**1、简单元素操作**

在前面我已经学会了定位元素，定位只是第一步，定位之后需要对这个元素进行操作，或点击（按钮）或输入（输入框）或提交（表单），下面我们就来认识一下这些最常用的方法。

在WebDriver 中，大多简单有趣的页面交互的方法都将通过 **WebElement** 接口提供。

**最常用的操作页面元素的方法有下面几个：**

** clear() --- 清除文本，如果是一个文件输入框**

** send\_keys(value) ---- 在元素上模拟按键输入**

** click() ---- 单击元素**

**maximize\_window() ---- 最大化浏览器**

driver.find\_element\_by\_id("input").send\_keys("selenium")

driver.find\_element\_by\_id("input").clear()

sleep(2)

driver.find\_element\_by\_id("input").send\_keys("selenium")

**clear()**方法用于清除文本输入框中的内容，例如登录框内一般默认会有“账号”“密码”等提示信息，用于引导用户输入正确的数据，如果直接向输入框中输入数据，可能会与输入框中的提示信息拼接，本来用户输入为“username”,结果与提示信息拼接为“帐号username”，从而造成输入信息的错误；这个时候可以先使用clear()方法清除输入框内的提示信息再进行输入。

**send\_keys()**方法模拟键盘输入向输入框里输入内容。如上面的例子中通过这个方法向用户名和密码框中输入用户名和密码。

**click()**方法可以用来单击一个按钮，前提是它是可以被点击元素，它与send\_keys()方法是web 页面操作中最常用到的两个方法。其实click()方法不仅仅用于点击一个按钮，还可以单击任何可以点击文字/图片连接、复选框、单选框、甚至是下拉框等。

**2、WebElement 接口常用方法**

通常所有的要与页面交互的方法都是由 WebElement 接口提供。包括前第一小节所讲到的定位方法和上一节中的3个方法都由其提供。除此之外，WebElement 还提供了另外一些非常有用的方法。

下面我们就来学习这些方法的用处。

**submit()**

submit()方法用于提交表单，这里特别**用于没提交按钮的情况**，例如，搜索框输入关键字之后的“回车”操作，那么就可以通过submit()来提交搜索框的内容。

driver.find\_element\_by\_id('search-button').submit()

通过上面的例子，我们通过定位360搜索框并通过submit()提交搜索框的内容，同样达到点击“搜索”按钮的效果。有些时候submit()可以与click()方法互换来使用，submit()同样可以提交一个按钮。

**size 返回元素的尺寸。**

**text 返回元素的文本**

**get\_attribute("name") 获得属性值。**

#获得输入框的尺寸

size=driver.find\_element\_by\_id('input').size

print size

#获得搜索结果的text信息

driver.find\_element\_by\_xpath("//input[@id='input' and @class='placeholder']").send\_keys("selenium")

# driver.find\_element\_by\_xpath("//\*[@id='search-button']").click()

#输出text值

# text = driver.find\_element\_by\_xpath("/html/body/div[2]/div[1]/div[1]/ul/li[2]/h3/a/em").text

# print text

#返回元素的属性值，可以是id、name、type或元素拥有的其它任意属性

attribute=driver.find\_element\_by\_id("search-button").get\_attribute('type')

print attribute

**size** 用于获取元素的宽、高。

**text** 用于获得指定元素的值。

**get\_attribute()** 用户获取元素的属性的值。

WebElement 接口的其它更多方法请参考webdriver API 官方文档。

**3、鼠标操作**

在webdriver中，关于鼠标操作的方法，由ActionChains类提供。ActionChains类提供的常见的方法有：

**move\_to\_element() 鼠标悬停**

**perfrom() 执行所有ActionChains中存储的行为**

在实际操作中，鼠标悬停，弹出下拉菜单，是一个非常常见的功能。如下图：



鼠标放到163邮箱的“登录”框，自动弹出登录窗口。

在webdriver中，关于鼠标操作的方法，由ActionChains类提供方法名是：

**move\_to\_element()**

使用该方法，首先要把ActionChains类导进来，引入ActionChains类的方法如下：

from selenium.webdriver.common.action\_chains import ActionChains

下面脚本演示下如何把鼠标悬停到上图的登陆框中。

#coding=utf-8

from selenium import webdriver

#引入ActionChains类

from selenium.webdriver.common.action\_chains import ActionChains

dr = webdriver.Firefox()

dr.get("http://www.163.com/")

#定位到要悬停的元素

login\_button = dr.find\_element\_by\_class\_name("ntes-nav-login-title")

