作者：深圳51TESTING

# WebDriver 进阶

欢迎阅读WebDriver进阶讲义。本篇讲义将会重点介绍Selenium WebDriver API的重点使用方法，以及使用模块化和参数化进行自动化测试的设计。

## WebDriver API 进阶使用

### 元素定位

从之前的讲义和学习中，我们知道，WebDriver API的调用以及自动化测试，务必从页面元素的定位开始，那么回顾之前的内容，WebDriver提供了一系列的定位符以便使用元素定位方法。常见的定位符有以下几种：

- id
- name
- class name
- tag
- link text
- partial link text
- xpath
- css selector

那么我们以下的操作将会基于上述的定位符进行定位操作。

对于元素的定位，WebDriver API可以通过定位简单的元素和一组元素来操作。在这里，我们需要告诉Selenium如何去找元素，以至于他可以充分的模拟用户行为，或者通过查看元素的属性和状态，以便我们执行一系列的检查。

在Selenium2中，WebDriver提供了多种多样的`find_element_by`方法在一个网页里面查找元素。这些方法通过提供过滤标准来定位元素。当然WebDriver也提供了同样多种多样的`find_elements_by`的方式去定位多个元素。

Selenium2提供的8个`find_element_by`方法去定位元素。在这里我们来具体查看每个方法的详细使用方式。下面的表格将会列出这些具体的方法：

| 方法Method | 描述Description | 参数Argument | 示例Example |
| --- | --- | --- | --- |
| `id` | 该方法通过ID的属性值去定位查找单个元素 | id: 需要被查找的元素的ID | `find_element_by_id('search')` |
| `name` | 该方法通过name的属性值去定位查找单个元素 | name: 需要被查找的元素的名称 | `find_element_by_name('q')` |
| `class name` | 该方法通过class的名称值去定位查找单个元素 | class_name: 需要被查找的元素的类名 | `find_element_by_class_name('input-text')` |
| `tag_name` | 该方法通过tag的名称值去定位查找单个元素 | tag: 需要被查找的元素的标签名称 | `find_element_by_tag_name('input')` |
| `link_text` | 该方法通过链接文字去定位查找单个元素 | link_text: 需要被查找的元素的链接文字 | `find_element_by_link_text('Log In')` |
| `partial_link_text` | 该方法通过部分链接文字去定位查找单个元素 | link_text: 需要被查找的元素的部分链接文字 | `find_element_by_partial_link_text('Long')` |
| `xpath` | 该方法通过XPath的值去定位查找单个元素 | xpath: 需要被查找的元素的xpath | `find_element_by_xpath('//*[@id="xx"]/a')` |
| `css_selector` | 该方法通过CSS选择器去定位查找单个元素 | css_selector: 需要被查找的元素的ID | `find_element_by_css_selector('#search')` |

接下来的列表将会详细展示`find_elements_by`的方法集合。这些方法依据匹配的具体标准返回一系列的元素。

| 方法Method | 描述Description | 参数Argument | 示例Example |
| --- | --- | --- | --- |
| | 该方法通过ID的属性 | | |

| | | | |
|---|---|---|---|
| `id` | 该方法通过ID的属性值去定位查找多个元素 | id: 需要被查找的元素的ID | `find_elements_by_id('search')` |
| `name` | 该方法通过name的属性值去定位查找多个元素 | name: 需要被查找的元素的名称 | `find_elements_by_name('q')` |
| `class_name` | 该方法通过class的名称值去定位查找多个元素 | class_name: 需要被查找的元素的类名 | `find_elements_by_class_name('input-text')` |
| `tag_name` | 该方法通过tag的名称值去定位查找多个元素 | tag: 需要被查找的元素的标签名称 | `find_elements_by_tag_name('input')` |
| `link_text` | 该方法通过链接文字去定位查找多个元素 | link_text: 需要被查找的元素的链接文字 | `find_elements_by_link_text('Log In')` |
| `partial_link_text` | 该方法通过部分链接文字去定位查找多个元素 | link_text: 需要被查找的元素的部分链接文字 | `find_elements_by_partial_link_text('Long')` |
| `xpath` | 该方法通过XPath的值去定位查找多个元素 | xpath: 需要被查找的元素的xpath | `find_elements_by_xpath("//div[contains(@class,'list')]")` |
| `css_selector` | 该方法通过CSS选择器去定位查找多个元素 | css_selector: 需要被查找的元素的ID | `find_element_by_css_selector('.input_class')` |

