python webdriver API

1、第三章　python webdriver API

这一章将详细的讲解基于 python 的 webdriver API,笔者更愿意读者自己去查询 webdriver API 中各种操作方法的使用，为了保持本书由浅入深的完整性，本章将用相当有篇幅介绍基于 python 语言的 webdriver 对种操作的使用。通过本章的学习，我们掌握 web 页面上各种元素、弹窗的定位与操作，以及浏览器 cookie 的操作，JavaScript 的调用等问题。

第一节、浏览器的操作

3.1.1、浏览器最大化

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。

```
#coding=utf-8
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://www.baidu.com")

print "浏览器最大化"
driver.maximize_window()  #将浏览器最大化显示
driver.quit()
```

3.1.2、设置浏览器宽、高

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；

```
#coding=utf-8
from selenium import webdriver


driver = webdriver.Firefox()
driver.get("http://m.mail.10086.cn")
#参数数字为像素点
```

```
print "设置浏览器宽480、高800显示"
driver.set_window_size(480, 800)
driver.quit()
```

3.1.3、控制浏览器前进、后退

浏览器上有一个后退、前进按钮，对于浏览网页的人是比较方便的；对于 web 自动化测试来说是一个比较难模拟的操作；webdriver 提供了 back()和 forward()方法，使实现这个操作变得非常简单。

```
#coding=utf-8
from selenium import webdriver
import time

driver = webdriver.Firefox()

#访问百度首页
first_url= 'http://www.baidu.com'
print "now access %s" %(first_url)
driver.get(first_url)

#访问新闻页面
second_url='http://news.baidu.com'
print "now access %s" %(second_url)
driver.get(second_url)

#返回（后退）到百度首页
print "back to  %s "%(first_url)
driver.back()

#前进到新闻页
```

```
print "forward to  %s"%(second_url)

driver.forward()



driver.quit()
```

为了使脚本的执行过程看得更清晰，在每一步操作上都加了 print 来打印当前的 URL 地址。

运行结果如下：

```
>>> ================================ RESTART ================================
>>>
now access http://www.baidu.com
now access http://news.baidu.com
back to  http://www.baidu.com
forward to  http://news.baidu.com
```

在 python 2 中使用 print 语句进行打印输出，如果是字符信息的话需要对打印的信息加单引号（‘’）或双引号（“”），它们本质上没有任何区别，不过必须要成对出现才行。打印语句中指定了输出信息的类型，“%s”表示输出的类型为字符串，“%d”标识输出类型为整型数字。

说实话，这两个功能平时不太常用，所能想到的场景就是几个页面来回跳转，但又不想用 get url 的情况下。

第二节　简单对象的定位

对象（元素）的定位和操作是自动化测试的核心部分，其中操作又是建立在定位的基础上的，因此元素定位就显得非常重要。（本书中用到的对象与元素同为一个事物）

一个对象就像是一个人，他会有各种的特征（属性），如比我们可以通过一个人的身份证号、姓名或者他的住址找到这个人。那么一个元素也有类似的属性，我们可以通过这种唯一区别于其它元素的属性来定位这个元素。当然，除了要操作元素时需要定位元素外，有时候我们只是为了获得元素的属性（class 属性，name 属性）、text 或数量也需要定位元素。

webdriver 提供了一系列的元素定位方法，常用的有以下几种

- id
- name
- class name

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

分别对应 python webdriver 中的方法为：

find_element_by_id()

find_element_by_name()

find_element_by_class_name()

find_element_by_tag_name()

find_element_by_link_text()

find_element_by_partial_link_text()

find_element_by_xpath()

find_element_by_css_selector()

3.2.1 id 和 name 定位

id 和 name 是我们最常用的定位方式，因为大多数元素都有这两个属性，而且在对控件的 id 和 name 命名时一般使其有意义也会取不同的名字。通过这两个属性使我们找一个页面上的属性变得相当容易。

```
<input id="gs_htif0" class="gsfi" aria-hidden="true" dir="ltr">

<input type="submit" name="btnK" jsaction="sf.chk" value="Google 搜索">

<input type="submit" name="btnI" jsaction="sf.lck" value=" 手气不错 ">
```

通过元素中所带的 id 和 name 属性对元素进行定位：

id=”gs_htif0”

find_element_by_id("gs_htif0")

name=”btnK”

find_element_by_name("btnK")

name=”btnI”

find_element_by_name("btnI")

3.2.2 tag name 和 class name 定位

不是所有的前端开发人员都喜欢为每一个元素添加 id 和 name 两个属性，但除此之外你一定发现了一个元素不单单只有 id 和 name，它还有 class 属性；而且每个元素都会有标签。

```
<div id="searchform" class="jhp_big" style="margin-top:-2px">
<form id="tsf"  onsubmit="return  name="f" method="GET" action="/search">
<input id="kw" class="s_ipt" type="text"  name="wd" autocomplete="off">
```

通过元素中带的 class 属性对元素进行定位：

class=”jhp_big”

find_element_by_class_name(″jhp_big″)

class=”s_ipt”

find_element_by_class_name(″s_ipt″)

通过 tag 标签名对对元素进行定位：

find_element_by_tag_name(″div″)

<form>

find_element_by_tag_name(″form″)

<input>

find_element_by_tag_name(″input″)

tag name 定位应该是所有定位方式中最不靠谱的一种了，因为在一个页面中具有相同 tag name 的元素极其容易出现。

3.2.3 link text 与 partial link text 定位

有时候需要操作的元素是一个文字链接，那么我们可以通过 link text 或 partial link text 进行元素定位。

```
<a href="http://news.baidu.com" name="tj_news">新 闻</a>
<a href="http://tieba.baidu.com" name="tj_tieba">贴 吧</a>
<a href="http://zhidao.baidu.com" name="tj_zhidao">一个很长的文字连接</a>
```

通过 link text 定位元素：

find_element_by_link_text("新 闻")

find_element_by_link_text("贴 吧")

find_element_by_link_text("一个很长的文字连接")

通 partial link text 也可以定位到上面几个元素：

find_element_by_partial_link_text("新")

find_element_by_partial_link_text("吧")

find_element_by_partial_link_text("一个很长的")

当一个文字连接很长时，我们可以只取其中的一部分，只要取的部分可以唯一标识元素。一般一个页面上不会出现相同的文件链接，通过文字链接来定位元素也是一种简单有效的定位方式。

下面介绍 xpath 与 CSS 定位相比上面介绍的方式来说比较难理解，但他们的灵活性与定位能力比上面的方式要强大。

3.2.4 XPath 定位

XPath 是一种在 XML 文档中定位元素的语言。因为 HTML 可以看做 XML 的一种实现，所以 selenium 用户可是使用这种强大语言在 web 应用中定位元素。

XPath 扩展了上面 id 和 name 定位方式，提供了很多种可能性，比如定位页面上的第三个多选框。

```
<html class="w3c">
<body>
    <div class="page-wrap">
    <div id="hd" name="q">
        <form target="_self" action="http://www.so.com/s">
        <span id="input-container">
            <input id="input" type="text" x-webkit-speech="" autocomplete="off"
suggestwidth="501px" >
```

我们看到的是一个有层级关系页面，下面我看看如果用 xpath 来定位最后一个元素。

用绝对路径定位:

find_element_by_xpath("/html/body/div[2]/form/span/input")

当我们所要定位的元素很难找到合适的方式时，都可以通这种绝对路径的方式位，缺点是当元素在很多级目录下时，我们不得不要写很长的路径，而且这种方式难以阅读和维护。

相对路径定位：

find_element_by_xpath("//input[@id=' input']") #通过自身的 id 属性定位

find_element_by_xpath("//span[@id=' input-container']/input") #通过上一级目录的 id 属性定位

find_element_by_xpath("//div[@id=' hd']/form/span/input") #通过上三级目录的 id 属性定位

find_element_by_xpath("//div[@name=' q']/form/span/input")#通过上三级目录的 name 属性定位

通过上面的例子，我们可以看到 XPath 的定位方式非常灵活和强大的，而且 XPath 可以做布尔逻辑运算，例如：//div[@id=' hd' or @name=' q']

当然，它的缺陷也非常明显：1、性能差，定位元素的性能要比其它大多数方式差；2、不够健壮，XPath 会随着页面元素布局的改变而改变；3. 兼容性不好，在不同的浏览器下对 XPath 的实现是不一样的。

通过我们第一章中介绍的 firebug 的 HTML 和 firePath 可以非常方便的通过 XPath 方式对页面元素进行定位。

打开 firefox 浏览器的 firebug 插件，点击插件左上角的鼠标箭头，再点击页面上的元素，firebug 插件的 HTML 标签页将看到页面代码，鼠标移动到元素的标签上（如图图3.1，<imput>）将显示当前元素的绝对路径。

