软件测试教程 自动化测试selenium篇（二）

第一篇主要讲解了自动化测试的基本内容。最后用IDE生成了一个样例。这节课将详细的介绍webdriver常用的API。

本课程主要讲述以下内容：

> webdriver API讲解

webdriver API

一个简单自动化脚本的构成：

```
# coding = utf-8
from selenium import webdriver
import time
browser = webdriver.Firefox()
time.sleep(3)
browser.get("http://www.baidu.com")
time.sleep(3)
browser.find_element_by_id("kw").send_keys("selenium")
time.sleep(3)
browser.find_element_by_id("su").click()
browser.quit()
```


脚本解析

- coding = utf-8

防止乱码，在编辑器里面可以不用加，因为编辑器默认的就是UTF-8模式。

- from selenium import webdriver

导入webdriver工具包，这样就可以使用里面的API

- browser = webdriver.Firefox()

获得被控制浏览器的驱动，这里是获得Firefox的，当然还可以获得Chrome浏览器，不过要想使这一段代码有效，必须安装相应的浏览器驱动。

- browser.find_element_by_id("kw").send_keys("selenium")

通过元素的ID定位想要操作的元素，并且向元素输入相应的文本内容 。

- browser.find_element_by_id("su").click()

通过元素的ID定位到元素，并进行点击操作。

- browser.quit()

退出并关闭窗口。

browser.close()也可以关闭窗口。两者的区别是：

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。

元素的定位

对象的定位应该是自动化测试的核心，要想操作一个对象，首先应该识别这个对象。一个对象就是一个人一样，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，或者他住在哪个街道、楼层、门牌找到这个人。

那么一个对象也有类似的属性，我们可以通过这些属性找到这对象。

注意：不管用那种方式，必须保证页面上该属性的唯一性

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

我们可以看到，一个百度的输入框，可以用这么多种方式去定位。

```
<input id="kw" class="s_ipt" type="text" maxlength="100" name="wd"
autocomplete="off">
```

```
#coding=utf-8
from selenium import webdriver
import time
browser = webdriver.Chrome()
browser.get("http://www.baidu.com")
#########百度输入框的定位方式##########
#通过id 方式定位
    browser.find_element_by_id("kw").send_keys("selenium")
#通过name 方式定位
browser.find_element_by_name("wd").send_keys("selenium")
#通过tag name 方式定位
browser.find_element_by_tag_name("input").send_keys("selenium")  不能成功，因为
input太多了不唯一。
#通过class name 方式定位
browser.find_element_by_class_name("s_ipt").send_keys("selenium")
#通过CSS 方式定位
browser.find_element_by_css_selector("#kw").send_keys("selenium")
#通过xphan 方式定位
browser.find_element_by_xpath("//*[@id='kw']").send_keys("selenium")
###############################################
browser.find_element_by_id("su").click()
time.sleep(3)
browser.quit()
```

id定位

id是页面元素的属性，我们最常用元素定位方式，但是不是所有的元素都有id的。如果一个元素有id属性，那么一般在整个页面是唯一的。所以我们一般可以用id来唯一的定位到这个元素

通过前端工具，例如Chrome浏览器的F12，找到了百度输入框的属性信息，如下：

```
<input id="kw" class="s_ipt" type="text" maxlength="100" name="wd"
autocomplete="off">
```

属性 id=”kw”

通过find_element_by_id("kw") 函数就可以定位到百度输入框

name 定位

如果这个元素有name，并且元素的name命名在整个页面是唯一的，那么我们可以用name来定位这个元素。

用上面百度输入框的例子，其中元素的属性name=”wd”

通过find_element_by_name("wd")函数同样也可以定位到百度输入框

tag name 定位和class name 定位

从上面的百度输入框的属性信息中，我们看到，不单单只有id 和name 两个属性， 比如class 和tag name(标签名)

input 就是一个标签的名字，可以通过find_element_by_tag_name("input") 函数来定位。

class="s_ipt"，通过find_element_by_class_name("s_ipt")函数定位百度输入框。

在这里要注意的是，不是所有的元素用 tag name或者 class name来定位元素，首先要保证该元素的这两种属性在页面上是唯一的，才能够准确的定位。

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 的比较灵活可以选择控件的任意属性，上面的例子中：

find_element_by_css_selector("#kw")