### 依据ID查找

请查看如下HTML的代码，以便实现通过ID的属性值去定义一个查找文本框的查找：

```
<input id="search" type="text" name="q" value=""
       class="input-text" maxlength="128" autocomplete="off"/>
```

根据上述代码，这里我们使用`find_element_by_id()`的方法去查找搜索框并且检查它的最大长度`maxlength`属性。我们通过传递ID的属性值作为参数去查找，参考如下的代码示例：

```
def test_search_text_field_max_length(self):
    # get the search textbox
    search_field = self.driver.find_element_by_id("search")
    # check maxlength attribute is set to 128
    self.assertEqual("128", search_field.get_attribute("maxlength"))
```

如果使用`find_elements_by_id()`方法，将会返回所有的具有相同ID属性值的一系列元素。

### 依据名称name查找

这里还是根据上述ID查找的HTML代码，使用`find_element_by_name`的方法进行查找。参考如下的代码示例：

```
# get the search textbox
self.search_field = self.driver.find_element_by_name("q")
```

同样，如果使用`find_elements_by_name()`方法，将会返回所有的具有相同name属性值的一系列元素。

### 依据class name查找

除了上述的ID和name的方式查找，我们还可以使用class name的方式进行查找和定位。

> 事实上，通过ID，name或者类名class name查找元素是最提倡推荐的和最快的方式。当然Selenium2 WebDriver也提供了一些其他的方式，在上述三类方式条件不足，查找无效的时候，可以通过这些其他方式来查找。这些方式将会在后续的内容中讲述。

请查看如下的HTML代码，通过改代码进行练习和理解.

```
<button type="submit" title="Search" class="button">
  <span><span>Search</span></span>
</button>
```

根据上述代码，使用`find_element_by_class_name()`方法去定位元素。

```
def test_search_button_enabled(self):
    # get Search button
    search_button = self.driver.find_element_by_class_name("button")
    # check Search button is enabled
    self.assertTrue(search_button.is_enabled())
```

同样的如果使用`find_elements_by_class_name()`方法去定位元素，将会返回所有的具有相同name属性值的一系列元素。

### 依据标签名tag name查找

利用标签的方法类似于利用类名等方法进行查找。我们可以轻松的查找出一系列的具有相同标签名的元素。例如我们可以通过查找表中的`<tr>`来获取行数。

下面有一个HTML的示例，这里在无序列表中使用了`<img>`标签。

```
<ul class="promos">
    <li>
        <a href="http://demo.magentocommerce.com/home-decor.html">
            <img src="/media/wysiwyg/homepage-three-column-promo-
        01B.png" alt="Physical &amp;amp; Virtual Gift Cards">
        </a>
    </li>
    <li>
        <a href="http://demo.magentocommerce.com/vip.html">
            <img src="/media/wysiwyg/homepage-three-column-promo-
        02.png" alt="Shop Private Sales - Members Only">
        </a>
    </li>
    <li>
        <a href="http://demo.magentocommerce.com/accessories/
        bags-luggage.html">
            <img src="/media/wysiwyg/homepage-three-columnpromo-
        03.png" alt="Travel Gear for Every Occasion">
        </a>
    </li>
</ul>
```

这里面我们使用`find_elements_by_tag_name()`的方式去获取全部的图片，在此之前，我们将会使用`find_element_by_class_name()`去获取到指定的`<ul>`。

具体代码如下：

```
def test_count_of_promo_banners_images(self):
    # get promo banner list
    banner_list = self.driver.find_element_by_class_name("promos")
    # get images from the banner_list
    banners = banner_list.find_elements_by_tag_name("img")
    # check there are 20 tags displayed on the page
    self.assertEqual(20, len(banners))
```

### 依据链接文字link查找

链接文字查找通常比较简单。使用`find_element_by_link_text`请查看以下示例

```
<a href="#header-account" class="skip-link skip-account">
    <span class="icon"></span>
    <span class="label">Account</span>
</a>
```