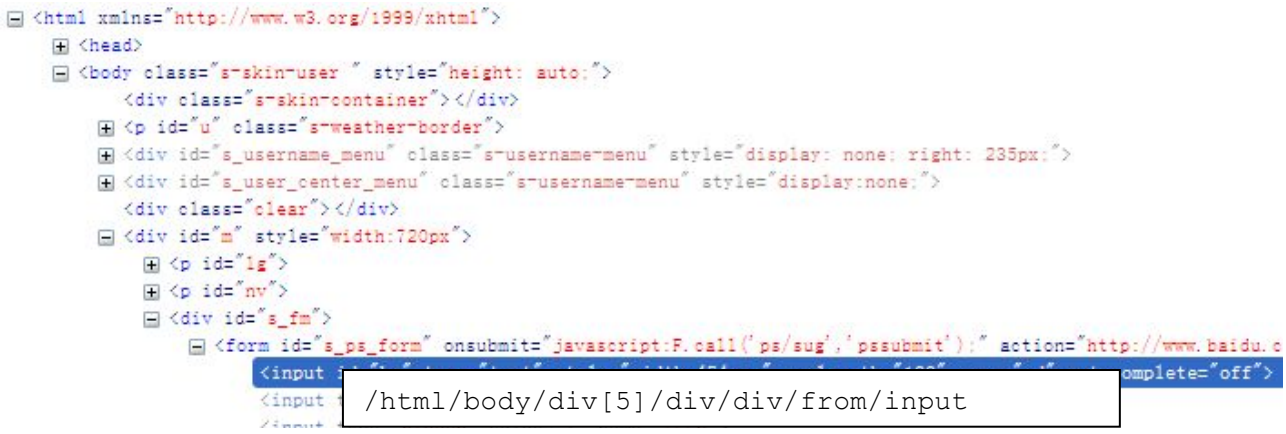

图3.1

或者直接在元素上右击弹出快捷菜单，选择 Copy XPath，将当前元素的 XPath 路径拷贝要脚本本中（如图3.2）。

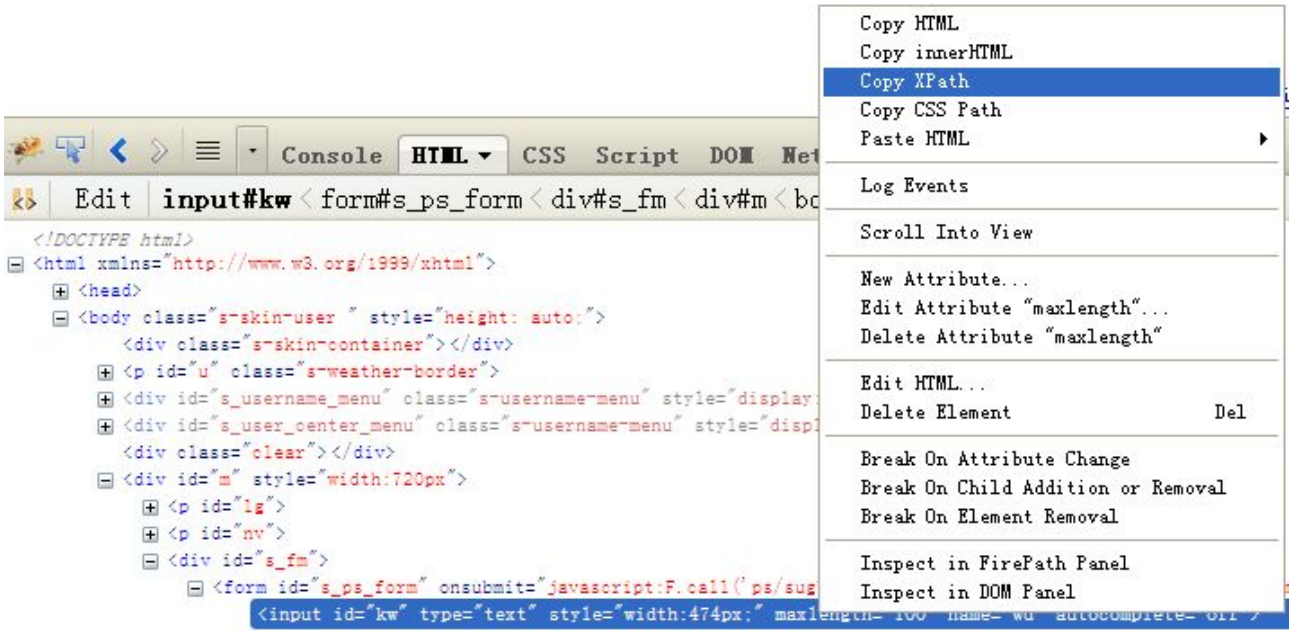

图3.2

firePath 工具的使用就更加方便和快捷了，选中元素后，直接在 XPath 的输入框中显示当前元素的 XPath 的定位信息（如图3.3）。

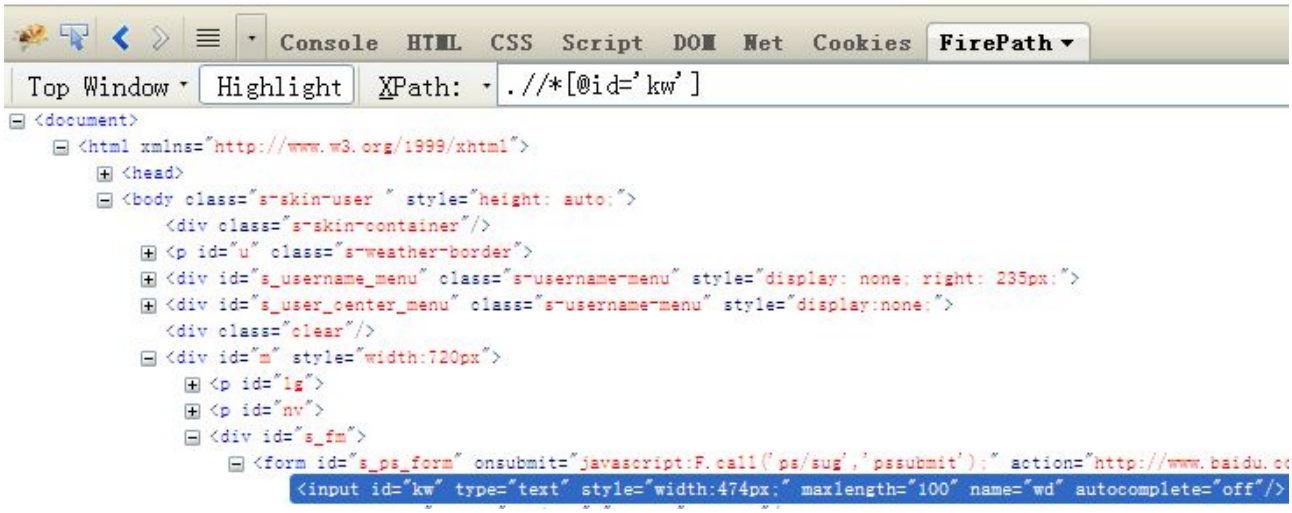

图3.3

3.2.5 CSS 定位

CSS(Cascading Style Sheets)是一种语言，它被用来描述 HTML 和 XML 文档的表现。CSS 使用选择器来为页面元素绑定属性。这些选择器可以被 selenium 用作另外的定位策略。

CSS 可以比较灵活选择控件的任意属性，一般情况下定位速度要比 XPath 快，但对于初学者来说比较难以学习使用，下面我们就详细的介绍 CSS 的语法与使用。

CSS 选择器的常见语法：

*	通用元素选择器，匹配任何元素
E	标签选择器，匹配所有使用 E 标签的元素
.info	class 选择器，匹配所有 class 属性中包含 info 的元素
#footer	id 选择器，匹配所有 id 属性等于 footer 的元素
E,F	多元素选择器，同时匹配所有 E 元素或 F 元素，E 和 F 之间用逗号分隔
E F	后代元素选择器，匹配所有属于 E 元素后代的 F 元素，E 和 F 之间用空格分隔
E > F	子元素选择器，匹配所有 E 元素的子元素 F
E + F	毗邻元素选择器，匹配紧随 E 元素之后的同级元素 F （只匹配第一个）
E ~ F	同级元素选择器，匹配所有在 E 元素之后的同级 F 元素
E[att='val']	属性 att 的值为 val 的 E 元素 （区分大小写）
E[att^='val']	属性 att 的值以 val 开头的 E 元素 （区分大小写）
E[att$='val']	属性 att 的值以 val 结尾的 E 元素 （区分大小写）
E[att*='val']	属性 att 的值包含 val 的 E 元素 （区分大小写）
E[att1='v1'][att2*='v2']	属性 att1 的值为 v1，att2 的值包含 v2 （区分大小写）
E:contains('xxxx')	内容中包含 xxxx 的 E 元素
E:not(s)	匹配不符合当前选择器的任何元素

例如下面一段代码：

```
<div class="formdiv">
<form name="fnfn">
<input name="username" type="text"></input>
<input name="password" type="text"></input>
<input name="continue" type="button"></input>
<input name="cancel" type="button"></input>
<input value="SYS123456" name="vid" type="text">
<input value="ks10cf6d6" name="cid" type="text">
</form>
<div class="subdiv">
<ul id="recordlist">
<p>Heading</p>
<li>Cat</li>
<li>Dog</li>
<li>Car</li>
<li>Goat</li>
</ul>
</div>
</div>
```

通过 CSS 语法进行匹配的实例：

locator	**匹配**
css=div css=div.formdiv	
css=#recordlist css=ul#recordlist	<ul id="recordlist">
css=div.subdiv p css=div.subdiv > ul > p	Heading
css=form + div	
css=p + li css=p ~ li	二者定位到的都是 Cat 但是 storeCssCount 的时候，前者得到 1，后者得到 4
css=form > input[name=username]	<input name="username">
css=input[name$=id][value^=SYS]	<input value="SYS123456" name="vid" type="hidden">
css=input:not([name$=id][value^=SYS])	<input name="username" type="text"></input>
css=li:contains('Goa')	Goat
css=li:not(contains('Goa'))	Cat

css 中的结构性定位

结构性定位就是根据元素的父子、同级中位置来定位，css3标准中有定义一些结构性定位伪类如 nth-of-type，nth-child，但是使用起来语法很不好理解，这里就不做介绍了。

Selenium 中则是采用了来自 Sizzle 的 css3定位扩展，它的语法更加灵活易懂

Sizzle Css3的结构性定位语法：

E:nth(n) E:eq(n)	在其父元素中的 E 子元素集合中排在第 n+1 个的 E 元素 (第一个 n=0)
E:first	在其父元素中的 E 子元素集合中排在第 1 个的 E 元素
E:last	在其父元素中的 E 子元素集合中排在最后 1 个的 E 元素
E:even	在其父元素中的 E 子元素集合中排在偶数位的 E 元素 (0,2,4…)
E:odd	在其父元素中的 E 子元素集合中排在奇数的 E 元素 (1,3,5…)
E:lt(n)	在其父元素中的 E 子元素集合中排在 n 位之前的 E 元素 (n=2,则匹配 0,1)

E:gt(n)	在其父元素中的 E 子元素集合中排在 n 位之后的 E 元素 (n=2，在匹配 3,4)
E:only-child	父元素的唯一一个子元素且标签为 E
E:empty	不包含任何子元素的 E 元素，注意，文本节点也被看作子元素

例如下面一段代码：

```
<div class="subdiv">
    <ul id="recordlist">
        <p>Heading</p>
        <li>Cat</li>
        <li>Dog</li>
        <li>Car</li>
        <li>Goat</li>
    </ul>
</div>
```

匹配示例：

locator	匹配
css=ul > li:nth(0)	Cat
css=ul > *:nth(0) css=ul > :nth(0)	Heading
css=ul > li:first	Cat
css=ul > :first	Heading
css=ul > *:last css=ul > li:last	Goat
css=ul > li:even	Cat, Car
css=ul > li:odd	Dog, Goat
css=ul > :even	Heading
css=ul > p:odd	[error] not found
css=ul > li:lt(2)	Cat
css=ul > li:gt(2)	Goat
css=ul > li:only-child css=ul > :only-child css=ul > *:only-child	[error] not found (ul 没有 only-child)
css=div.subdiv > :only-child	<ul id="recordlist"> … … … …

Sizzle Css3还提供一些直接选取 form 表单元素的伪类：

:input: Finds all input elements (includes textareas, selects, and buttons).