通过find_element_by_css_selector( )函数，选择取百度输入框的id 属性来定义

CSS的获取可以用chrome的F12开发者模式中Element-右键-copy-copy selector来获取

XPath 定位

什么是XPath：http://www.w3.org/TR/xpath/

XPath 基础教程：http://www.w3schools.com/xpath/default.asp

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。

XPATH的获取可以用chrome的F12开发者模式中Element-右键-copy-copy xpath来获取

link text定位

有时候不是一个输入框也不是一个按钮，而是一个文字链接，我们可以通过链接内容，也就是 来定位。

需要注意的是链接内容必须这个页面唯一，否则会报错。

```
#coding=utf-8
from selenium import webdriver
browser = webdriver.Chrome()
browser.get("http://www.baidu.com")
browser.find_element_by_link_text("hao123").click()
browser.quit()
```

Partial link text 定位

通过部分链接定位，这个有时候也会用到，我还没有想到很好的用处。拿上面的例子，我可以只用链接的一部分文字进行匹配：

```
#coding=utf-8
from selenium import webdriver
browser = webdriver.Chrome()
browser.get("http://www.baidu.com")
browser.find_element_by_partial_link_text("hao").click()
browser.quit()
```

操作测试对象

前面讲到了不少知识都是定位元素，定位只是第一步，定位之后需要对这个元素进行操作。是鼠标点击还是键盘输入，或者清除元素的内容，或者提交表单等。这个取决于定位元素需要进行的下一步操作。

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 清除对象输入的文本内容
- submit 提交
- text 用于获取元素的文本信息

鼠标点击与键盘输入

```
#coding=utf-8
from selenium import webdriver
import time
driver = webdriver.Chrome()
driver.get("http://www.baidu.com")
time.sleep(2)
driver.find_element_by_id("kw").send_keys("test")
time.sleep(2)
driver.find_element_by_id("kw").clear()
driver.find_element_by_id("kw").send_keys("selenium")
time.sleep(2)
#通过submit() 来操作
driver.find_element_by_id("su").submit()
time.sleep(3)
driver.quit()
```

send_keys("xx") 用于在一个输入框里输入xx 内容。

click() 用于点击一个按钮。

clear() 用于清除输入框的内容，比如百度输入框里默认有个“请输入关键字”的信息，再比如我们的登陆框一般默认会有“账号”“密码”这样的默认信息。clear 可以帮助我们清除这些信息。

submit 提交表单

打开百度搜索页面，按钮“百度一下”元素的类型type=“submit”，所以把“百度一下”的操作从click 换成submit 可以达到相同的效果：

```
driver.find_element_by_id("su").submit()
```

text 获取元素文本

text 用于获取元素的文本信息

```
#coding=utf-8
from selenium import webdriver
import time
driver = webdriver.Chrome()
driver.get("http://www.baidu.com")
time.sleep(2)
#id = cp 元素的文本信息
data=driver.find_element_by_id("bottom_layer").text
print data #打印信息
time.sleep(3)
driver.quit()
```

输出：

```
©2018 Baidu 使用百度前必读 意见反馈 京ICP证030173号
```

添加等待

sleep休眠

添加休眠非常简单，我们需要引入time 包，就可以在脚本中自由的添加休眠时间了，这里的休眠指固定休眠

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

隐式等待

通过添加implicitly_wait() 方法就可以方便的实现智能等待；implicitly_wait(30)的用法比time.sleep()更智能，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。

```
selenium.webdriver.remote.webdriver.implicitly_wait(time_to_wait)
```

time_to_wait 设置的等待时长。

隐式地等待并非一个固定的等待时间，当脚本执行到某个元素定位时，如果元素可以定位，则继续执行；如果元素定位不到，则它以轮询的方式不断的判断元素是否被定位到。直到超出设置的时长

用法:

browser.implicitly_wait(30)

```
# coding = utf-8
from selenium import webdriver
import time #调入time 函数
browser = webdriver.Chrome()
browser.get("http://www.baidu.com")
browser.implicitly_wait(30) #隐式等待30秒
browser.find_element_by_id("kw").send_keys("selenium")
browser.find_element_by_id("su").click()
browser.quit()
```

打印信息

打印title

示例

```
#coding = utf-8
from selenium import webdriver
driver = webdriver.Chrome()
driver.get('http://www.baidu.com')
print(driver.title) # 把页面title 打印出来
```

打印url

示例

```
#coding = utf-8
from selenium import webdriver
driver = webdriver.Chrome()
driver.get('http://www.baidu.com')
print(driver.current_url) #打印url
```


浏览器的操作

浏览器最大化

我们知道调用启动的浏览器不是全屏的，这样不会影响脚本的执行，但是有时候会影响我们“观看”脚本的执行。

```
browser.maximize_window()
```

