selenium webdriver (python)

(第一版)

声明：

前言

如果你是一位有 python 语言基础的同学，又想通过 python+ selenium 去实施自动化，那么你非常幸运的找到了这份文档，我也非常荣幸能为你的自动化学习之路带来一丝帮助。

其实，我在 selenium 自动化方面也是初学者，虽然在刚开始从事测试工作的就尝试学习 selenium 自动化工具，购买了面《零成本实现 web 自动化测试---基于 seleinum 与 Bromine》学了一段时间不得门，当时水平有限，甚至一直都不理解什么自动化测试框架，后来就放弃了自动化的学习。

今年换工作后，所测试的项目用 python 开发，为了更好的测试项目，开始学习 python 语言，花一个月多月的时候将 python 基础学习了一下。正好所测试的项目也比较适合进行自动化，于是尝试通过 python+selenium 对项目进行自动化工作。

学习的过程还是比较艰难的，对于 selenium webdriver 的 ruby 和 java 的实例比较多，python 甚少，查询 API 文档有方法，但没有实例，不知道如何实现。在学习过程中要感谢 乙醇 同学，他在 ruby+selenium 方面有很深的功力；另一位要感谢的是 rabbit ，他在 python+selenium 的学习中给我了很多帮助。

下面要简单说说本文档的内容：

为了学习 selenium webdriver 本身的用法，全文没有引入 python 的 unittest 测试框架；所有的脚本也都是单个的用例的学习，引入框架后将会提供更强大的功能，比如用例的运行时间，用例的批量执行等。本文档是第一版，我此后也会继续学习 unittest 测试框架的技术，并将在第二版中加入。

本文档不是 API，所以还有很多方法没有整理，如果在学习的过程中有任何疑问，请查阅在线 AIP 文档：

http://selenium.googlecode.com/git/docs/api/py/index.html

2013.8.30

目录

一、selenium+python 环境搭建

1.1 selenium 介绍

selenium 是一个 web 的自动化测试工具，不少学习功能自动化的同学开始首选 selenium ，相因为它相比 QTP 有诸多有点：

* 免费，也不用再为破解 QTP 而大伤脑筋

* 小巧，对于不同的语言它只是一个包而已，而 QTP 需要下载安装1个多 G 的程序。

* 这也是最重要的一点，不管你以前更熟悉 C、 java、ruby、python、或都是 C# ，你都可以通过 selenium 完成自动化测试，而 QTP 只支持 VBS

* 支持多平台：windows、linux、MAC ，支持多浏览器：ie、ff、safari、opera、chrome

* 支持分布式测试用例的执行，可以把测试用例分布到不同的测试机器的执行，相当于分发机的功能。

1.2 准备工作

搭建平台 windows

准备工具如下：

下载 python

http://python.org/getit/

下载 setuptools 【python 的基础包工具】

http://pypi.python.org/pypi/setuptools

下载 pip 【python 的安装包管理工具】

https://pypi.python.org/pypi/pip

因为版本都在更新，pyhton 选择2.7.xx ，setuptools 选择你平台对应的版本，pip 不要担心 tar.gz 在 windows 下一样可用。

1.3 安装步骤

1、python 的安装 ，这个不解释，exe 文件运行安装即可，既然你选择 python，相信你是熟悉 python 的，我安装目录 C:\Python27

2、setuptools 的安装也非常简单，同样是 exe 文件，默认会找到 python 的安装路径，将安装到 C:\Python27\Lib\site-packages 目录下

3、安装 pip ，我默认解压在了 C:\pip-1.3.1 目录下

4、打开命令提示符（开始---cmd 回车）进入 C:\pip-1.3.1目录下输入：

C:\pip-1.3.1 > python setup.py install

（如果提示 python 不是内部或外部命令！别急，去配置一下环境变量吧）

修改我的电脑->属性->高级->环境变量->系统变量中的 PATH 为: 变量名：PATH 变量值：;C:\Python27

5、再切换到 C:\Python27\Scripts 目录下输入：

C:\Python27\Scripts > easy_install pip

6、安装 selenium，（下载地址： https://pypi.python.org/pypi/selenium ）

如果是联网状态的话，可以直接在 C:\Python27\Scripts 下输入命令安装：

C:\Python27\Scripts > pip install -U selenium

如果没联网（这个一般不太可能），下载 selenium 2.33.0 （目前的最新版本）

并解压把整个目录放到 C:\Python27\Lib\site-packages 目录下。

7、下载并安装（http://www.java.com/zh_CN/download/chrome.jsp?locale=zh_CN）什么？你没整过 java，参考其它文档吧！这不难。

8、 下载 selenium 的服务端（https://code.google.com/p/selenium/）在页面的左侧列表中找到

selenium-server-standalone-XXX.jar

对！就是这个东西，把它下载下来并解压；

在 selenium-server-standalone-xxx.jar 目录下使用命令 java -jar selenium-server-standalone-xxx.jar 启动（如果打不开，查看是否端口被占 用：netstat -aon|findstr 4444）。