:text, :checkbox, :file, :password, :submit, :image, :reset, :button: Finds the input element with the specified input type (:button also finds button elements).

下面是一些 XPATH 和 CSS 的类似定位功能比较（缺乏一定的严谨性）。

定位方式	XPath	CSS
标签	//div	div
By id	//div[@id='eleid']	div#eleid
By class	//div[@class='eleclass'] //div[contains(@class,'eleclass')]	div.eleclass
By 属性	//div[@title='Move mouse here']	div[title=Move mouse here] div[title^=Move] div[title$=here] div[title*=mouse]
定位子元素	//div[@id='eleid']/* //div/h1	div#eleid >* div#eleid >h1
定位后代元素	//div[@id='eleid']//h1	div h1
By index	//li[6]	li:nth(5)
By content	//a[contains(.,'Issue 1164')]	a:contains(Issue 1164)

通过对比，我们可以看到，CSS 定位语法比 XPath 更为简洁，定位方式更多灵活多样；不过对 CSS 理解起来要比 XPath 较难；但不管是从性能还是定位更复杂的元素上，CSS 优于 XPath，笔者更推荐使用 CSS 定位页面元素。

关于自动化的定位问题

自动化测试的元素定位一直是困扰自动化测试新手的一个障碍，因为我们在自动化实施过程中会碰到各式各样的对象元素。虽然 XPath 和 CSS 可以定位到复杂且比较难定位的元素，但相比较用 id 和 name 来说增加了维护成本和学习成本，相比较来说 id/name 的定位方式更直观和可维护，有新的成员加入的自动化时也增加了人员的学习成本。所以，测试人员在实施自动化测试时一定要做好沟通，规范前端开发人员对元素添加 id/name 属性，或者自己有修改 HTML 代码的权限。

第三节 操作测试对象

前面讲到了不少知识都是定位对象，定位只是第一步，定位之后需要对这个对象进行操作。鼠标点击呢还是键盘输入，这要取决于我们定位的对象所支持的操作。

一般来说，所有有趣的操作与页面交互都将通过 WebElement 接口，包括上一节中介绍的对象定位，以及本节中需要介绍的常对象操作。

webdriver 中比较常用的操作元素的方法有下面几个：

- clear　　清除元素的内容，如果可以的话

- send_keys　在元素上模拟按键输入
- click　单击元素
- submit　提交表单

3.3.1、登录实例

下面以快播私有云登录实例来展示常见元素操作的使用：

```
#coding=utf-8
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://passport.kuaibo.com/login/?referrer=http%3A%2F%2Fwebcloud.kuaibo.com%2F")

driver.find_element_by_id("user_name").clear()
driver.find_element_by_id("user_name").send_keys("username")
driver.find_element_by_id("user_pwd").clear()
driver.find_element_by_id("user_pwd").send_keys("password")
driver.find_element_by_id("dl_an_submit").click()
#通过 submit() 来提交操作
#driver.find_element_by_id("dl_an_submit").submit()

driver.quit()
```

clear()　用于清除输入框的默认内容

比如登录框内一般默认会有“账号”“密码”等提示信息，如果直接输入内容，和可能会与输入框的默认提示信息拼接，从而造成输入信息的错误；这时 clear（）将变得非常有用。

send_keys("xx")　用于在一个输入框里输入 xx 内容

python 是个容易出现编码问题的语言，有时候当我们在 send_keys()方法中输入中文时，然后脚本在运行时就报编码错误,这个时候我们可以在脚本开头声明编码为 utf-8,然后在中文字符的前面加个小 u 就解决了（表示转成 python Unicode 编码）：

#coding=utf-8

send_keys(u"中文内容")

需要注意的是 utf-8并不是万能的，如果 utf-8不能解决，可以尝试将编码声明为 GBK；关于 python 的编码问题，请参考 python 相关书籍。

click()　用于单击一个按钮

其实 click()方法不仅仅用于点击一个按钮，可以单击任何可以点击的元素，文字/图片连接，按钮，下拉按钮等。

submit()　　提交表单

从上面有例子，我们可看到可以使用 submit()方法来代替 click()对输入的信息进行提交，在有些情况下两个方法可以相互使用；submit()要求提交对象是一个表单，更强调对信息的提交。click()更强调事件的独立性。（比如，一个文字链接就不能用 submit()方法。）

3.3.2 WebElement 接口常用方法

WebElement 接口除了我们前面介绍的方法外，它还包含了别一些有用的方法。下面，我们例举例几个比较有用的方法。

size

返回元素的尺寸。例：

```
#返回百度输入狂的宽高
size=driver.find_element_by_id("kw").size
print size
```

text

获取元素的文本，例：

```
#返回百度页面底部备案信息
text=driver.find_element_by_id("cp").text
print text
```

get_attribute(name)

获得属性值。例：

```
#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("kw").get_attribute('type')
print attribute
```

需要说明的是这个方法在定位一组时将变得非常有用，后面将有运行的实例。

is_displayed()

设置该元素是否用户可见。例：

```
#返回元素的结果是否可见，返回结果为 True 或 False
result=driver.find_element_by_id("kw").is_displayed()
print result
```

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

第四节　鼠标事件

前面例子中我们已经学习到可以用 click()来模拟鼠标的单击操作，而我们在实际的 web 产品测试中发现，有关鼠标的操作，不单单只有单击，有时候还要和到右击，双击，拖动等操作，这些操作包含在 ActionChains 类中。

ActionChains 类鼠标操作的常用方法：

- context_click()　右击
- double_click()　双击
- drag_and_drop()　拖动
- move_to_element()　鼠标悬停在一个元素上
- click_and_hold()　按下鼠标左键在一个元素上

鼠标右击操作

context_click()

右键点击一个元素。

图3.x

如图3.x ，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()
....
```

from selenium.webdriver.common.action_chains import ActionChains

这里需要注意的是，在使用 ActionChains 类下面的方法之前，要先将包引入。

ActionChains(driver)

driver: wedriver 实例执行用户操作。

ActionChains 用于生成用户的行为；所有的行为都存储在 actionchains 对象。通过 perform()执行存储的行为。

perform()

执行所有 ActionChains 中存储的行为。perfrome()同样也是 ActionChains 类提供的的方法，通常与 ActionChains（）配合使用。

除了鼠标右击之外，ActionChains 类还提供了其它的比较复杂的鼠标方法。

鼠标双击操作

double_click(on_element)

双点击页面元素。例：

```
#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()
```

对于操作系统的操作来说，双击使用相当频繁，但对于 web 应用来说双击的用户比较少，笔者唯一能想到想的场景是地图程序（如 百度地图），可以通过双击鼠标放大地图。

鼠标拖放操作

drag_and_drop(source, target)

在源元素上按下鼠标左键，然后移动到目标元素上释放。

source: 鼠标按下的源元素。

target: 鼠标释放的目标元素。

鼠标拖放操作的参考代码如下：

```
#引入 ActionChains 类
```

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

...

#定位元素的原位置

element = driver.find_element_by_name("xxx")

#定位元素要移动到的目标位置

target =  driver.find_element_by_name("xxx")



#执行元素的移动操作

ActionChains(driver).drag_and_drop(element, target).perform()
```

move_to_element()
模拟鼠标移动到一个元素上。

click_and_hold ()
按住鼠标左键在一个元素。

第五节　键盘事件

我们在实际的测试工作中，有时候我们在测试时需要使用 tab 键将焦点转移到下一个元素，用于验证元素的排序是否正确。webdriver 的 Keys()类提供键盘上所有按键的操作，甚至可以模拟一些组合建的操作，如 Ctrl+A ,Ctrl+C/Ctrl+V 等。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。

```
#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')
time.sleep(3)

#ctrl+x 剪切输入框内容
driver.find_element_by_id("kw").send_keys(Keys.CONTROL,'x')
time.sleep(3)

#输入框重新输入内容，搜索
driver.find_element_by_id("kw").send_keys(Keys.CONTROL,'v')
time.sleep(3)

#通过回车键盘来代替点击操作
driver.find_element_by_id("su").send_keys(Keys.ENTER)
time.sleep(3)

driver.quit()
```

需要说明的是上面脚本没什么实际意义，但向我们展示了组合键及键盘按键的用法。为了使用脚本的的运行过程更我们可以看得更加清晰，在每一步操作之后都加上了三秒的休眠时间 time.sleep()，后面会再介绍 time.sleep()方法的使用。

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

在使用键盘按键方法前需要先导入 keys 类包。

下面经常使用到的键盘操作：

```
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）
send_keys(Keys.CONTROL,'x') 剪切（Ctrl+X）
send_keys(Keys.CONTROL,'v') 粘贴（Ctrl+V）
```

Keys 类所提供的按键请查阅 webdriver API.