示例：

```
#coding=utf-8
from selenium import webdriver
import time
browser = webdriver.Chrome()
browser.get("http://www.baidu.com")
print "浏览器最大化"
browser.maximize_window() #将浏览器最大化显示
time.sleep(2)
browser.find_element_by_id("kw").send_keys("selenium")
browser.find_element_by_id("su").click()
time.sleep(3)
browser.quit()
```

设置浏览器宽、高

最大化还是不够灵活，能不能随意的设置浏览的宽、高显示？当然是可以的。

```
browser.set_window_size(width, high)
```

示例：

```
#coding=utf-8
from selenium import webdriver
import time
browser = webdriver.Chrome()
browser.get("http://www.baidu.com")
time.sleep(2)
#参数数字为像素点
print("设置浏览器宽480、高800显示")
browser.set_window_size(480, 800)
time.sleep(3)
browser.quit()
```

操作浏览器的前进、后退

浏览器上有一个后退、前进按钮，对于做web 自动化测试的同学来说也比较容易实现。

```
#浏览器的前进
browser.forward()
#浏览器的后退
browser.back()
```

示例

```
#coding=utf-8
from selenium import webdriver
import time
browser = webdriver.Chrome()
#访问百度首页
first_url= 'http://www.baidu.com'
print("now access %s" %(first_url))
browser.get(first_url)
time.sleep(2)
#访问新闻页面
second_url='http://news.baidu.com'
```

```
print("now access %s" %(second_url))
browser.get(second_url)
time.sleep(2)
#返回（后退）到百度首页
print("back to %s "%(first_url))
browser.back()
time.sleep(1)
#前进到新闻页
print("forward to %s"%(second_url))
browser.forward()
time.sleep(2)
browser.quit()
```

控制浏览器滚动条

浏览器滚动条的控制需要依靠js脚本

```
#将浏览器滚动条滑到最顶端
document.documentElement.scrollTop=0

#将浏览器滚动条滑到最底端
document.documentElement.scrollTop=10000

#将浏览器滚动条滑到最底端，  示例
js="var q=document.documentElement.scrollTop=10000"
driver.execute_script(js)

其中，execute_script(script, *args)，在当前窗口/框架同步执行javaScript
```

示例：

```
#coding=utf-8
from selenium import webdriver
import time
#访问百度
driver=webdriver.Chrome()
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()
```

键盘事件

键盘按键用法

要使用键盘按键，必须引入keys 包：

from selenium.webdriver.common.keys import Keys

通过send_keys()调用按键：

send_keys(Keys.TAB) # TAB

send_keys(Keys.ENTER) # 回车

send_keys(Keys.SPACE) #空格键

send_keys(Keys.ESCAPE) #回退键（Esc）

.....

示例：

```
#coding=utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys #需要引入keys 包
import os,time
driver = webdriver.Chrome()
driver.get("http://demo.zentao.net/user-login-Lw==.html")
time.sleep(3)
driver.maximize_window() # 浏览器全屏显示
driver.find_element_by_id("account").clear()
time.sleep(3)
driver.find_element_by_id("account").send_keys("demo")
time.sleep(3)
#tab 的定位相当于清除了密码框的默认提示信息，等同上面的clear()
driver.find_element_by_id("account").send_keys(Keys.TAB)
time.sleep(3)
#通过定位密码框，enter（回车）来代替登陆按钮
driver.find_element_by_name("password").send_keys(Keys.ENTER)
'''
#也可定位登陆按钮，通过enter（回车）代替click()
driver.find_element_by_id("login").send_keys(Keys.ENTER)
'''
time.sleep(3)
driver.quit()
```

键盘组合键用法

send_keys(Keys.CONTROL,'a') #全选（Ctrl+A）

send_keys(Keys.CONTROL,'c') #复制（Ctrl+C）

send_keys(Keys.CONTROL,'x') #剪贴（Ctrl+X）

send_keys(Keys.CONTROL,'v') #粘贴（Ctrl+V）

示例：

```
#coding=utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
driver = webdriver.Chrome()
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)
#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("webdriver")
driver.find_element_by_id("su").click()
time.sleep(3)
driver.quit()
```

鼠标事件

要使用鼠标事件需要导入工具包：

from selenium.webdriver.common.action_chains import ActionChains

语法示例如下：

```
#鼠标拖动事件
ActionChains(driver).move_to_element(element).perform()
```

ActionChains(driver)