1.4 安装 chrome driver

chrome driver 的下载地址在这里。

1. 下载解压，你会得到一个 chromedriver.exe 文件（我点开，运行提示 started no prot 9515 ，这是干嘛的？端口9515被占了？中间折腾了半天），后来才知道需要把这家伙放到 chrome 的安装目录下...\Google\Chrome\Application\ ，然后设置 path 环境变量，把 chrome 的安装目录（我的：C:\Program Files\Google\Chrome\Application），然后再调用运行：

```
# coding = utf-8

from selenium import webdriver

driver =webdriver.Chrome()

driver.get('http://radar.kuaibo.com')

print driver.title

driver.quit()
```

报错提示：

Chrome version must be >= 27.0.1453.0\n (Driver info: chromedriver=2.0,platform=Windows NT 5.1 SP3 x86)

说我 chrome 的版本没有大于27.0.1453.0 ，这个好办，更新到最新版本即可。

1.5 安装 IE driver

在新版本的 webdriver 中，只有安装了 ie driver 使用 ie 进行测试工作。

ie driver 的下载地址在这里，记得根据自己机器的操作系统版本来下载相应的 driver。

暂时还没尝试，应该和 chrome 的安装方式类似。

记得配置 IE 的保护模式

如果要使用 webdriver 启动 IE 的话，那么就需要配置 IE 的保护模式了。

把 IE 里的保护模式都选上或都勾掉就可以了。

二、开始第一个脚本

2.1 为什么选 python

之前的菜鸟系列是基于 java 的，一年没学其实也忘的差不多了，目前所测的产品部分也是 python 写的，而且团队也在推广 python ，其实就测试人员来说，python 也相当受欢迎。易学，易用。翻翻各测试招聘，python 出现的概率也颇高。(个人原因)

最重要的还是 python 简单易学，应用也相对广泛；是测试人员学习编程的不二之选。

下面看看 python 穿上 selenium webdriver 是多么的性感：

2.2 第一个脚本

```
# coding = utf-8

from selenium import webdriver

browser = webdriver.Firefox()

browser.get("http://www.baidu.com")

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

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

browser.quit()
```

2.3 脚本解析

```
# coding = utf-8
```

可加可不加，开发人员喜欢加一下，防止乱码嘛。

```
from selenium import webdriver
```

要想使用 selenium 的 webdriver 里的函数，首先把包导进来嘛

```
browser = webdriver.Firefox()
```

我们需要操控哪个浏览器呢？Firefox ，当然也可以换成 Ie 或 Chrome 。browser 可以随便取，但后面要用它操纵各种函数执行。

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

一个控件有若干属性 id 、name、（也可以用其它方式定位），百度输入框的 id 叫 kw ，我要在输入框里输入 selenium 。多自然语言呀！

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

搜索的按钮的 id 叫 su ，我需要点一下按钮（ click() ）。

```
browser.quit()
```

退出并关闭窗口的每一个相关的驱动程序，它还有个类似的表弟。

```
browser.close()
```

关闭当前窗口 ，用哪个看你的需求了。

三、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，我们可以通过这个属性找到这对象。

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，可以用这么用种方式去定位。

```
#coding=utf-8

from selenium import webdriver

browser = webdriver.Firefox()



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")

#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("//input[@id='kw']").send_keys("selen
ium")

##########################################
browser.find_element_by_id("su").click()
time.sleep(3)
browser.quit()
```

3.1 id 和 name 定位

id 和 name 是我们最最常用的定位方式，因为大多数控件都有这两个属性，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

我们通过前端工具，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">
```

id=”kw”

通过 find_element_by_id("kw") 函数就是捕获到百度输入框

name=”wd”

通过 find_element_by_name("wd")函数同样也可以捕获百度输入框

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

<input>

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")函数捕获百度输入框。

3.3 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

也可以取 name 属性

```
<a href="http://news.baidu.com" name="tj_news">新 闻</a>
```

driver.find_element_by_css_selector("a[name=\"tj_news\"]").click()

可以取 title 属性

```
<a      onclick="queryTab(this);"      mon="col=502&pn=0"      title="web"
href="http://www.baidu.com/">网页</a>
```

driver.find_element_by_css_selector("a[title=\"web\"]").click()

也可以是取...:

```
<a class="RecycleBin xz" href="javascript:void(0);">
```

driver.find_element_by_css_selector("a.RecycleBin").click()

虽然我也没全部理解 CSS 的定位，但是看上去应该是一种非常灵活和牛 X 的定位方式

扩展阅读：

http://www.w3.org/TR/css3-selectors/

http://www.w3school.com.cn/css/css_positioning.asp

3.4 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

selenium 中被误解的 XPath ： http://magustest.com/blog/category/webdriver/

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。