第六节　打印信息

当我们要设计功能测试用例时，一般会有预期结果，有些预期结果是由测试人员通过肉眼进行判断的。因为自动化测试运行过程是无人值守，一般情况下，脚本运行成功，没有异样信息就标识用户执行成功。当然，这还不走在足够去正明一个用例确实是执行成功的。所以我们需要获得更多的信息来证明用例执行结果确实是成功的。

通常我们可以通过获得页面的 title 、URL 地址，页面上的标识性信息（如，登录成功的“欢迎，xxx”信息）来判断用例执行成功。

在实际测试中，访问 1 个页面然判断其 title 是否符合预期是很常见的一个用例，假如一个页面的 title 应该是“快播私有云”，那么用例可以这样描述：访问该页面，判断页面 title 是否等于“快播私有云”。

获取当前 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。如果页面的 ULR 返回不正确，而表示当前操作没有进行正常的跳转。

下面通过快播私有云登录实例进行讲解：

```
#coding=utf-8
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://passport.kuaibo.com/login/?referrer=http%3A%2F%2Fwebcloud.kuaibo.com%2
F")
#登录
driver.find_element_by_id("user_name").clear()
driver.find_element_by_id("user_name").send_keys("username")
driver.find_element_by_id("user_pwd").clear()
driver.find_element_by_id("user_pwd").send_keys("password")
driver.find_element_by_id("dl_an_submit").click()

#获得前面 title，打印
title = driver.title
print title

#拿当前 URL 与预期 URL 做比较
if title == u"快播私有云":
    print "title ok!"
else:
```

```
    print "title on!"


#获得前面 URL，打印
now_url = driver.current_url
print now_url
#拿当前 URL 与预期 URL 做比较
if now_url == "http://webcloud.kuaibo.com/":
    print "url ok!"
else:
    print "url on!"

#获得登录成功的用户，打印
now_user=driver.find_element_by_xpath("//div[@id='Nav']/ul/li[4]/a[1]/span").text
print now_user

driver.quit()
```

运行结果：

```
>>> ================================ RESTART ================================

>>>

快播私有云

title ok!

http://webcloud.kuaibo.com/

url ok！

虫师
```

本例中涉及到新的方法如下：

title

返回当前页面的标题

current_url

获取当前加载页面的 URL

在上面的例子中我们用到了 python 的 if 判断语句，与其它语言没有差异，python 的 if 语句块用冒号（：）表示后面需要执行的语句。

第七节　设置等待时间

有时候为了保证脚本运行的稳定性，需要脚本中添加等待时间。

sleep()：设置固定休眠时间。python 的 time 包提供了休眠方法 sleep() ，导入 time 包后就可以使用 sleep() 进行脚本的执行过程进行休眠。

implicitly_wait()：是 webdirver 提供的一个超时等待。隐的等待一个元素被发现，或一个命令完成。如果超出了设置时间的则抛出异常。

WebDriverWait()：同样也是 webdirver 提供的方法。在设置时间内，默认每隔一段时间检测一次当前页面元素是否存在，如果超过设置时间检测不到则抛出异常。

下面通过实例来展示方法的具体使用：

```
#coding=utf-8
from selenium import webdriver
#导入 WebDriverWait 包
from selenium.webdriver.support.ui import WebDriverWait
#导入 time 包
import time

driver = webdriver.Firefox()
driver.get("http://www.baidu.com")

#WebDriverWait()方法使用
element=WebDriverWait(driver, 10).until(lambda driver : driver.find_element_by_id("kw"))
element.send_keys("selenium")

#添加智能等待
driver.implicitly_wait(30)
driver.find_element_by_id("su").click()

#添加固定休眠时间
time.sleep(5)

driver.quit()
```

sleep()

sleep()方法以秒为单位，假如休眠时间小时 1 秒，可以用小数表示。

```
import time
....
time.sleep(5)
time.sleep(0.5)
```

当然，也可以直接导入 sleep()方法，使脚本中的引用更简单

```
from time import sleep
....
sleep(3)
sleep(30)
```

implicitly_wait()

implicitly_wait()方法比 sleep() 更加智能，后者只能选择一个固定的时间的等待，前者可以在一个时间范围内智能的等待。

WebDriverWait()

详细格式如下：

WebDriverWait(driver, timeout, poll_frequency=0.5, ignored_exceptions=None)

driver - WebDriver 的驱动程序(Ie, Firefox, Chrome 或远程)

timeout - 最长超时时间，默认以秒为单位

poll_frequency - 休眠时间的间隔（步长）时间，默认为 0.5 秒

ignored_exceptions - 超时后的异常信息，默认情况下抛 NoSuchElementException 异常。

实例：

```
from selenium.webdriver.support.ui import WebDriverWait
....
element = WebDriverWait(driver, 10).until(lambda x: x.find_element_by_id("someId"))
is_disappeared = WebDriverWait(driver, 30, 1, (ElementNotVisibleException)).
           until_not(lambda x: x.find_element_by_id("someId").is_displayed())
```

WebDriverWai()一般由 unit()或 until_not()方法配合使用，下面是 unit()和 until_not()方法的说明。

until(method, message='')

调用该方法提供的驱动程序作为一个参数，直到返回值不为 False。

until_not(method, message='')

调用该方法提供的驱动程序作为一个参数，直到返回值为 False。

第八节 定位一组对象

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，WebElement 接口同样提供了定位一组元素的方法 find_elements。

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。

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>
            </form>
        </div>
    </body>
</html>
```

将这段代码保存复制到记事本中，将保存成 checkbox.html 文件。（注意，这个页面需要和我们的自动化脚本放在同一个目录下，否则下面的脚本将指定 checkbox.html 的所在目录）

通过浏览器打开 checkbox.html，将看到以下页面：

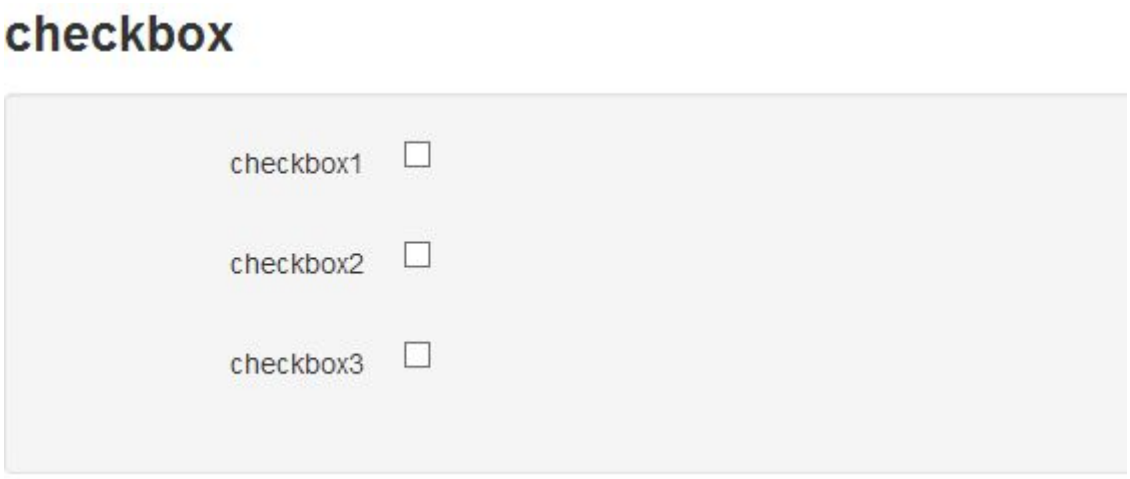

图 3.4

通过图 3.4 可以看到页面提供了三个复选框和两个单选按钮。下面通过脚本来单击勾选三个复选框。

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

driver = webdriver.Firefox()
file_path =  'file:///' + os.path.abspath('checkbox.html')
driver.get(file_path)

# 选择页面上所有的 tag name 为 input 的元素
inputs = driver.find_elements_by_tag_name('input')
#然后从中过滤出 tpye 为 checkbox 的元素，单击勾选
for input in inputs:
    if input.get_attribute('type') == 'checkbox':
        input.click()

driver.quit()
```

`import os`

`os.path.abspath()`

`os` 模块为 python 语言标准库中的 `os` 模块包含普遍的操作系统功能。主要用于操作本地目录文件。`path.abspath()` 方法用于获取当前路径下的文件。另外脚本中还使用到 `for` 循环，对 `inputs` 获取的一组元素进行循环，在 python 语言中循环变量（input）可以不用事先声明直接使用。

`find_elements_by_xx('xx')`

`find_elements` 用于获取一组元素。

下面通过 css 方式来勾选一组元素，打印当所勾选元素的个数并对最后一个勾选的元素取消勾选。

```
#coding=utf-8
from selenium import webdriver
import os

driveriver = webdriver.Firefox()
file_path =  'file:///' + os.path.abspath('checkbox.html')
driver.get(file_path)

# 选择所有的 type 为 checkbox 的元素并单击勾选
checkboxes = driver.find_elements_by_css_selector('input[type=checkbox]')
for checkbox in checkboxes:
    checkbox.click()

# 打印当前页面上 type 为 checkbox 的个数
print len(driver.find_elements_by_css_selector('input[type=checkbox]'))

# 把页面上最后1个 checkbox 的勾给去掉
driver.find_elements_by_css_selector('input[type=checkbox]').pop().click(
)

driver.quit()
```

len()

len 为 python 语言中的方法，用于返回一个对象的长度（或个数）。

pop()

pop 也为 python 语言中提供的方法，用于删除指定们位置的元素，pop()为空默认选择最一个元素。

第九节　层级定位

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。

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="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" />
</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/bootstrap.min.js"></script>
</html>
```

将上面的代码保存为 level_locate.html，通过浏览器打开将看到以下页面。

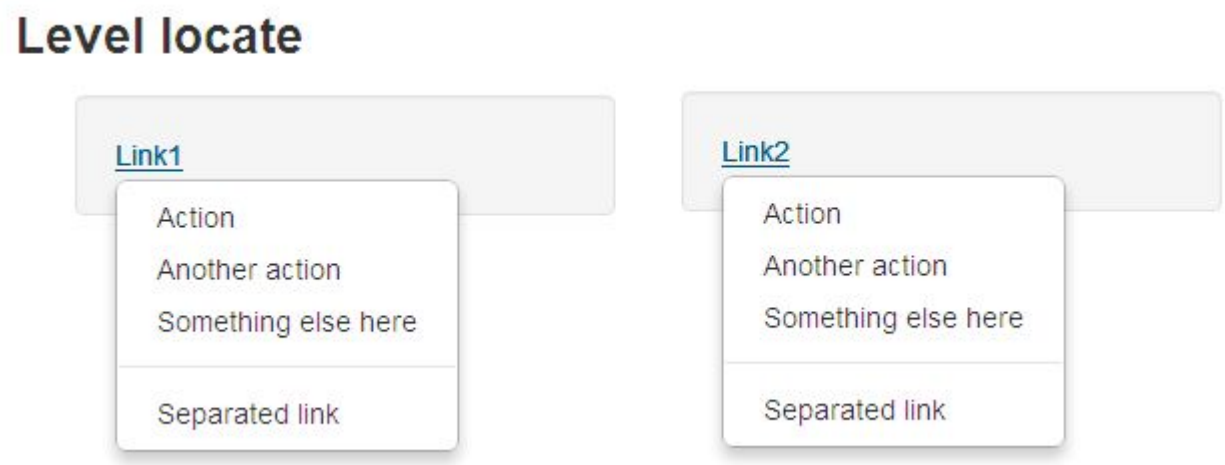

图 3.5

通过对上面代码的分析，发现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。