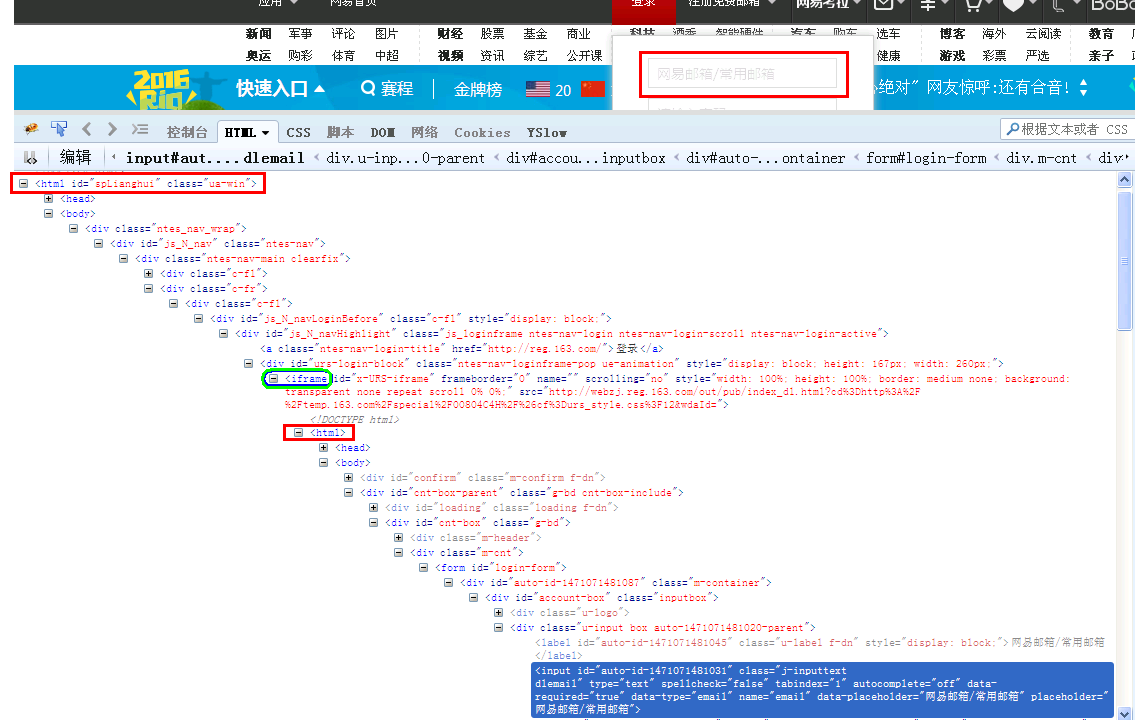
#把鼠标悬停到定位到的元素上

ActionChains(dr).move\_to\_element(login\_button).perform()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**4、多表单切换**

接上面的例子，如果我们想在弹出的窗口中输入用户名和密码，直接用：

dr.find\_element\_by\_id(“xxxx”).send\_keys(“xxx”)，是无法输入的。原因是，这里采用了frame页面嵌套技术，webdriver每次只能在一个页面上识别元素。如下图：

这时候，就需要通过**switch\_to\_frame()**方法将当前定位的主题切换的内嵌的frame里面去。

下面脚本，演示如何切换到指定frame并进行操作：

#coding=utf-8

from selenium import webdriver

from time import \*

#引入ActionChains类

from selenium.webdriver.common.action\_chains import ActionChains

dr = webdriver.Firefox()

dr.get("http://www.163.com/")

#定位到要悬停的元素

login\_button = dr.find\_element\_by\_class\_name("ntes-nav-login-title")

#把鼠标悬停到定位到的元素上

ActionChains(dr).move\_to\_element(login\_button).perform()

sleep(3)