```
xpath:attributer （属性）

driver.find_element_by_xpath("//input[@id='kw']").send_keys("selenium")

#input 标签下 id =kw 的元素


xpath:idRelative （id 相关性）

driver.find_element_by_xpath("//div[@id='fm']/form/span/input").send_keys("s
elenium")

#在/form/span/input 层级标签下有个 div 标签的 id=fm 的元素


driver.find_element_by_xpath("//tr[@id='check']/td[2]").click()

# id 为'check' 的 tr ，定闪他里面的第2个


xpath:position （位置）

driver.find_element_by_xpath("//input").send_keys("selenium")

driver.find_element_by_xpath("//tr[7]/td[2]").click()

#第7个 tr 里面的第2个 td


xpath: href （水平参考）

driver.find_element_by_xpath("//a[contains(text(),'网页')]").click()

#在 a 标签下有个文本（text）包含（contains）'网页' 的元素


xpath:link

driver.find_element_by_xpath("//a[@href='http://www.baidu.com/']").click()

#有个叫 a 的标签，他有个链接 href='http://www.baidu.com/ 的元素
```

3.5 link 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 link

```
#coding=utf-8


from selenium import webdriver

browser = webdriver.Firefox()

browser.get("http://www.baidu.com")

browser.find_element_by_link_text("贴 吧").click()

browser.quit()
```

一般一个那页面上不会出现相同的文件链接，通过文字链接来定位也是一种简单有效的定位方式。

3.6 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：

```
browser.find_element_by_partial_link_text("贴").click()

#通过 find_element_by_partial_link_text() 函数，我只用了“贴”字，脚本一样找到了"贴 吧
" 的链接
```

四、添加等待时间

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。

4.1、添加休眠

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。

```
# coding = utf-8



from selenium import webdriver

import  time  #调入 time 函数

browser = webdriver.Firefox()



browser.get("http://www.baidu.com")

time.sleep(0.3)  #休眠0.3秒

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

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

time.sleep(3)  # 休眠3秒

browser.quit()
```

4.2、智能等待

通过添加 implicitly_wait() 方法就可以方便的实现智能等待；implicitly_wait(30) 的用法应该比 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。

文档解释：

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

隐式地等待一个无素被发现或一个命令完成；这个方法每次会话只需要调用一次

time_to_wait: 等待时间

用法:

browser.implicitly_wait(30)

```
# coding = utf-8



from selenium import webdriver

import  time  #调入 time 函数

browser = webdriver.Firefox()



browser.get("http://www.baidu.com")

browser.implicitly_wait(30)  #智能等待30秒

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

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



browser.quit()
```

五、打印信息

很多时间我们不可能盯着脚本执行，我们需要一些打印信息来证明脚本运行是否正确：

5.1、打印 tile

把刚才访问页面的 title 打印出来。

```
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

driver.quit()
```

虽然我没看到脚本的执行过程，但我在执行结果里看到了

```
>>>

百度一下，你就知道
```

说明页面正确被我打开了。

5.2、打印 URL

可以将浏览器的 title 打印出来，这里再讲个简单的，把当前 URL 打印出来。其实也没啥大用，可以做个凑数的用例。

```
#coding=utf-8

from selenium import webdriver

import time

browser = webdriver.Firefox()

url= 'http://www.baidu.com'



#通过 get 方法获取当前 URL 打印

print "now access %s" %(url)

browser.get(url)
```

```
time.sleep(2)

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

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

time.sleep(3)

browser.quit()
```

六、浏览器的操作

6.1、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。

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


browser = webdriver.Firefox()
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()
```

6.2、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。

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

browser = webdriver.Firefox()
browser.get("http://m.mail.10086.cn")
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()
```

七、操作浏览器的前进、后退

浏览器上有一个后退、前进按钮，对于浏览网页的人是比较方便的；对于做 web 自动化测试的同学来说应该算是一个比较难模拟的问题；其实很简单，下面看看 python 的实现方式。

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

```
browser = webdriver.Firefox()


#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()
```

为了使过程让你看得更清晰，在每一步操作上都加了 print 和 sleep 。

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。

八、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原素进行操作。鼠标点击呢还是键盘输入，这要取决于我们定位的是按钮还输入框。

一般来说，webdriver 中比较常用的操作对象的方法有下面几个

- click 点击对象
- send_keys 在对象上模拟按键输入
- clear 清除对象的内容，如果可以的话
- submit 清除对象的内容，如果可以的话

在我们本系列开篇的第一个例子里就用到了到 click 和 send_skys ，别翻回去找了，我再贴一下代码：

```
coding=utf-8

from selenium import webdriver

import time


driver = webdriver.Firefox()

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


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

我们把“百度一下”的操作从 click 换成 submit ：

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

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

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()
```

九、键盘事件

本章重点：

- 键盘按键用法
- 键盘组合键用法
- send_keys() 输入中文运行报错问题

9.1、键盘按键用法