```
#coding=utf-8
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.action_chains import ActionChains
```

```
import time

import os


driver = webdriver.Firefox()

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

driver.get(file_path)


#点击 Link1 链接（弹出下拉列表）

driver.find_element_by_link_text('Link1').click()


#在父亲元件下找到 link 为 Action 的子元素

menu =

driver.find_element_by_id('dropdown1').find_element_by_link_text('Another

action')


#鼠标移动到子元素上

ActionChains(driver).move_to_element(menu).perform()

time.sleep(5)

driver.quit()
```

driver.find_element_by_id('xx').find_element_by_link_text('xx').click()

这里用到了二次定位，通过对 Link1的单击之后，出现下拉菜单，先定位到下拉菜单，再定位下拉菜单中的选项。当然，如果菜单选项需要单击，可通过二次定位后也直接跟 click()操作。

ActionChains(driver)

driver: wedriver 实例执行用户操作。

ActionChains 用于生成用户的行为；所有的行为都存储在 actionchains 对象。通过 perform()执行存储的行为。

move_to_element(menu)

move_to_element 方法模式鼠标移动到一个元素上，上面的例子中 menu 已经定义了他所指向的是哪一个元素。

perform()

执行所有 ActionChains 中存储的行为。

定位后的效果如下，鼠标点击 Link1 菜单，鼠标移动下 Another action 选项上，选项出现选中色。

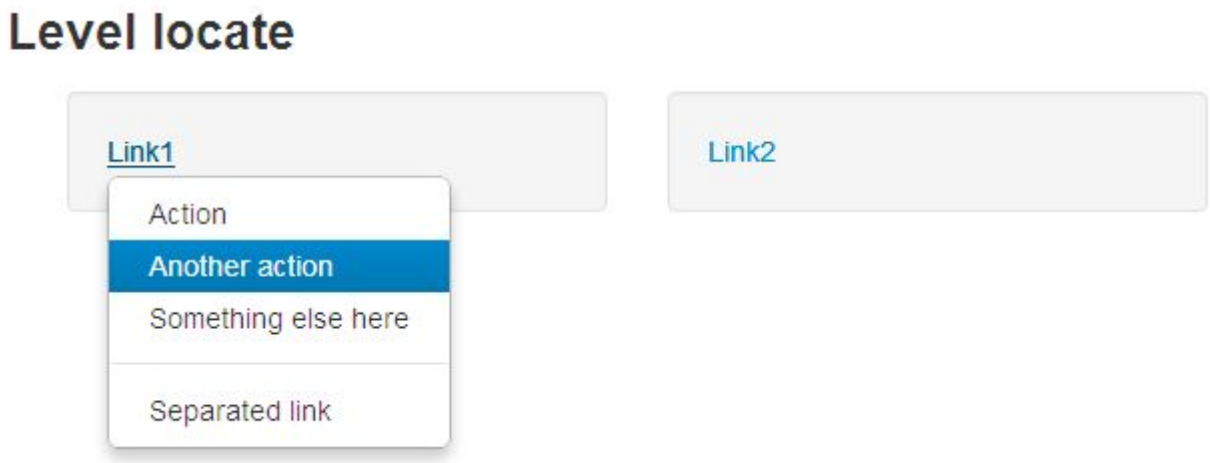

图 3.6

第十节　定位 frame 中的对象

在 web 应用中经常会出现 frame 嵌套的应用，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。

switch_to_frame 方法可以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。

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/bootstrap-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/bootstrap.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="400">
            </iframe>
        </div>
    </div>
</body>
</html>
```

frame.html 中嵌套 inner.html ，两个文件和我们的脚本文件放同一个目录下，通过浏览器打开，得到下列页面：

图 3.7

下面通过 switch_to_frame 方法来定位 frame 内的元素：

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

driver = webdriver.Firefox()
file_path =  'file:///' + os.path.abspath('frame.html')
driver.get(file_path)

driver.implicitly_wait(30)
#先找到到 ifrome1（id = f1）
driver.switch_to_frame("f1")
#再找到其下面的 ifrome2(id =f2)
driver.switch_to_frame("f2")

#下面就可以正常的操作元素了
driver.find_element_by_id("kw").send_keys("selenium")
driver.find_element_by_id("su").click()
```

```
time.sleep(3)
driver.quit()
```

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，你可以直接手动添加。

第十一节　对话框处理

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；很多情况下对话框是一个 iframe，如上一节中介绍的例子，处理起来稍微有点麻烦；但现在很多前端框架的对话框是 div 形式的，这就让我们的处理变得十分简单。

图 3.x

图 3.x 为百度首页的登录对话框，下面通过脚本对百度进行登录操作：

```
#coding=utf-8
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://www.baidu.com/")

#点击登录链接
driver.find_element_by_name("tj_login").click()
```

```
#通过二次定位找到用户名输入框
div=driver.find_element_by_class_name("tang-content").find_element_by_name("userName")
div.send_keys("username")
#输入登录密码
driver.find_element_by_name("password").send_keys("password")

#点击登录
driver.find_element_by_id("TANGRAM__PSP_10__submit").click()

driver.quit()
```

本例中并没有用到新方法，唯一的技巧是用到了二次定位，这个技巧在层级定位中已经有过使用。

driver.find_element_by_class_name("tang-content").find_element_by_name("userName")

第一次定位找到弹出的登录框，在登录框上再次进行定位找到了用户名输入框。

第十二节　浏览器多窗口处理

有时候我们在测试一个 web 应用时会出现多个浏览器窗口的情况，在 selenium1.0 中这个问题比较难处理。webdriver 提供了相关相方法可以很轻松的在多个窗口之间切换并操作不同窗口上的元素。

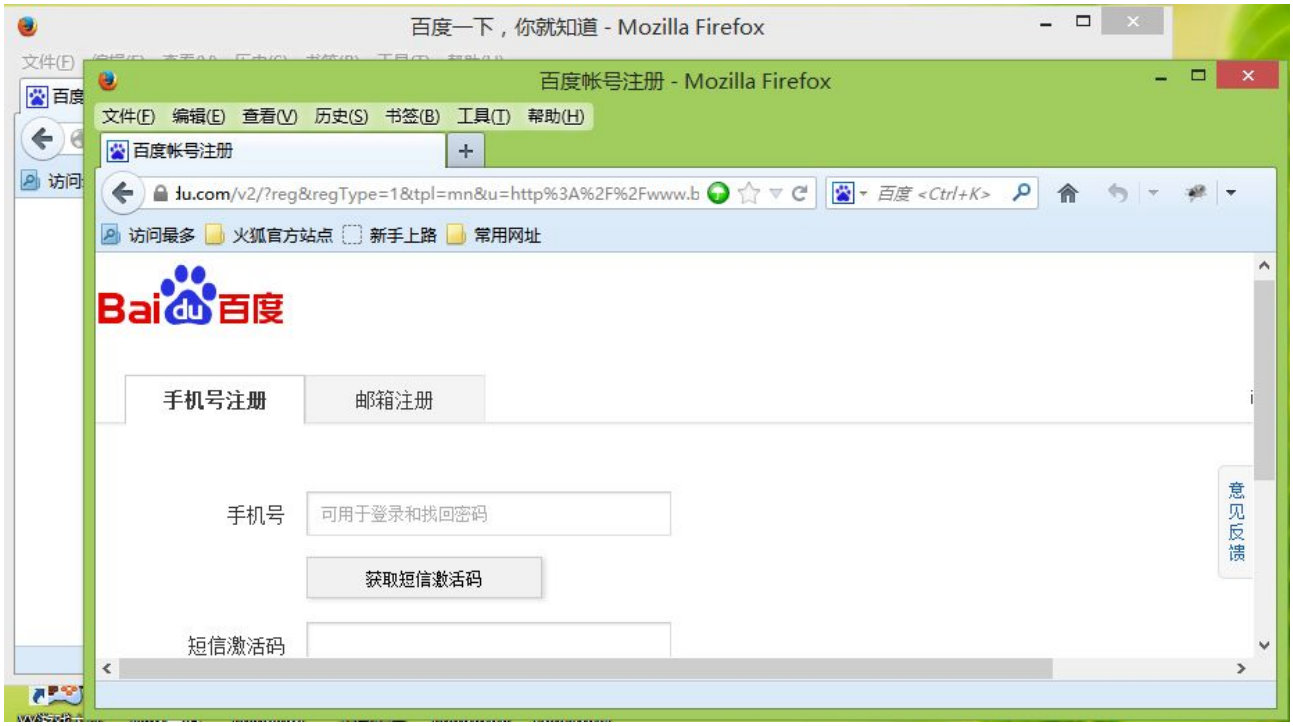

图 3.x

要想在多个窗口之间切换，首先要获得每一个窗口的唯一标识符号（句柄）。通过获得的句柄来区别分不同的窗口，从而对不同窗口上的元素进行操作。

```
#coding=utf-8
from selenium import webdriver
import time

driver = webdriver.Firefox()
driver.get("http://www.baidu.com/")

#获得当前窗口
nowhandle=driver.current_window_handle

#打开注册新窗口
driver.find_element_by_name("tj_reg").click()