**#切换到登录框的frame中**

**frame\_add = dr.find\_element\_by\_xpath("//iframe[@id='x-URS-iframe']")**

**dr.switch\_to\_frame(frame\_add)**

#接下来就可以正常操作元素了

#我们的手工操作是，先把鼠标定位到输入框内，然后在输入账号

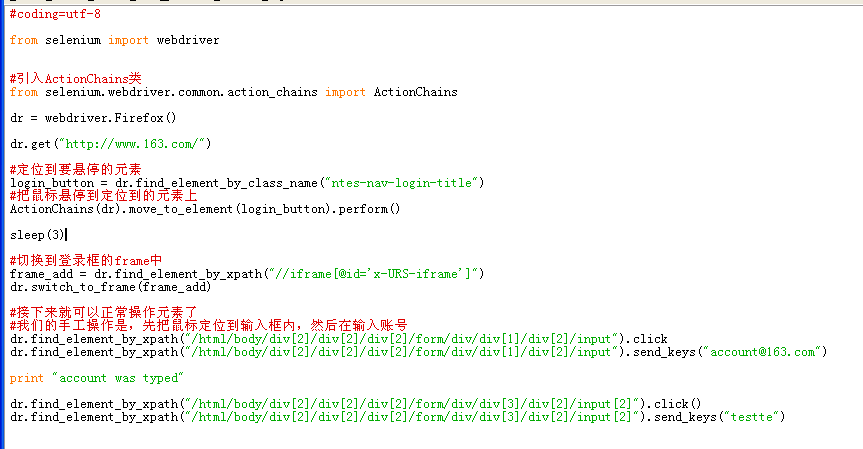
dr.find\_element\_by\_xpath("/html/body/div[2]/div[2]/div[2]/form/div/div[1]/div[2]/input").click

dr.find\_element\_by\_xpath("/html/body/div[2]/div[2]/div[2]/form/div/div[1]/div[2]/input").send\_keys("account@163.com")

print "account was typed"

dr.find\_element\_by\_xpath("/html/body/div[2]/div[2]/div[2]/form/div/div[3]/div[2]/input[2]").click()

dr.find\_element\_by\_xpath("/html/body/div[2]/div[2]/div[2]/form/div/div[3]/div[2]/input[2]").send\_keys("testte")



如果要在几个表单中切换，需要先切换到最初的表单中，再切换到另一个表单

例子：

dr.switch\_to\_frame(“表单1”)

#切换到默认表单

dr.switch\_to\_default\_content()

#切换到另一个表单

dr.switch\_to\_frame(“表单2”)

**5、sleep 休眠方法**

有时间我们希望脚本执行到某一位置时做固定时间的等待。那么可以使用sleep()方法，sleep()是由Python 的time 模块提供。

#coding=UTF-8

from selenium import webdriver

from time import \*

driver = webdriver.Firefox()

driver.get("http://www.so.com")

**sleep(2)**

driver.find\_element\_by\_link\_text(u"百科").click()

**sleep(2)**

driver.quit()

当执行到sleep()方法时会固定的等待所设置的时长，然后再继续执行。sleep()方法默认参数以秒为单位，如果设置时长小于1秒，可以用小数点表示，如：sleep(0.5)

**6、隐式等待**

隐式等待是通过一定的时长等待页面的某个元素加载完成。如果超出了设置的时长，元素还没有被加载出来，则抛出异常：NoSuchElementException。 Webdriver提供了implicitly\_wait()方法来实现隐式等待，默认的等待时间为0 。用法如下：

from selenium import webdriver

driver = webdriver.Firefox()

#设置等待超时时间

driver.implicitly\_wait(100)

#打开系统搜索页面

driver.get("https://www.so.com/")

#检查系统指定元素是否加载完成

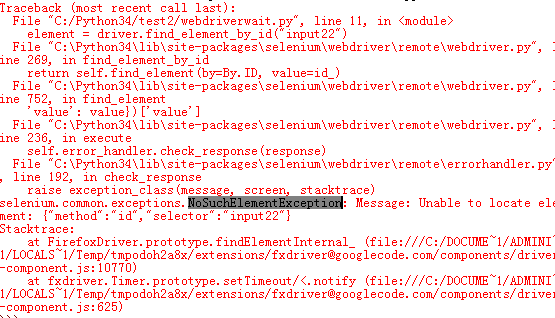
element = driver.find\_element\_by\_id("input22")

#在输入框中输入 selenium

element.send\_keys("selenium")

implicitly\_wait()默认参数的单位为秒，本例中设置等待时长为10秒，首先这10秒并非一个固定的等待时间，它并不影响脚本的执行速度。其次，它并不真对页面上的某一元素进行等待，当脚本执行到某个元素定位时，如果元素可定位那么继续执行，如果元素定位不到，那么它将以轮询的方式不断的判断元素是否被定位到，假设在第6秒钟定位到元素则继续执行。直接超出设置时长（10秒）还没定位到元素则抛出异常。