```
#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.Firefox()

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


time.sleep(3)

driver.maximize_window() # 浏览器全屏显示

driver.find_element_by_id("user_name").clear()

driver.find_element_by_id("user_name").send_keys("fnngj")


#tab 的定位相相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("user_name").send_keys(Keys.TAB)

time.sleep(3)

driver.find_element_by_id("user_pwd").send_keys("123456")


#通过定位密码框，enter（回车）来代替登陆按钮

driver.find_element_by_id("user_pwd").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)


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

要想调用键盘按键操作需要引入 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
```

注意：这个操作和页面元素的遍历顺序有关，假如当前定位在账号输入框，按键盘的 tab 键后遍历的不是密码框，那就不法输入密码。 假如输入密码后，还有需要填写验证码，那么回车也起不到登陆的效果。

9.1、键盘组合键用法

```
#coding=utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import os,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)

#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(u"虫师 cnblogs")
driver.find_element_by_id("su").click()

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

上面的操作没有实际意义，但向我们演示了键盘组合按键的用法。

十、鼠标事件

本章重点：

ActionChains 类

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

测试的产品中有一个操作是右键点击文件列表会弹出一个快捷菜单，可以方便的选择快捷菜单中的选择对文件进行操作（删除、移动、重命名），之前学习元素的点击非常简单：

driver.find_element_by_id(“xxx”).click()

那么鼠标的双击、右击、拖动等是否也是这样的写法呢？例如右击：

driver.find_element_by_id(“xxx”).context_click()

经过运行脚本得到了下面的错误提示：

AttributeError: 'WebElement' object has no attribute 'context_click'

提示右点方法不属于 webelement 对象，通过查找文档，发现属于 ActionChains 类，但文档中没有具体写法。这里要感谢 北京-QC-rabbit 的指点，其实整个 python+selenium 学习过程都要感谢 北京-QC-rabbit 的指点。

10.1、鼠标右键

下面介绍鼠标右键的用法，以快播私有云为例：

```
#coding=utf-8

from selenium import webdriver

from selenium.webdriver.common.action_chains import ActionChains

import time


driver = webdriver.Firefox()

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


#登陆快播私有云

driver.find_element_by_id("user_name").send_keys("username")

driver.find_element_by_id("user_pwd").send_keys("123456")

driver.find_element_by_id("dl_an_submit").click()

time.sleep(3)
```

```
#定位到要右击的元素
qqq
=driver.find_element_by_xpath("/html/body/div/div[2]/div[2]/div/div[3]/table/tbody/tr/td[2]")
#对定位到的元素执行鼠标右键操作
ActionChains(driver).context_click(qqq).perform()


'''
#你也可以使用三行的写法，但我觉得上面两行写法更容易理解
chain = ActionChains(driver)
implement =
driver.find_element_by_xpath("/html/body/div/div[2]/div[2]/div/div[3]/table/tbody/tr/td[2]")
chain.context_click(implement).perform()
'''

time.sleep(3) #休眠3秒
driver.close()
```

这里需要注意的是，在使用 ActionChains 类之前，要先将包引入。

右击的操作会了，下面的其它方法比葫芦画瓢也能写出来。

10.2、鼠标双击

鼠标双击的写法：

```
#定位到要双击的元素
qqq =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(qqq).perform()
```

10.3、鼠标拖放

鼠标拖放操作的写法：

```
#定位元素的原位置
element = driver.find_element_by_name("source")
#定位元素要移动到的目标位置
target =  driver.find_element_by_name("target")

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

十一、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

checkbox.html

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

```
        <script
src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></
script>
    </head>
    <body>
        <h3>checkbox</h3>
        <div class="well">
            <form class="form-horizontal">
                <div class="control-group">
                    <label class="control-label" for="c1">checkbox1</label>
                    <div class="controls">
                        <input type="checkbox" id="c1" />
                    </div>
                </div>
                <div class="control-group">
                    <label class="control-label" for="c2">checkbox2</label>
                    <div class="controls">
                        <input type="checkbox" id="c2" />
                    </div>
                </div>
                <div class="control-group">
                    <label class="control-label" for="c3">checkbox3</label>
                    <div class="controls">
                        <input type="checkbox" id="c3" />
                    </div>
                </div>

                <div class="control-group">
                    <label class="control-label" for="r">radio</label>
                    <div class="controls">
                        <input type="radio" id="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>
```

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）

11.1、第一种定位方法

通过浏览器打个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.Firefox()
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()
```

11.2、第二种定位方法

第二种写法与第一种写法差别不大，都是通过一个循环来勾选控件。

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

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

# 选择所有的 checkbox 并全部勾上
checkboxes = dr.find_elements_by_css_selector('input[type=checkbox]')
for checkbox in checkboxes:
    checkbox.click()
time.sleep(2)