#获得所有窗口
allhandles=driver.window_handles

#循环判断窗口是否为当前窗口
for handle in allhandles:
    if handle != nowhandle:
        driver.switch_to_window(handle)
        print 'now register window!'
        #切换到邮箱注册标签
        driver.find_element_by_id("mailRegTab").click()
        time.sleep(5)
        driver.close()

#回到原先的窗口
driver.switch_to_window(nowhandle)

driver.find_element_by_id("kw").send_keys(u"注册成功！")
time.sleep(3)
driver.quit()
```

处理过程：

这个处理过程相比我们前面的元素操作来说稍微复杂一些，执行过程为：首选通过 nowhandle 获得当前窗口（百度首页）的句柄；然后，打开注册窗口（注册页）；通过 allhandles 获得所有窗口的句柄；对所有句柄进行循环遍历；判断窗口是否为 nowhandle（百度首页），如果不是则获得当前窗口（注册页）的句柄；然后，对注册页上的元素进行操作。最后，回返到首页。

为了使执行过程更多更容易理解，在切换到注册页时，打印了'now register window!'一条信息；切换回百度首页时，我们在输入框输入了“注册成功!”。注意，我们在切换到注册页时，只是切换了一下邮箱注册标签，如果要直执行注册过程还需要添加更多的操作步骤。

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()

用于处理多窗口操作的方法，与我们前面学过的 switch_to_frame() 是类似，switch_to_window()

用于处理多窗口之前切换，switch_to_frame() 用于处理多框架的切换。

close()

如果你足够细心会发现我们在关闭“注册页”时用的是 close()方法，而非 quit()；close()用于关闭当前窗口，quit()用于退出驱动程序并关闭所有相关窗口。

第十二节 alert/confirm/prompt 处理

webdriver 中处理 JavaScript 所生成的 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 输入值，这个 alert\confirm 没有对话框就不能用了，不然会报错。

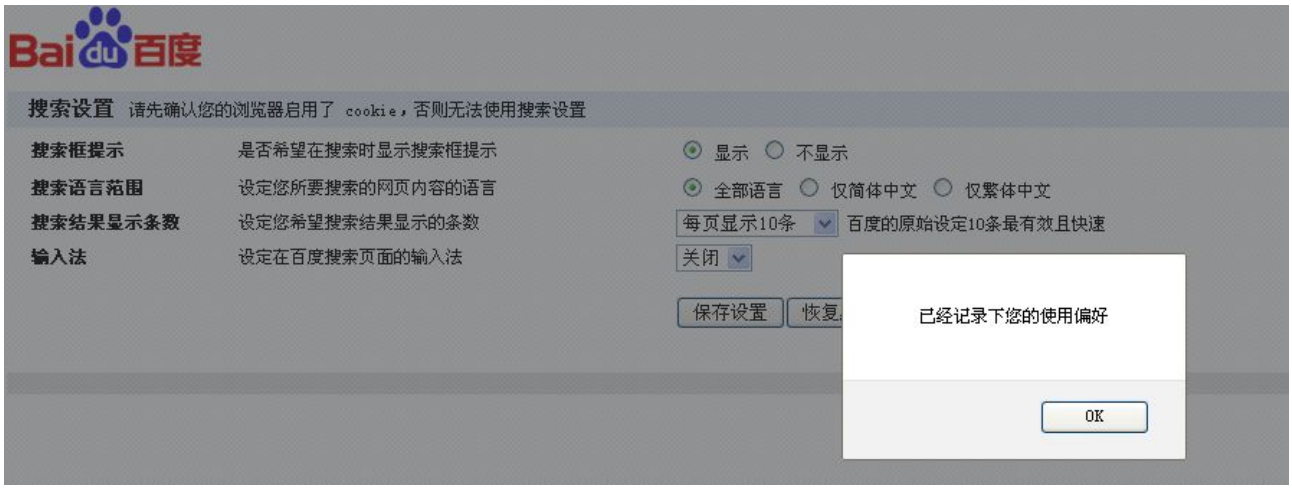

图 3.x

图 3.x 所给出的是百度设置页面，在设置完成后点击“保存设置”所弹的提示框。下面通过脚本来处理这个弹窗。

```
#coding=utf-8
from selenium import webdriver
import time

driver = webdriver.Firefox()
driver.get("http://www.baidu.com/")
#点击打开搜索设置
driver.find_element_by_name("tj_setting").click()
driver.find_element_by_id("SL_1").click()
#点击保存设置
driver.find_element_by_xpath("//div[@id='gxszButton']/input").click()

#获取网页上的警告信息
alert=driver.switch_to_alert()

#接收警告信息
alert.accept()

dirver.quit()
```

```
switch_to_alert()
```

用于获取网页上的警告信息。我们可以对警告信息做以下操作：

```
#接受警告信息
alert = driver.switch_to_alert()
```

```
alert.accept()

#得到文本信息并打印
alert = driver.switch_to_alert()
print  alert.text()

#取消对话框（如果有的话）
alert = driver.switch_to_alert()
alert.dismiss()

#输入值（如果有的话）
alert = driver.switch_to_alert()
alert.send_keys("xxx")
```

第十三节　下拉框处理

下拉框也是 web 页面上非常常见的功能，webdriver 对于一般的下拉框处理起来也相当简单，要想定位下拉框中的内容，首先需要定位到下拉框；这样的二次定位，我们在前面的例子中已经有过使用，下面通过一个具体的例子来说明具体定位方法。

drop_down.html

```
<html>
   <body>
   <select    id="ShippingMethod"    onchange="updateShipping(options[selectedIndex]);"
   name="ShippingMethod">
       <option value="12.51">UPS Next Day Air ==> $12.51</option>
       <option value="11.61">UPS Next Day Air Saver ==> $11.61</option>
       <option value="10.69">UPS 3 Day Select ==> $10.69</option>
       <option value="9.03">UPS 2nd Day Air ==> $9.03</option>
       <option value="8.34">UPS Ground ==> $8.34</option>
       <option value="9.25">USPS Priority Mail Insured ==> $9.25</option>
       <option value="7.45">USPS Priority Mail ==> $7.45</option>
       <option value="3.20" selected="">USPS First Class ==> $3.20</option>
   </select>
    </body>
</html>
```

保存并通过浏览器打开，如下：

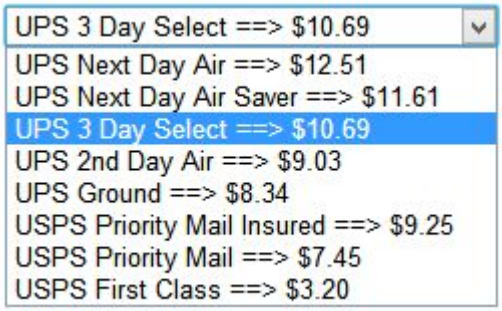

图3.x

图 3.x 是最常现在我们来通过脚本选择下拉列表里的$10.69

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


driver= webdriver.Firefox()
file_path =  'file:///' + os.path.abspath('drop_down.html')
driver.get(file_path)
time.sleep(2)


#先定位到下拉框
m=driver.find_element_by_id("ShippingMethod")
#再点击下拉框下的选项
m.find_element_by_xpath("//option[@value='10.69']").click()
time.sleep(3)


driver.quit()
```

需要说明的是在实际的 web 测试时，会发现各种类型的下拉框，并非我们我们上面所介绍的传统的下拉框。如图 3.x ，对这种类型的下拉框一般的处理是两次点击，第一点击弹出下拉框，第二次点击操作元素。当然，也有些下拉框是鼠标移上去直接弹出的，那么我们可以使用 move_to_element()进行操作。

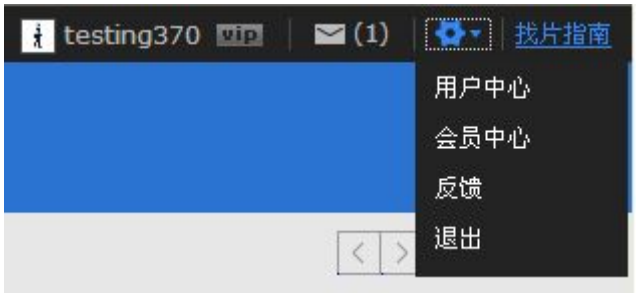

图 3.x

第十四节　分页处理

对于 web 页面上的分页功能，我们一般做做以下操作：

- 获取总页数
- 翻页操作（上一页，下一页）

对于有些分页功能提供上一页，下一页按钮，以及可以输入具体页面数跳转功能不在本例的讨论范围。

```
....
<select id="pageElm_a74e_ce2c" class="yem" action="page" data-page="5">
    <option value="1">1/5</option>
    <option value="2">2/5</option>
    <option value="3">3/5</option>
    <option value="4">4/5</option>
    <option value="5">5/5</option>
</select>
....
```

上面代码为分页功能的代码片断，显示效果如下：

图 3.x

```
#coding=utf-8
from selenium import webdriver
from time import sleep
```

```
driver = webdriver.Firefox()
driver.get("http://passport.kuaibo.com/login/?referrer=http%3A%2F%2Fvod.kuaibo.com%2F%3Fly%3Ddefault")
#登录系统
driver.find_element_by_id("user_name").clear()
driver.find_element_by_id("user_name").send_keys("username")
driver.find_element_by_id("user_pwd").clear()
driver.find_element_by_id("user_pwd").send_keys("password")
driver.find_element_by_id("dl_an_submit").click()
sleep(2)

#获取所有分页的数量，并打印
total_pages=len(driver.find_element_by_tag_name("select").find_elements_by_tag_name("option"))
print "total page is %s" %(total_pages)
sleep(3)

#再次获取所分页，并执行循环翻页操作
pages=driver.find_element_by_tag_name("select").find_elements_by_tag_name("option")
for page in pages:
    page.click()
    sleep(2)

sleep(3)
driver.quit()
```

len()方法在定位一组对象有时已经用过，用于获取对象的个数。

这里同样用到了二次定位，只是第二次定位用的是 find_elements 方法，获取的是一组元素。通过上面的脚本可以看到，我们第一次获取到一组元素后，打印了所有分页的个数。第二次获取所有分页后， 通过 for 循环来翻阅每一页，每翻一页休眠 2 秒，当然，我们也可以在翻页后对列表的文件做更多操作。