在上面的例子中，显然360输入框的定位 id= input22 是有误的，那么在超出10秒后将抛出异常。



**7、多窗口切换**

有时候需要在不同的窗口切换，从而操作不同的窗口上的元素。WebDriver 提供了**switch\_to\_window()**方法可以切换到任意的窗口。



以下代码，演示不同窗口切换的方法：

#coding=UTF-8

from selenium import webdriver

from time import \*

dr = webdriver.Firefox()

dr.get("http://www.so.com")

#获得360搜索窗口当前页面的句柄

sreach\_windows = dr.**current\_window\_handle**

print sreach\_windows

#点击登录链接

dr.find\_element\_by\_id("user-login").click()

#点击“忘记密码？”链接，进入找回密码窗口

dr.find\_element\_by\_link\_text("忘记密码？").click()

#获得当前所有打开的窗口的句柄

all\_handles = dr.**window\_handles**

#切换窗口

#1、切换到搜索窗口

dr.**switch\_to\_window**(sreach\_windows)

sleep(1)

#2、关闭 登录 窗口

dr.find\_element\_by\_xpath("/html/body/div[4]/div[1]/a").click()

print "This is search window"

dr.find\_element\_by\_id("input").send\_keys("searhWindows")

sleep(1)

#3、切换到找回密码窗口

for hand in all\_handles:

if hand != sreach\_windows:

dr.switch\_to\_window(hand)

print "This is forget\_windows"

dr.find\_element\_by\_xpath("/html/body/div[3]/div[1]/div[2]/div/form/ul/li[1]/span[2]/input").send\_keys("forget\_windows")

整个脚本的处理过程：首先打开360搜索首页，通过current\_window\_handle 获得当前窗口的句柄，并给变量sreach\_handle。接着打开登录弹窗，在登录窗口上点击“忘记密码？”从而打开找回密码窗口。通过window\_handles 获得当前打开的所窗口的句柄，赋值给变量all\_handles。

for循环遍历all\_handles，如果handle不等于sreach\_windows，那么一定是忘记密码窗口，因为脚本执行只打开的两个窗口。所以，通过switch\_to\_window()切换到忘记密码页面进行操作。

在本例中所有用到的新方法：

**current\_window\_handle** 获得当前窗口句柄

**window\_handles** 返回的所有窗口的句柄到当前会话

**switch\_to\_window()**用于切换到相应的窗口

8、警告框处理

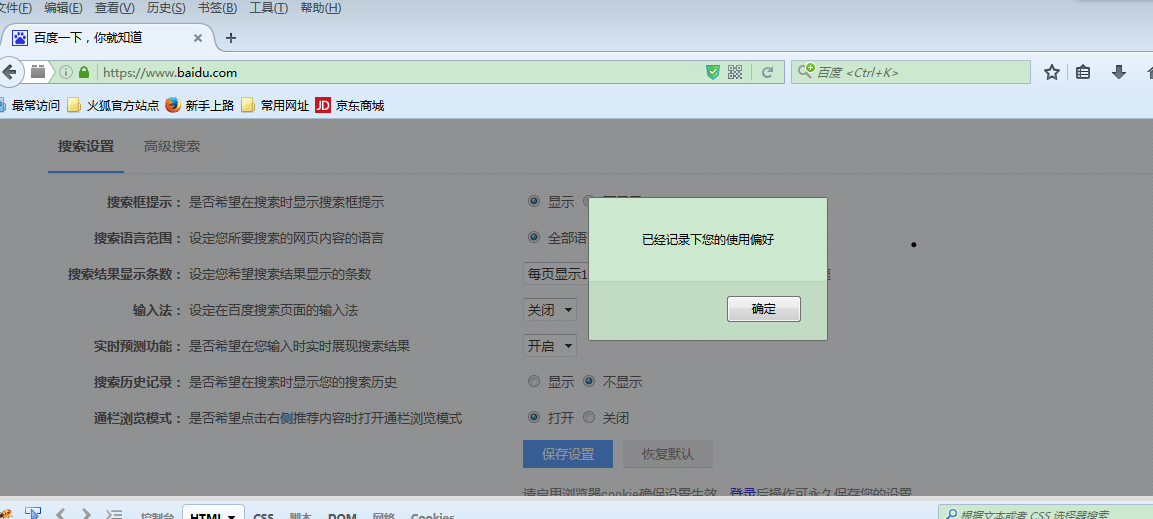
在WebDriver 中处理JavaScript 所生成的网页告警信息可以使用

switch\_to\_alert()方法具体方法如下。该方法，包含三个属性：

 **text 返回告警信息中的文字信息。**

** accept 点击确认按钮。**

** dismiss 点击取消按钮。**



#coding=utf-8

from selenium import webdriver

from selenium.webdriver.common.action\_chains import ActionChains

from time import \*

driver = webdriver.Firefox()

driver.get('http://www.baidu.com')