# 打印当前页面上有多少个 checkbox
print len(dr.find_elements_by_css_selector('input[type=checkbox]'))
time.sleep(2)

dr.quit()
```

11.3、去掉最后一个勾选

还有一个问题，有时候我们并不想勾选页面的所有的复选框（checkbox），可以通过下面办法把最后一个被勾选的框去掉。如下：

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

```
import os


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

# 选择所有的 checkbox 并全部勾上
checkboxes = dr.find_elements_by_css_selector('input[type=checkbox]')
for checkbox in checkboxes:
    checkbox.click()
time.sleep(2)

# 把页面上最后1个 checkbox 的勾给去掉
dr.find_elements_by_css_selector('input[type=checkbox]').pop().click()
time.sleep(2)

dr.quit()
```

其实，去掉勾选表也逻辑也非常简单，就是再次点击勾选的按钮。可能我们比较迷惑的是如何找到“最后一个”按钮。pop() 可以实现这个功能。

十二、多层框架/窗口定位

本节知识点：

多层框架或窗口的定位：

- switch_to_frame()
- switch_to_window()

对于一个现代的 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个难题。

有时候我们定位一个元素，定位器没有问题，但一直定位不了，这时候就要检查这个元素是否在一个 frame 中，seelnium webdriver 提供了一个 switch_to_frame 方

法，可以很轻松的来解决这个问题。

12.1、多层框架定位

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/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>
```

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。

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.Firefox()
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()
```

12.1、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")

十三、层级定位

假如两个控件，他们长的一模样，还都叫“张三”，唯一的不同是一个在北京，一个在上海，那我们就可以通过，他们的城市，区，街道，来找到他们。

在实际的测试中也经常会遇到这种问题：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>
```

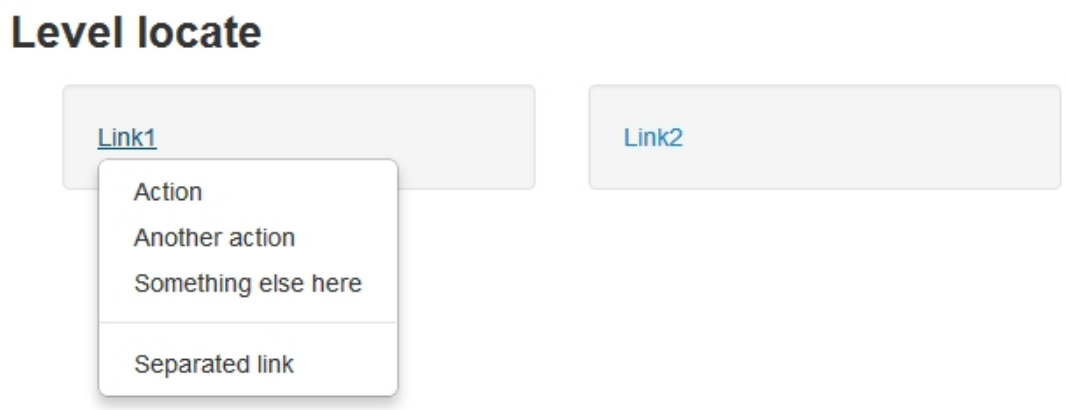
```
    </body>

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

   </html>
```

上面的 html 代码比较乱，请复制到编辑器中查看，如 nodepad ++ 编辑器。

（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）通过浏览器打开：



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。

脚本如下：

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
import time
import os


dr = webdriver.Firefox()
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()


#找到 id 为 dropdown1的父元素

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

#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()
```

WebDriverWait(dr, 10)

10秒内每隔500毫秒扫描1次页面变化，当出现指定的元素后结束。dr 就不解释了，前面操作 webdriver.firefox()的句柄

is_displayed()

该元素是否用户可以见

class ActionChains(driver)

driver：执行用户操作实例 webdriver

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(menu)

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

to_element：元件移动到

perform()

执行所有存储的行为

十四、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 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。

14.1、操作文件上传例子

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/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>
```

操作上传脚本：

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


driver = webdriver.Firefox()


#脚本要与 upload_file.html 同一目录
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_c
ase\upload_file.txt')
time.sleep(2)


driver.quit()
```

14.2、139 邮箱上传

其它有些应用不好找，所以就自己创建页面，这样虽然麻烦，但脚本代码突出重点。

这里找一139邮箱的实例，有帐号的同学可以测试一下~！

（登陆基础版的139邮箱，网盘模块上传文件。）

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

```
import os,time


driver = webdriver.Firefox()
driver.get("http://m.mail.10086.cn")
driver.implicitly_wait(30)

#登陆
driver.find_element_by_id("ur").send_keys("手机号")
driver.find_element_by_id("pw").send_keys("密码")
driver.find_element_by_class_name("loading_btn").click()
time.sleep(3)

#进入139网盘模块
driver.find_element_by_xpath("/html/body/div[3]/a[9]/span[2]").click()
time.sleep(3)