第十五节　上传文件

文件上传操作也比较常见功能之一，上传功能操作 webdriver 并没有提供对应的方法，关键上传文件的思路。

上传过程一般要打开一个系统的 window 窗口，从窗口选择本地文件添加。所以，一般会卡在如何操作本地 window 窗口。其实，上传本地文件没我们想的那么复杂；只要定位上传按钮，通 send_keys 添加本地文件路径就可以了。绝对路径和相对路径都可以，关键是上传的文件存在。下面通地例子演示操作过程。

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/bootstrap-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/bootstrap.min.js"></script>
</html>
```

通过浏览器打开，得到下列页面：

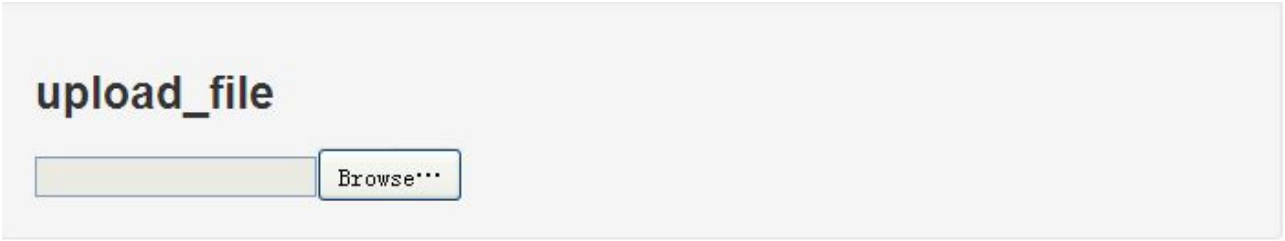

图 3.x

操作上传脚本：

```
#coding=utf-8
from selenium import webdriver
import os,time
```

```
driver = webdriver.Firefox()

#打开上传文件页面
file_path =  'file:///' + os.path.abspath('upload_file.html')
driver.get(file_path)

#定位上传按钮，添加本地文件
driver.find_element_by_name("file").send_keys('D:\\selenium_use_case\upload_file.txt')
time.sleep(2)

driver.quit()
```

从上面例子可以看到，send_keys()方法除可以输入内容外，也可以跟一个本地的文件路径。从而达到上传文件的目的。

文件上传成功的效果如图 3.x：

图 3.x

第十六节　下载文件

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。

要想下载文件，首选要先确定你所要下载的文件的类型。要识别自动文件的下载类型可以使用 curl ，如图 3.x：

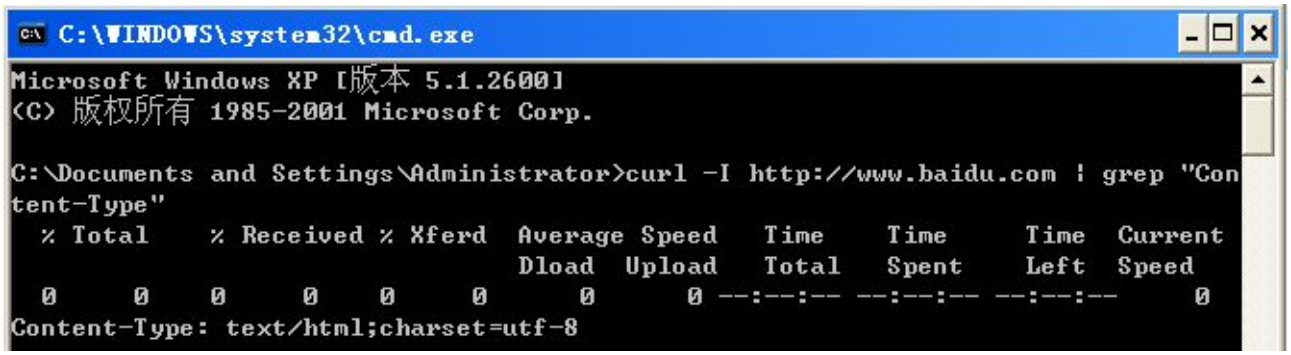

图 3.x

curl 是利用 URL 语法在命令行方式下工作的开源文件传输工具。

Content-Type ，内容类型，一般是指网页中存在的 Content-Type ，用于定义网络文件的类型和网页的编码，决定浏览器将以什么形式、什么编码读取这个文件。

另一种方法是使用 requests 模块来查找内容类型。Requests 是一个 Python 的 HTTP 客户端库，默认下载的 python 环境包不包含这个类库，需要另外安装。使用方法如下：

```
import requests
print requests.head(’ http://www.python.org’ ).headers[’ content-type’ ]
```

一旦确定了内容的类型，就可以用它来设置 Firefox 的默认配置文件，具体实例如下：

```
#coding=utf-8
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()
```

browser.download.dir 用于指定你所下载文件的目录。

os.getcwd()

该函数不需要传递参数，用于返回当前的目录。

application/octet-stream 为内容的类型。

第十七节 调用 JavaScript

当 webdriver 遇到没法完成的操作时，笔者可以考虑借用 JavaScript 来完成，比下下面的例子，通过 JavaScript 来隐藏页面上的元素。除了完成 webdriver 无法完成的操作，如果你熟悉 JavaScript 的话，那么使用 webdriver 执行 JavaScript 是一件非常高效的事情。

webdriver 提供了 execute_script() 接口用来调用 js 代码。

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=" selenium-webdriver(python)">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/bootstrap.min.js"></script>
</html>
```

保存 js.html 文件并通过浏览器打开，效果如图：

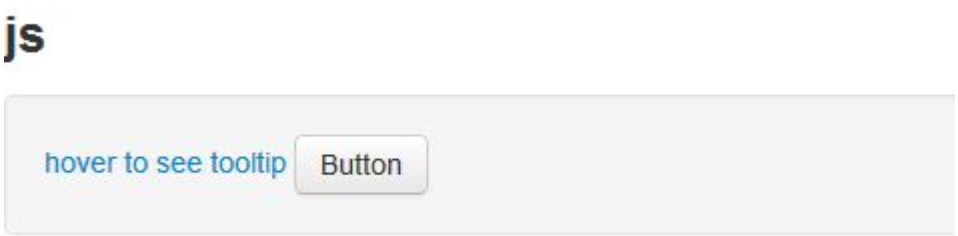

图3.x

执行 js 一般有两种场景：

- 一种是在页面上直接执行 JS
- 另一种是在某个已经定位的元素上执行 JS

```
#coding=utf-8
from selenium import webdriver
import time,os

driver = webdriver.Firefox()
file_path =  'file:///' + os.path.abspath('js.html')
driver.get(file_path)

#######通过 JS 隐藏选中的元素##########第一种方法：
#隐藏文字信息
driver.execute_script('$("#tooltip").fadeOut();')
time.sleep(5)

#隐藏按钮：
button = driver.find_element_by_class_name('btn')
driver.execute_script('$(arguments[0]).fadeOut()',button)
time.sleep(5)

driver.quit()
```

execute_script(script, *args)

在当前窗口/框架 同步执行 javaScript

script：JavaScript 的执行。

*args：适用任何 JavaScript 脚本。

关于 JavaScript 代码的解析不在本书的范围之内，请读者通过其它资料学习理解 JavaScript 的使用。

隐藏之后的效果如图 3.x

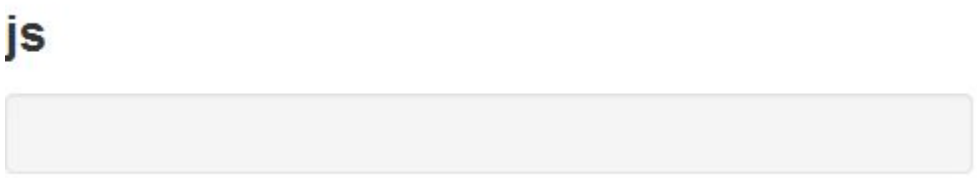

图 3.x

第十八节、控制浏览器滚动条

有时候 web 页面上的元素并非直接可见的，就算把浏览器最大化，我们依然需要拖动滚动条才能看到想要操作的元素，这个时候就要控制页面滚动条的拖动，但滚动条并非页面上的元素，可以借助 JavaScript 是来完成操作。

一般用到操作滚动条的会两个场景：

- 注册时的法律条文的阅读，判断用户是否阅读完成的标准是：滚动条是否拉到最下方。
- 要操作的页面元素不在视觉范围，无法进行操作，需要拖动滚动条

用于标识滚动条位置的代码

```
<body   onload= "document.body.scrollTop=0 ">
<body   onload= "document.body.scrollTop=100000 ">
```

