_to_default_content()


# 再到f1
dr.switch_to_frame('f1')
```

```
dr.find_element_by_link_text('click').click()


sleep(2)
dr.quit()
```

讨论

假设页面上有 A、B 两个 frame，其中 B 在 A 内，那么定位 B 中的内容则需要先到 A，然后再到 B。如果是定位 A 中的内容，那么直接 switch_to_frame('A')就可以了;

switch_to.frame 的参数问题。官方说 name 是可以的，但是经过实验发现 id 也可以。所以只要 frame 中 id 和 name，那么处理起来是比较容易的。如果 frame 没有这两个属性的话，你可以直接加上，这对整个页面影响不大;

页面中使用 frame 会影响页面渲染速度，如果你遇到页面中有多个 frame 的情况，你完全可以提出 1 个页面前端性能的缺陷;

如果实在搞不定页面上的 frame，送你一句歌词：也许放弃才能靠近你。那么及时放弃跟此 frame 相关的用例才是明智之举;

action

场景

由于 webdriver 是要模拟真实的用户操作，因此 webdriver 的 Action 类中提供了很多与操作有关的方法。

下面列举一下 Action 类的一些主要方法

- key_down。模拟按键按下
- key_up。模拟按键弹起
- click
- send_keys
- double_click。鼠标左键双击
- click_and_hold。鼠标左键点击住不放
- release。鼠标左键弹起，可以与 click_and_hold 配合使用
- move_to_element。把鼠标移动到元素的中心点
- content_click。鼠标右键点击
- drag_and_drop。拖拽

代码

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


element = wd.find_element_by_link_text('xxxxx')
hov = ActionChains(wd).move_to_element(element)
hov.perform()
```

讨论

- 具体的 api 文档参考这里

上传文件

场景

- 上传文件的方法是找到上传文件的对象，通常是的对象。然后直接往这个对象 send_keys，传入需要上传文件的正确路径。绝对路径和相对路径都可以，但是上传的文件必须存在，否则会报错。

代码

upload_file.html

```
 <html>
   <head>
     <meta http-equiv="content-type"
content="text/html;charset=utf-8" />
     <title>upload_file</title>
     <script type="text/javascript" async=""
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js
"></script>
     <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/boo
tstrap-combined.min.css" rel="stylesheet" />
     <script type="text/javascript">
     </script>
```

```
    </head>


    <body>
      <div class="row-fluid">
        <div class="span6 well">
          <h3>upload_file</h3>
          <input type="file" name="file" />
        </div>
      </div>
    </body>
    <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/boots
trap.min.js"></script>
  </html>
```

upload_file.py

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from time import sleep
import os
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()
file_path = 'file:///' + os.path.abspath('upload_file.html')


dr.get(file_path)
```

```
dr.find_element_by_name('file').send_keys('./upload_file.md')


sleep(2)
dr.quit()
```

下载

场景

- webdriver 允许我们设置默认的文件下载路径。也就是说文件会自动下载并且存在设置的那个目录中。
- 下面会给出 chrome 和 firefox 浏览器的具体设置方法。

代码

```
import os


from selenium import webdriver


fp = webdriver.FirefoxProfile()


fp.set_preference("browser.download.folderList",2)
fp.set_preference("browser.download.manager.showWhenStarting",False)
fp.set_preference("browser.download.dir", os.getcwd())
fp.set_preference("browser.helperApps.neverAsk.saveToDisk",
"application/octet-stream")


browser = webdriver.Firefox(firefox_profile=fp)
browser.get("http://pypi.python.org/pypi/selenium")
```

```
browser.find_element_by_partial_link_text("selenium-2").click()
```

超时设置

场景

- webdriver 中可以设置很多的超时时间
- implicit_wait。识别对象时的超时时间。过了这个时间如果对象还没找到的话就会抛出异常

代码