#上传文件
driver.find_element_by_id("id_file").send_keys('D:\\selenium_use_case\upload_file.txt')
time.sleep(5)


driver.quit()
```

十五、下拉框处理

本节重点

- 处理下拉框
- switch_to_alert()
- accept()

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

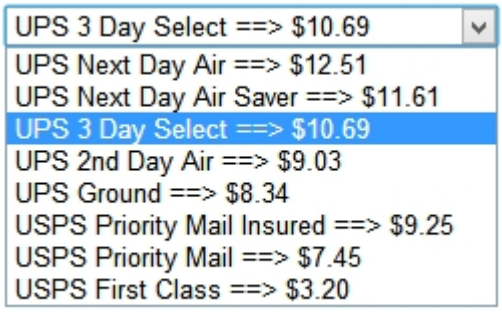
需要进行两次定位，先定位到下拉框，再定位到下拉框内里的选项。

15.1、操作下拉框例子

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：



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()
```

解析：

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。

m=driver.find_element_by_id("ShippingMethod")

m.find_element_by_xpath("//option[@value='10.69']").click()

15.2、百度搜索设置下拉框操作

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

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

#进入搜索设置页
driver.find_element_by_link_text("搜索设置").click()

#设置每页搜索结果为100条
m=driver.find_element_by_name("NR")
```

```
m.find_element_by_xpath("//option[@value='100']").click()
time.sleep(2)

#保存设置的信息
driver.find_element_by_xpath("//input[@value='保存设置']").click()
time.sleep(2)
driver.switch_to_alert().accept()

#跳转到百度首页后，进行搜索表（一页应该显示100条结果）
driver.find_element_by_id("kw").send_keys("selenium")
driver.find_element_by_id("su").click()
time.sleep(3)

driver.quit()
```

解析：

当我们在保存百度的设置时会会弹出一个确定按钮；我们并没按照常规的方法去定位弹窗上的“确定”按钮，而是使用：

driver.switch_to_alert().accept()

完成了操作，这是因为弹窗比较是一个具有唯一性的警告信息，所以可以用这种简便的方法处理。

- switch_to_alert()

焦点集中到页面上的一个警告（提示）

- accept()

接受警告提示

十六、调用 js

本节重点：

调用 js 方法

- 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。

使用：

driver.execute_script（‘document.title’）

15.1、通过 js 隐藏元素

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"></s
cript>
      <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(){

       $('#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>
```

(保持 html 文件与执行脚本在同一目录下)

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：
driver.execute_script('$("#tooltip").fadeOut();')
time.sleep(5)

#第二种方法：
button = driver.find_element_by_class_name('btn')
driver.execute_script('$(arguments[0]).fadeOut()',button)
time.sleep(5)

driver.quit()
```

15.2、通过 js 使输入框标红

```
#coding=utf-8

from selenium import webdriver

import time


driver = webdriver.Firefox()

driver.get("http://passport.kuaibo.com/login/?referrer=http%3A%2F%2Fvod.kuaibo.com%2F%3Ft%3Dhome")


#给用户名的输入框标红

js="var   q=document.getElementById(\"user_name\");q.style.border=\"1px   solid red\";"

#调用 js

driver.execute_script(js)

time.sleep(3)


driver.find_element_by_id("user_name").send_keys("username")

driver.find_element_by_id("user_pwd").send_keys("password")

driver.find_element_by_id("dl_an_submit").click()

time.sleep(3)


driver.quit()
```

js 解释：

q=document.getElementById(\"user_name\")

元素 q 的 id 为 user_name

q.style.border=\"1px solid red\

元素 q 的样式，边框为1个像素红色

十六、控制浏览器滚动条

有时候我们需要控制页面滚动条上的滚动条，但滚动条并非页面上的元素，这个时候就需要借助 js 是来进行操作。一般用到操作滚动条的会两个场景：

- 注册时的法律条文需要阅读，判断用户是否阅读的标准是：滚动条是否拉到最下方。
- 要操作的页面元素不在吸视范围，无法进行操作，需要拖动滚动条

其实，实现这个功能只要一行代码，但由于不懂 js ，所以花了不小力气找到这种方法。

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。

16.1、场景一

先来解决场第一个问题，法律条款是一个内嵌窗口，通过 firebug 工具可以定位到内嵌入窗口可以定位到元素的 id ，可以通过下面的代码实现。

```
js="var q=document.getElementById('id').scrollTop=10000"
driver.execute_script(js)
```

16.1、场景二

有滚动条的页面到处可见，这个就比较容易找例子，我们以操作百度搜索结果页为例：