如果滚动条在最上方的话，scrollTop=0 ，那么要想使用滚动条在最可下方，可以 scrollTop=100000 ，这样就可以使滚动条在最下方。

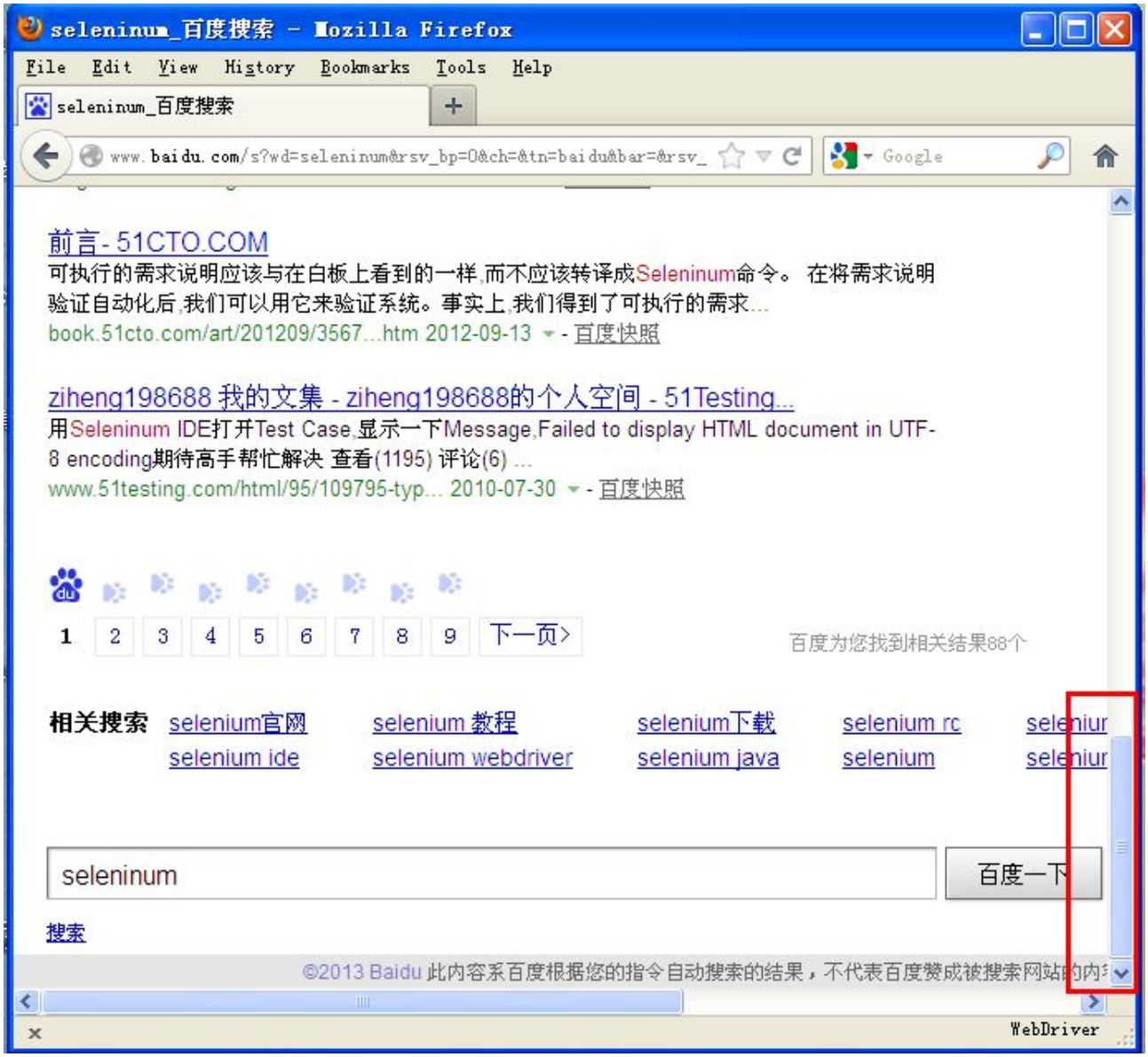

图 3. x

图 3. x 百度搜索结果页且滚动条在页面底，下面通过脚本实现：

```
#coding=utf-8
from selenium import webdriver
import time

#访问百度
driver=webdriver.Firefox()
driver.get("http://www.baidu.com")

#搜索
driver.find_element_by_id("kw").send_keys("selenium")
driver.find_element_by_id("su").click()
```

```
time.sleep(3)


#将页面滚动条拖到底部
js="var q=document.documentElement.scrollTop=10000"
driver.execute_script(js)
time.sleep(3)


#将滚动条移动到页面的顶部
js_="var q=document.documentElement.scrollTop=0"
driver.execute_script(js_)
time.sleep(3)


driver.quit()
```

第十九节　cookie 处理

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。webdriver 可以读取、添加和删除 cookie 信息。

webdriver 操作 cookie 的方法有：

- get_cookies()　获得所有 cookie 信息
- get_cookie(name)　返回特定 name 有 cookie 信息
- add_cookie(cookie_dict)　添加 cookie，必须有 name 和 value 值
- delete_cookie(name)　删除特定(部分)的 cookie 信息
- delete_all_cookies()　删除所有 cookie 信息

通过 webdriver 操作 cookie 是一件非常有意思的事儿，有时候我们需要了解浏览器中是否存在了某个 cookie 信息，webdriver 可以帮助我们读取、添加，删除 cookie 信息。

3.19.1 打印 cookie 信息

```
#coding=utf-8

from selenium import webdriver

import time


driver = webdriver.Chrome()

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


# 获得 cookie 信息

cookie= driver.get_cookies()

#将获得 cookie 的信息打印

print cookie

driver.quit()
```

运行打印信息：

```
   [{u'domain': u'.youdao.com', u'secure': False, u'value': u'aGFzbG9nZ2VkPXRydWU=',
   u'expiry': 1408430390.991375, u'path': u'/', u'name': u'_PREF_ANONYUSER__MYTH'},
   {u'domain': u'.youdao.com', u'secure': False, u'value':
   u'1777851312@218.17.158.115', u'expiry': 2322974390.991376, u'path': u'/', u'name':
   u'OUTFOX_SEARCH_USER_ID'}, {u'path': u'/', u'domain': u'www.youdao.com', u'name':
   u'JSESSIONID', u'value': u'abcUX9zdw0minadIhtvcu', u'secure': False}]
```

3.19.2、对 cookie 操作

上面的方式打印了所有 cookie 信息，太多太乱，我们只想有真对性的打印自己想要的信息，看下面的例子

```
#coding=utf-8
from selenium import webdriver
import time


driver = webdriver.Firefox()
driver.get("http://www.youdao.com")


#向 cookie 的 name 和 value 添加会话信息。
driver.add_cookie({'name':'key-aaaaaaa', 'value':'value-bbbb'})


#遍历 cookies 中的 name 和 value 信息打印，当然还有上面添加的信息
```

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


##### 下面可以通过两种方式删除 cookie #####
# 删除一个特定的 cookie
driver.delete_cookie("CookieName")
# 删除所有 cookie
driver.delete_all_cookies()


time.sleep(2)
driver.close()
```

运行打印信息：

```
YOUDAO_MOBILE_ACCESS_TYPE -> 1
_PREF_ANONYUSER__MYTH -> aGFzbG9nZ2VkPXRydWU=
OUTFOX_SEARCH_USER_ID -> -1046383847@218.17.158.115
JSESSIONID -> abc7qSE_SBGsVgnVLBvcu
key-aaaaaaa -> value-bbbb  # 这一条是我们自己添加的
```

第二十节　获取对象的属性

获取测试对象的属性能够帮我们更好的进行对象的定位。比如页面上有很多标签为 input 元素，而我们需要定位其中 1 个有具有 data-node 属性不一样的元素。由于 webdriver 是不支持直接使用 data-node 来定位对象的，所以我们只能先把所有标签为 input 都找到，然后遍历这些 input，获取想要的元素。

例如，有下面一组元素：

```
<input type="checkbox"   data-node="594434499" data-convert="1" data-type="file">

<input type="checkbox"   data-node="594434498" data-convert="1" data-type="file">

<input type="checkbox"   data-node="594434493" data-convert="1" data-type="file">

<input type="checkbox"   data-node="594434497" data-convert="1" data-type="file">
```

通过 find_elements 获得一组元素，通过循环遍历打到想要的元素：

```
# 选择页面上所有的 tag name 为 input 的元素
```

```
inputs = driver.find_elements_by_tag_name('input')
#然后循环遍历出 data-node 为594434493的元素，单击勾选
for input in inputs:
    if input.get_attribute('data-node') == '594434493':
        input.click()
……
```

如果读者细心会发现，我们在前面定位一组对象时已经用到了这个方法，当时判断是具有一组相同属性的元素，对其进行操作。这里判断是属性值不同的元素对其进行操作。灵活的运用这个技巧，才会让我们面对各种对象和需求时变得游刃有余。

第二十一节　验证码问题

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棘手的问题。

图 3.x

下面笔者根据自己的经验来谈一下处理验证码的几种方法。

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。

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。

验证码识别技术

例如可以通过 Python-tesseract 来识别图片验证码，Python-tesseract 是光学字符识别 Tesseract OCR 引擎的 Python 封装类。能够读取任何常规的图片文件(JPG, GIF ,PNG , TIFF 等)。不过，目前市面上的验证码形式繁多，目前任何一种验证码识别技术，识别率都不是 100% 。

记录 cookie

通过向浏览器中添加 cookie 可以绕过登录的验证码，这是比较有意思的一种解决方案。我们可以在用户登录之前，通过 add_cookie() 方法将用户名密码写入浏览器 cookie ，再次访问系统登录链接将自动登录。例如下面的方式：

```
....
#访问 xxxx 网站
driver.get("http://www.xxxx.cn/")
#将用户名密码写入浏览器 cookie
driver.add_cookie({'name':'Login_UserNumber', 'value':'username'})
driver.add_cookie({'name':'Login_Passwd', 'value':'password'})
#再次访问 xxxx 网站，将会自动登录
driver.get("http://www.xxxx.cn/")
time.sleep(3)
....
driver.quit()
```

使用 cookie 进行登录最大的难点是如何获得用户名密码的 name ，如果找到不到 name 的名字，就没办法向 value 中输用户名、密码信息。

笔者的建议是可以通过 get_cookies()方法来获取登录的所有的 cookie 信息，从而进行找到用户名、密码的 name 对象的名字；当然，最简单的方法还是询问前端开发人员。

第二十二节 weddriver 原理

webdriver 原理：

1. WebDriver 启动目标浏览器，并绑定到指定端口。该启动的浏览器实例，做为 web driver 的 remote server。

2. Client 端通过 CommandExcuter 发送 HTTPRequest 给 remote server 的侦听端口（通信协议： the webriver wire protocol）

3. Remote server 需要依赖原生的浏览器组件（如：IEDriverServer.exe、chromedriver.exe），来转化转化浏览器的 native 调用。

总结：

通过本章的学习，我们比较全面的掌握了如何使用 webdriver 所提供的方法对页面上各种元素进行操作。不过在实际的自动化测试过程中，读者会遇到各种各样的问题，笔者建议读者从以下几个方面进行提高：

1、熟练掌握 xpath\CSS 定位的使用，这样在遇到各种难以定位的属性时才不会变得束手无策。

2、准备一份 python 版本的 webdriver API ，遇到不理解地方，及时查到 API 的使用

3、学习掌握 JavaScript 语言，掌握 JavaScript 好处前面已经有过阐述，可以让我们的自动化测试工作更加游刃有余。

4、自动化测试归根结底是与前端打交道，多多熟悉前端技术，如 http 请求，HTML 语言 ，cookie /session 机制等。