生成用户的行为。所有的行动都存储在actionchains 对象。通过perform()存储的行为。

move_to_element(element)

移动鼠标到一个元素中，menu 上面已经定义了他所指向的哪一个元素

perform()

执行所有存储的行为

ActionChains 类

- context_click() 右击
- double_click() 双击
- drag_and_drop() 拖动
- move_to_element() 移动

示例：

```
#coding=utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.action_chains import ActionChains
import time
driver = webdriver.Chrome()
driver.get("http://news.baidu.com")
qqq =driver.find_element_by_xpath(".//*[@id='s_btn_wr']")
ActionChains(driver).context_click(qqq).perform() #右键
ActionChains(driver).double_click(qqq).perform() #双击
#定位元素的原位置
element = driver.find_element_by_id("s_btn_wr")
#定位元素要移动到的目标位置
```

```
target = driver.find_element_by_class_name("btn")
#执行元素的移动操作
ActionChains(driver).drag_and_drop(element, target).perform()
```

定位一组元素

webdriver 可以很方便的使用findElement 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findElements 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

用以下HTML示例说明：

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8" />
<title>Checkbox</title>
</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="r1" />
</div>
</div>
<div class="control-group">
<label class="control-label" for="r">radio</label>
<div class="controls">
<input type="radio" id="r2" />
</div>
</div>
</form>
```

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

用浏览器打开这个页面我们看到三个复选框和两个单选框。下面我们就来定位这三个复选框。

```
#coding=utf-8
from selenium import webdriver
import time
import os
dr = webdriver.Chrome()
file_path = 'file:///' + os.path.abspath('checkbox.html')
dr.get(file_path)
# 选择页面上所有的input，然后从中过滤出所有的checkbox 并勾选之
inputs = dr.find_elements_by_tag_name('input')
for input in inputs:
    if input.get_attribute('type') == 'checkbox':
        input.click()
time.sleep(2)
dr.quit()
```

get_attribute：获得属性值。

```
大家思考一下，根据我们之前学习的定位元素的方式，还有没有其它操作方法可以达到相同的效果？
```


多层框架/窗口定位

对于一个web 应用，经常会出现框架（frame）或窗口（window）的应用，这也就给我们的定位带来了一定的困难。

- 定位一个frame ： switch_to.frame(name_or_id_or_frame_element)
- 定位一个窗口window：switch_to.window(name_or_id_or_frame_element)

多层框架的定位

switch_to.frame(name_or_id_or_frame_element)：通过frame的id或者name或者frame自带的其它属性来定位框架，这里switch_to.frame()把当前定位的主体切换了frame里。

switch_to.default_content：从frame中嵌入的页面里跳出，跳回到最外面的默认页面中。

用以下HTML示例说明：

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8" />
<title>frame</title>
<link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstra
p-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="500"></iframe>
<a href="javascript:alert('watir-webdriver better than
selenium webdriver;')">click</a>
</div>
</div>
</body>
</html>
```

下面通过switch_to.frame() 方法来进行定位：

```
#coding=utf-8
from selenium import webdriver
import time
import os
browser = webdriver.Chrome()
file_path = 'file:///' + os.path.abspath('frame.html')
browser.get(file_path)
browser.implicitly_wait(30)
#先找到到ifrome1（id = f1）
browser.switch_to.frame("f1")
#再找到其下面的ifrome2(id =f2)
browser.switch_to.frame("f2")
#下面就可以正常的操作元素了
browser.find_element_by_id("kw").send_keys("selenium")
browser.find_element_by_id("su").click()
time.sleep(3)
browser.quit()
```

多层窗口定位

有可能嵌套的不是框架，而是窗口，还有真对窗口的方法：switch_to.window

用法与switch_to.frame 相同：

driver.switch_to.window("windowName")

层级定位

有时候我们需要定位的元素没有直接在页面展示，而是需要对页面的元素经过一系列操作之后才展示出来，这个时候我们就需要一层层去定位.