```
ff = webdriver.Firefox()
ff.implicitly_wait(10) # seconds
ff.get("http://somedomain/url_that_delays_loading")
myDynamicElement = ff.find_element_by_id("myDynamicElement")
```

Remote Webdriver

- 场景
- 简单来说，我们可以把 remote webdriver 理解成在远程机器上运行 webdriver 脚本。
- 想像一下最简单的一个应用场景：你和你的同事两人一起开发一段 webdriver 脚本，然后你们需要在一个公共的环境去运行这段脚本。为什么要在公共的环境运行？那是因为每个人的开发机器是有差异的，但是如果用同一台测试机的话，那么环境差异的因素就可以基本排除。我们应该经常听到开

发说这样的话:"这个 bug 在我的环境上是好的啊！"。因为运行环境不同而造成的 bug 比比皆是，因此我们需要一个统一的运行环境来消除差异。

- 在这样的应用场景下，我们就需要使用 remote webdriver，我们在本地开发脚本，然后调用 remote webdriver，在测试机器上执行我们的测试。

安装

Remote Webdriver 的安装很简单。

首先下载 selenium-server-standalone-LAST-VERSION.jar。

然后运行 java -jar selenium-server-standalone.jar 命令。如果没有错误出现的话，这台机器已经被配置成远程机器了，以后 webdriver 就会在这台机器上启动浏览器，执行脚本。

启动 driver

下面的代码可以启动远程机器上的 driver，默认情况下这会打开 localhost 也就是本机上的 firefox 浏览器

```
    driver = Selenium::WebDriver.for(:remote)
```

如果你的 remote webdriver 不在本地运行，而且你又想指定除 firefox 以外的浏览器，那么使用下面的代码

```
    driver = Selenium::WebDriver.for(:remote, :url =>
"http://myserver:4444/wd/hub", :desired_capabilities => :chrome)
```

通常情况下 myserver 可以是 192.168.x.x 之类的 ip 地址。

另外还可以通过配置 Selenium::WebDriver::Remote::Capabilities 来实现更加定制化的浏览器配置，这个超出本文范围，不做描述。

使用 watir-webdriver 启动 driver

可以使用下面的代码让 watir-webdriver 也使用 remote webdriver 模式

```
    browser = Watir::Browser.new(:remote,
{desired_capabilities: :chrome, url: "http://myserver:4444/wd/hub"})
```

java 版本

```
   // We could use any driver for our tests...
    DesiredCapabilities capabilities = new DesiredCapabilities();

    // ... but only if it supports javascript
    capabilities.setJavascriptEnabled(true);

    // Get a handle to the driver. This will throw an exception
    // if a matching driver cannot be located
    WebDriver driver = new RemoteWebDriver(capabilities);

    // Query the driver to find out more information
    Capabilities actualCapabilities = ((RemoteWebDriver)
driver).getCapabilities();

    // And now use it
    driver.get("http://www.google.com");
```

注意，java 版本的代码我没有时间去调试，这里只是把 wiki 上的代码放出来而已。另外 remote server 在发生错误时会自动截图，下面是获得截图的代码

```
    public String extractScreenShot(WebDriverException e) {
        Throwable cause = e.getCause();
        if (cause instanceof ScreenshotException) {
            return ((ScreenshotException)
cause).getBase64EncodedScreenshot();
        }
        return null;
    }
```

python 版本

```
    c = webdriver.DesiredCapabilities.CHROME
    driver = webdriver.Remote(command_executor='http://127.0.0.1:4444/wd/hub', desired_capabilities=c)
```

注意，python binding 的 wiki 中使用的启动 remote webdriver 的代码跟我上面给出的不太相同，可能是因为我的 selenium 版本较低(30)，最新版本的同学可以试试 wiki 上的代码。

cookie

场景

webdriver 可以读取并添加 cookie。有时候我们需要验证浏览器中是否存在某个 cookie，因为基于真实的 cookie 的测试是无法通过白盒和集成测试完成的。

另外更加常见的一个场景是自动登陆。有很多系统的登陆信息都是保存在 cookie 里的，因此只要往 cookie 中添加正确的值就可以实现自动登陆了。什么图片验证码、登陆的用例就都是浮云了。

代码

下面的代码演示了如何自动登陆百度。其中敏感信息我使用了 xxxx 来代替。

cookie.py

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from time import sleep
import os
if 'HTTP_PROXY'in os.environ: del os.environ['HTTP_PROXY']


dr = webdriver.Chrome()


url = 'http://www.baidu.com'
dr.get(url)


print dr.get_cookies()
dr.delete_all_cookies()
dr.add_cookie({'name': 'BAIDUID', 'value': 'xxxxxx'})
dr.add_cookie({'name': 'BDUSS', 'value': 'xxxxxx'})
```

```
dr.get(url)


sleep(3)
dr.quit()
```