测试代码如下：

```
def test_my_account_link_is_displayed(self):
    # get the Account link
```

```
    account_link =
    self.driver.find_element_by_link_text("ACCOUNT")
    # check My Account link is displayed/visible in
    # the Home page footer
    self.assertTrue(account_link.is_displayed())
```

### 依据部分链接文字partial text查找

这里依旧使用上述的列子进行代码编写：

```
def test_account_links(self):
    # get the all the links with Account text in it
    account_links = self.driver.\
    find_elements_by_partial_link_text("ACCOUNT")
    # check Account and My Account link is
    displayed/visible in the Home page footer
    self.assertTrue(2, len(account_links))
```

### 依据XPath进行查找

XPath是一种在XML文档中搜索和定位节点node的一种查询语言。所有的主流Web浏览器都支持XPath。Selenium2可以用强大的XPath在页面中查找元素。

常用的XPath的方法有`starts-with()`，`contains()`和`ends-with()`等

> 若想要了解更多关于XPath的内容，请查看http://www.w3schools.com/XPath/

如下有一段HTML代码，其中里面的`<img>`没有使用ID，name或者类属性，所以我们无法使用之前的方法。亚这里我们可以通过`<img>`的`alt`属性，定位到指定的tag。

```
<ul class="promos">
    <li>
        <a href="http://demo.magentocommerce.com/home-decor.html">
            <img src="/media/wysiwyg/homepage-three-column-promo-
        01B.png" alt="Physical &amp;amp; Virtual Gift Cards">
        </a>
    </li>
    <li>
        <a href="http://demo.magentocommerce.com/vip.html">
            <img src="/media/wysiwyg/homepage-three-column-promo-
        02.png" alt="Shop Private Sales - Members Only">
        </a>
    </li>
    <li>
        <a href="http://demo.magentocommerce.com/accessories/
        bags-luggage.html">
            <img src="/media/wysiwyg/homepage-three-columnpromo-
        03.png" alt="Travel Gear for Every Occasion">
        </a>
    </li>
</ul>
```

具体代码如下：

```
def test_vip_promo(self):
    # get vip promo image
    vip_promo = self.driver.\
    find_element_by_xpath("//img[@alt='Shop Private Sales - Members Only']")
    # check vip promo logo is displayed on home page
    self.assertTrue(vip_promo.is_displayed())
    # click on vip promo images to open the page
    vip_promo.click()
    # check page title
    self.assertEqual("VIP", self.driver.title)
```

当然，如果使用`find_elements_by_xpath()`的方法，将会返回所有匹配了XPath查询的元素。

### 依据CSS选择器进行查找

CSS是一种设计师用来描绘HTML文档的视觉的层叠样式表。一般来说CSS用来定位多种多样的风格，同时可以用来是同样的标签使用同样的风格等。类似于XPath，Selenium2也可以使用CSS选择器来定位元素。

请查看如下的HTML文档。

```
<div class="minicart-wrapper">
    <p class="block-subtitle">Recently added item(s)
        <a class="close skip-link-close" href="#" title="Close">×</a>
    </p>
    <p class="empty">You have no items in your shopping cart.
    </p>
</div>
```

我们来创建一个测试，验证这些消息是否正确。

```
def test_shopping_cart_status(self):
    # check content of My Shopping Cart block on Home page
    # get the Shopping cart icon and click to open the
    # Shopping Cart section
    shopping_cart_icon = self.driver.\
    find_element_by_css_selector("div.header-minicart
                                 span.icon")
    shopping_cart_icon.click()
    # get the shopping cart status
    shopping_cart_status = self.driver.\
    find_element_by_css_selector("p.empty").text
    self.assertEqual("You have no items in your shopping cart.",
    shopping_cart_status)
    # close the shopping cart section
    close_button = self.driver.\
    find_element_by_css_selector("div.minicart-wrapper
                                 a.close")
    close_button.click()
```

## 鼠标事件

Web测试中，有关鼠标的操作，不只是单击，有时候还要做右击、双击、拖动等操作。这些操作包含在ActionChains类中。

常用的鼠标方法：

- context_click() ##右击
- douch_click() ##双击
- drag_and_drop() ##拖拽
- move_to_element() ##鼠标停在一个元素上
- click_and_hold() # 按下鼠标左键在一个元素上