```
#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()
```

十七、webdriver 原理解析

之前看乙醇视频中提到，selenium 的 ruby 实现有一个小后门，在代码中加上 $DEBUG=1 ，再运行脚本的过程中，就可以看到客户端请求的信息与服务器端返回的数据；觉得这个功能很强大，可以帮助理解 webdriver 的运行原理。

后来查了半天，python 并没有提供这样一个方便的后门，不过我们可以通过代理的方式获得这些交互信息；

一、需要安装 java 虚拟机与 selenium-server-standalone ，参考本文档第一章环境搭建第7、8步操作：

二、通过下面命令启动服务：

C:\selenium>java -jar selenium-server-standalone-2.33.0.jar

在命令结尾加 >d:\log.txt 可以将命令信息存入文件，但信息很少。

运行下面的自动化脚本：

```
#coding = utf-8
import time
from selenium import webdriver
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities

driver = webdriver.Remote(desired_capabilities=DesiredCapabilities.CHROME)
driver.get("http://www.youdao.com")
driver.find_element_by_name("q").send_keys("hello")
driver.find_element_by_name("q").send_keys("key.ENTER")

driver.close()
```

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：IEDriver.dll,chromedriver.exe），来转化转化浏览器的 native 调用。

查看命令提示符下的运行日志：

咋一看很乱，慢慢分析一下就发现很有意思！结合上面的脚本分析

```
------------------------------------------------------------------------启动代理进入监听状态
C:\selenium>java -jar selenium-server-standalone-2.33.0.jar
八月 22, 2013 10:19:48 上午 org.openqa.grid.selenium.GridLauncher main
INFO: Launching a standalone server
10:19:48.734 INFO - Java: Oracle Corporation 23.21-b01
```

```
10:19:48.734 INFO - OS: Windows XP 5.1 x86
10:19:48.734 INFO - v2.33.0, with Core v2.33.0. Built from revision 4e90c97
    10:19:48.843 INFO - RemoteWebDriver instances should connect to: http://127.0.0.
1:4444/wd/hub
10:19:48.843 INFO - Version Jetty/5.1.x
10:19:48.843 INFO - Started HttpContext[/selenium-server/driver,/selenium-server
/driver]
10:19:48.843 INFO - Started HttpContext[/selenium-server,/selenium-server]
10:19:48.843 INFO - Started HttpContext[/,/]
10:19:48.890 INFO - Started org.openqa.jetty.jetty.servlet.ServletHandler@176343e
10:19:48.890 INFO - Started HttpContext[/wd,/wd]
    10:19:48.906 INFO - Started SocketListener on 0.0.0.0:4444
10:19:48.906 INFO - Started org.openqa.jetty.jetty.Server@388c74
-------------------------------------------------------------------------------
创建新 session
10:20:38.593 INFO - Executing: [new session: {platform=ANY, javascriptEnabled=tr
ue, browserName=chrome, version=}] at URL: /session)
10:20:38.593 INFO - Creating a new session for Capabilities [{platform=ANY, java
scriptEnabled=true, browserName=chrome, version=}]

webdrivr 通过 GET 方式发送请求
[0.921][INFO]: received Webriver request: GET /status

向 webdrver 返回响应，返回码200表示成功
[0.921][INFO]: sending Webriver response: 200 {
   "sessionId": "",
   "status": 0,
   "value": {
      "build": {
         "version": "alpha"      },
      "os": {
         "arch": "x86",
         "name": "Windows NT",
         "version": "5.1 SP3"     }
   }
}
```

webdriver 再次以 POST 方式发送请求，并启动浏览器相关信息

```
[0.984][INFO]: received Webriver request: POST /session {
   "desiredCapabilities": {
      "browserName": "chrome",
      "javascriptEnabled": true,
      "platform": "ANY",
      "version": ""   }
}
    [0.984][INFO]: Launching chrome: "C:\ocuments and Settings\Administrator\Local S
    ettings\Application ata\Google\Chrome\Application\chrome.exe" --remote-debugging
    -port=4223 --no-first-run --enable-logging --logging-level=1 --user-data-dir="C:
    \OCUME~1\AMINI~1\LOCALS~1\Temp\scoped_dir1808_7550"
    --load-extension="C:\OCUME~1\AMINI~1\LOCALS~1\Temp\scoped_dir1808_26821\internal"
    --ignore-certificate-error
s data:text/html;charset=utf-8,
[1.773][INFO]: sending Webriver response: 303
```

webdriver 再次以 GET 方法请求，这附加上了 session 的信息

```
[1.778][INFO]: received Webriver request: GET /session/32b33aa585ccbbf7ba7853588
2852af3
```

服务器先对 sesssionID 进行解析，确认是 selenium 调用的以及要访问的网址，

```
[1.779][INFO]: sending Webriver response: 200 {
   "sessionId": "32b33aa585ccbbf7ba78535882852af3",
   "status": 0,
   "value": {
      "acceptSslCerts": true,
      "applicationCacheEnabled": false,
      "browserConnectionEnabled": false,
      "browserName": "chrome",
      "chrome": {
         "chromedriverVersion": "2.0"      },
      "cssSelectorsEnabled": true,
      "databaseEnabled": true,
      "handlesAlerts": true,
      "javascriptEnabled": true,
      "locationContextEnabled": true,
```