#鼠标悬停相“设置”链接

driver.find\_element\_by\_link\_text(u'设置').click()

sleep(1)

#打开搜索设置

driver.find\_element\_by\_xpath('/html/body/div[3]/div[5]/a[1]').click()

#保存设置

driver.find\_element\_by\_css\_selector('#gxszButton > a.prefpanelgo').click()

#打印弹窗提示信息

sleep(2)

alartText = driver.switch\_to\_alert().text

print alartText

#接收弹窗

sleep(2)

driver.switch\_to\_alert().accept()

**通过switch\_to\_alert()方法获取当前页上的警告框，accept()接受警告框。**

9、窗口截图

自动化脚本是交给工具去执行，有时候打印的错误信息并不十分明确，如果在脚本执行出错的时候将对当前窗口截图保存，那么通过图片信息会更直观地帮助我们找到脚本出错的原因。Webdriver 提供了截图函数 **get\_screenshot\_as\_file()** 来截取当前窗口。

#coding=UTF-8

from selenium import webdriver

dr = webdriver.Firefox()

dr.get("http://www.so.com")

try:

dr.find\_element\_by\_id("error").send\_keys("selenium")

dr.find\_element\_by\_id("search-button").click()

except:

dr.**get\_screenshot\_as\_file**("D:\\test\\so\_search\_error.jpg")

dr.quit()

在本例中用到了Python 的异常处理。在360搜索页面中找不到id=error的元素，那么try

就会捕捉到这个异常，从而执行except，在except中执行 **get\_screenshot\_as\_file()** 对当前窗口进行截图，这里需要指定图片的保存路径及文件名，并且关闭当前驱动。

10、关闭窗口

在前页的例子中我们一直在使用quit()方法，其含义为退出相关的驱动程序和关闭所有窗口。除此之外WebDriver 还提供了close()方法，用于关闭当前窗口。当脚本在执行时打开了多个窗口，这个时候如果只关闭其中的某一个窗口，这个时候就需要使用close()来关闭。

11、验证码的处理

对于web 应用来说，大部分的系统在用户登录时都要求用户输入验证码，验证码的类型的很多，有字母数字的，有汉字的，甚至还要用户输入一条算术题的答案的，对于系统来说使用验证码可以有效果的防止采用机器猜测方法对口令的刺探，在一定程度上增加了安全性。

但对于测试人员来说，不管是进行性能测试还是自动化测试都是一个比较棘手的问题。在WebDriver中并没有提供相应的方法来处理验证码。这里我根据自己的经验来谈谈处理验证码的几种常见方法。对验证码的常见处理方式有以下几种。

1、去掉验证码

这是最简单的方法，对于开发人员来说，只是把验证码的相关代码注释掉即可，如果是在测试环境，这样做可省去了测试人员不少麻烦，如果自动化脚本是要在正式环境跑，这样就给系统带来了一定的风险。

2、设置万能码

去掉验证码的主要是安全问题，为了应对在线系统的安全性威胁，可以在修改程序时不取消验证码，而是程序中留一个“后门”---设置一个“万能验证码”，只要用户输入这个“万能验证码”，程序就认为验证通过，否则按照原先的验证方式进行验证。设计万能码的方式非常简单，只对于用户的输入信息多加一个逻辑判断。

12、自动化的应用范围和测试用例设计

1）、自动化测试用例的流程不要太长，不要有太多分支。

2）、自动化测试脚本是把手工测试过的用例转换成脚本，但是，不是所有手工用例都能转化成脚本的，比如，UI测试的用例，所以，每个项目都有自己的自动化脚本转换率，我以前做过的项目，自动化脚本转换率要在90%以上。

3）、自动化测试，适用于回归测试。

4）、自动化测试，适用于项目周期长，版本多，界面元素稳定的项目。

5）、自动化测试脚本，要包含四个部分：

5.1）预置条件的构造。比如，你要查询某一天的数据，就要先在环境中构造这些数据；

5.2）操作步骤。

5.3）结果验证。(加入断言 或 验证，loadrunner里叫 检查点)

5、4）环境恢复。把执行这条脚本所用到的数据全部删掉，把环境恢复到初始状态，以免影响后续程序的执行。



**后记**：基于python的selenium自动化测试，我们就讲到这里，但，这不是终点，而是起点，大家还要在以后的工作中，不断深入学习，多总结，多思考。