例子：

```
#图3.4
#如图3.x4，假如一个web 应用的列表文件提供了右击弹出快捷菜单的的操作。可以通过context_click()
#方法模拟鼠标右键，参考代码如下：
#引入ActionChains 类
from selenium.webdriver.common.action_chains import ActionChains
...
#定位到要右击的元素
right =driver.find_element_by_xpath("xx")
#对定位到的元素执行鼠标右键操作
ActionChains(driver).context_click(right).perform()
```

```
#引入ActionChains 类
from selenium.webdriver.common.action_chains import ActionChains
...
#定位到要双击的元素
double =driver.find_element_by_xpath("xxx")
#对定位到的元素执行鼠标双击操作
ActionChains(driver).double_click(double).perform()
```

## 键盘事件

键盘操作经常处理的如下：

| 代码 | 描述 |
| --- | --- |
| `send_keys(Keys.BACK_SPACE)` | 删除键(BackSpace) |
| `send_keys(Keys.SPACE)` | 空格键(Space) |
| `send_keys(Keys.TAB)` | 制表键(Tab) |
| `send_keys(Keys.ESCAPE)` | 回退键(Esc) |
| `send_keys(Keys.ENTER)` | 回车键(Enter) |
| `send_keys(Keys.CONTROL,'a')` | 全选（Ctrl+A） |
| `send_keys(Keys.CONTROL,'c')` | 复制（Ctrl+C） |

代码如下

```
#coding=utf-8
from selenium import webdriver
#引入Keys 类包
from selenium.webdriver.common.keys import Keys
import time
driver = webdriver.Firefox()
driver.get("http://www.baidu.com")
#输入框输入内容
driver.find_element_by_id("kw").send_keys("selenium")
time.sleep(3)
#删除多输入的一个m
driver.find_element_by_id("kw").send_keys(Keys.BACK_SPACE)
time.sleep(3)
#输入空格键+ “教程”
driver.find_element_by_id("kw").send_keys(Keys.SPACE)
driver.find_element_by_id("kw").send_keys(u"教程")
time.sleep(3)
#ctrl+a 全选输入框内容
driver.find_element_by_id("kw").send_keys(Keys.CONTROL,'a')
```

# 模块化与类库

## 线性测试

至此之前，我们介绍的测试脚本，尽管使用了unittest测试框架，但是测试是按照指定的线路进行的，是线性的测试，完全遵循了脚本的执行顺序。

测试脚本1

```
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("http://wwww.xxx.com")
driver.find_element_by_id("tbUserName").send_keys("username")
driver.find_element_by_id("tbPassword").send_keys("123456")
driver.find_element_by_id("btnLogin").click()
#执行具体用例操作
......
driver.quit ()
```

如上图，其实登录的模块可以共用。

## 模块化

模块话是自动化测试的第一个延伸和基础。需要对自动化重复编写的脚本进行重构（refactor），将重复的脚本抽取出来，放到指定的代码文件中，作为共用的功能模块。

测试脚本1：Login.py

```
#登录模块
def login():
driver.find_element_by_id("tbUserName").send_keys("username")
driver.find_element_by_id("tbPassword").send_keys("456123")
driver.find_element_by_id("btnLogin").click()
```

另一份文件 quit.py

```
#退出模块
def quit():
.............
```

自动化的测试：代码如下

```
#coding=utf-8
from selenium import webdriver
import login,quit_ #调用登录、退出模块
driver = webdriver.Firefox()
driver.get("http://wwww.xxx.com")
#调用登录模块
login.login()
#其它个性化操作
......
#调用退出模块
quit.quit()
```

# 参数化驱动

## 数据驱动

如果说模块化是自动化测试的第一步，那么数据驱动是自动化的第二步，从本意上来讲。数据改变更新驱动自动化的执行。从而引起测试结果的改变。其实类似于参数化。

示例代码

```
#coding=utf-8
from selenium import webdriver
import time
values=['selenium','webdriver',u'软件自动化测试']
# 执行循环
for serch in values:
    driver = webdriver.Firefox()
    driver.get("https://www.baidu.com")
    driver.find_element_by_id("kw").send_keys(serch)
    time.sleep(3)
    .....
```

关于参数化驱动，我们可以将数据放到csv中，然后通过读取csv的数据进行自动化测试。

## 关键字驱动

这里的数据换成了特别的数据，就是关键字。