用以下HTML示例说明：

```
    <html>
    <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8" />
    <title>Level Locate</title>
    <script type="text/javascript" src="http://code.jquery.com/jquery-
1.9.1.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>
```

定位思路：

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Action 这个选项。

```
#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)
#点击Link1链接（弹出下拉列表）
dr.find_element_by_link_text('Link1').click()

#在父亲元件下找到link 为Action 的子元素
menu = dr.find_element_by_id('dropdown1').find_element_by_link_text('Action')
#鼠标定位到子元素上
webdriver.ActionChains(dr).move_to_element(menu).perform()
time.sleep(2)
dr.quit()
```

下拉框处理

下拉框是我们最常见的一种页面元素，对于一般的元素，我们只需要一次就定位，但下拉框里的内容需要进行两次定位，先定位到下拉框对下拉框进行操作后，再定位到下拉框内里的选项。

用以下HTML示例说明：

```
<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>
```

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.Chrome()
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()
```

这里可能和之前的操作有所不同，首先要定位到下拉框的元素，然后选择下拉列表中的选项进行点击操作。

alert、confirm、prompt 的处理

- 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，如果alert 没有对话框就不能用了，不然会报错

注意：switch_to.alert()只能处理原生的alert

用以下HTML示例说明：

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8" />
<title>alert</title>
<script type="text/javascript" src="http://code.jquery.com/jquery-1.9.1.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"}); $('#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/bootstrap.min.js"></script>
</html>
```

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

```
import os
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()
sleep(2)
alert = dr.switch_to.alert()
alert.accept()
sleep(2)
dr.quit()
```

```
#接受警告信息
alert = dr.switch_to.alert()
alert.accept()
#得到文本信息打印
alert = dr.switch_to.alert()
print alert.text
#取消对话框（如果有的话）
alert = dr.switch_to.alert()
alert.dismiss()
#输入值
alert = dr.switch_to.alert()
alert.send_keys("hello word")
```

当alert中有对话框，而我们期望在alert的对话框中输入信息的时候要怎么处理呢?

用以下HTML示例说明:

```
<html>
<head>
<meta charset="UTF-8">
<title></title>
<script type="text/javascript">
function disp_prompt(){
var name=prompt("Please enter yourname","")
if (name!=null &&name!=""){
document.write("Hello " +name + "!")
}
}
</script>
</head>
<body>
<input type="button" onclick="disp_prompt()"
value="请点击"/>
</body>
</html>
```

```
#coding:utf-8

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

```
from selenium.common.exceptions import
NoSuchElementException,UnexpectedTagNameException
from selenium.webdriver.support.ui import Select
from selenium.webdriver.common.alert import Alert
from time import sleep
import os

driver=webdriver.Chrome()

driver.implicitly_wait(30)
file_path = 'file:///' + os.path.abspath('send.html')
driver.get(file_path)
#driver.get('file:///D:/PycharmProjects/test/send.html')
#点击“请点击”
driver.find_element_by_xpath("html/body/input").click()
#输入内容
driver.switch_to.alert().send_keys('webdriver')
driver.switch_to.alert().accept()
sleep(5)
driver.quit()
```

DIV对话框的处理

如果页面元素比较多，利用元素的属性无法准确的定位这个元素的时候，我们可以先定位元素所在的div块，再去定位这个元素。

用以下HTML示例说明：

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8" />
<title>modal</title>
<script type="text/javascript" src="http://code.jquery.com/jquery-1.9.1.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(){
$('#click').click(function(){
$(this).parent().find('p').text('Click on the link to success!');
});
});
</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">x</button>
<h3 id="myModalLabel">Modal header</h3>
</div>
<div class="modal-body">
<p>Congratulations, you open the window!</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/bootstrap.min.js"></script>
</html>
```

操作脚本如下：

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from time import sleep
import os
import selenium.webdriver.support.ui as ui
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()
sleep(3)
# 点击对话框中的链接
link = dr.find_element_by_id('myModal').find_element_by_id('click')
link.click()
#dr.execute_script('$(arguments[0]).click()', link)
sleep(4)
# 关闭对话框
buttons =dr.find_element_by_class_name('modal
footer').find_elements_by_tag_name('button')
buttons[0].click()
sleep(2)
dr.quit()
```


上传文件操作

文件上传操作也比较常见功能之一，上传功能没有用到新有方法或函数，关键是思路。

上传过程一般要打开一个本地窗口，从窗口选择本地文件添加。所以，一般会卡在如何操作本地窗口添加上传文件。

其实，在selenium webdriver 没我们想的那么复杂；只要定位上传按钮，通过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。

upload.html

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8" />
<title>upload_file</title>
<script type="text/javascript" src="http://code.jquery.com/jquery-1.9.1.min.js">
</script>
<link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstra
p-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>
```

```
#coding=utf-8
from selenium import webdriver
import os,time
driver = webdriver.Chrome()
#脚本要与upload_file.html 同一目录
file_path = 'file:///' + os.path.abspath('upload.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:\\PycharmProjects\\test\\upload
.txt')
time.sleep(2)
driver.quit()
```