```
      "nativeEvents": true,
      "platform": "Windows NT",
      "rotatable": false,
      "takesScreenshot": true,
      "version": "27.0.1453.116",
      "webStorageEnabled": true   }
}
10:20:40.640 INFO - Done: /session
10:20:40.640 INFO - Executing: org.openqa.selenium.remote.server.handler.GetSess
ionCapabilities@14cf7a1 at URL: /session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc)
10:20:40.640 INFO - Done: /session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc
    10:20:40.656 INFO - Executing: [get: http://www.youdao.com] at URL: /session/ac5
b2c71-5b1a-469e-814c-fdd09a2061fc/url)

webdriver 正试向服务器请求 youdao 网站
[1.820][INFO]: received Webriver request: POST /session/32b33aa585ccbbf7ba785358
82852af3/url {
   "url": "http://www.youdao.com"}
[1.822][INFO]: waiting for pending navigations...
[1.829][INFO]: done waiting for pending navigations
[2.073][INFO]: waiting for pending navigations...
[2.900][INFO]: done waiting for pending navigations

获得服务器数据的应答
[2.900][INFO]: sending Webriver response: 200 {
   "sessionId": "32b33aa585ccbbf7ba78535882852af3",
   "status": 0,
   "value": null}
10:20:41.734 INFO - Done: /session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/url
------------------------------------------------------------下面接着发送定位输入框的信息
10:20:41.734 INFO - Executing: [find element: By.name: q] at URL: /session/ac5b2
c71-5b1a-469e-814c-fdd09a2061fc/element)
[2.905][INFO]: received Webriver request: POST /session/32b33aa585ccbbf7ba785358
82852af3/element {
   "using": "name",
   "value": "q"}
```

```
[2.905][INFO]: waiting for pending navigations...
[2.905][INFO]: done waiting for pending navigations
[2.922][INFO]: waiting for pending navigations...
[2.922][INFO]: done waiting for pending navigations

得到服务器应答
[2.922][INFO]: sending Webriver response: 200 {
   "sessionId": "32b33aa585ccbbf7ba78535882852af3",
   "status": 0,
   "value": {
      "ELEMENT": "0.19427558477036655:1"   }
}
10:20:41.765 INFO - Done: /session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/element
10:20:41.765 INFO - Executing: [send keys: 0 org.openqa.selenium.support.events.
EventFiringWebDriver$EventFiringWebElement@a8215ba9, [h, e, l, l, o]] at URL: /s
ession/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/element/0/value)

向定位到的输入框写入 hello
[2.936][INFO]: received Webriver request: POST /session/32b33aa585ccbbf7ba785358
82852af3/element/0.19427558477036655:1/value {
   "id": "0.19427558477036655:1",
   "value": [ "h", "e", "l", "l", "o" ]
}
[2.936][INFO]: waiting for pending navigations...
[2.936][INFO]: done waiting for pending navigations
[3.002][INFO]: waiting for pending navigations...
[3.002][INFO]: done waiting for pending navigations
[3.002][INFO]: sending Webriver response: 200 {
   "sessionId": "32b33aa585ccbbf7ba78535882852af3",
   "status": 0,
   "value": null}
    10:20:41.843 INFO - Done: /session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/element/
0/value

再次发送定位输入框的请求
10:20:41.843 INFO - Executing: [find element: By.name: q] at URL: /session/ac5b2
```

```
c71-5b1a-469e-814c-fdd09a2061fc/element)
[3.006][INFO]: received Webriver request: POST /session/32b33aa585ccbbf7ba785358
82852af3/element {
   "using": "name",
   "value": "q"}
[3.006][INFO]: waiting for pending navigations...
[3.006][INFO]: done waiting for pending navigations
[3.016][INFO]: waiting for pending navigations...
[3.016][INFO]: done waiting for pending navigations
[3.016][INFO]: sending Webriver response: 200 {
   "sessionId": "32b33aa585ccbbf7ba78535882852af3",
   "status": 0,
   "value": {
      "ELEMENT": "0.19427558477036655:1"   }
}
10:20:41.859 INFO - Done: /session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/element
10:20:41.859 INFO - Executing: [send keys: 0 org.openqa.selenium.support.events.
EventFiringWebDriver$EventFiringWebElement@a8215ba9, [k, e, y, ., E, N, T, E, R]
] at URL: /session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/element/0/value)

对定位的到的输入框发送回车（ENTER）事件请求
[3.021][INFO]: received Webriver request: POST /session/32b33aa585ccbbf7ba785358
82852af3/element/0.19427558477036655:1/value {
   "id": "0.19427558477036655:1",
   "value": [ "k", "e", "y", ".", "E", "N", "T", "E", "R" ]
}
[3.021][INFO]: waiting for pending navigations...
[3.021][INFO]: done waiting for pending navigations
[3.064][INFO]: waiting for pending navigations...
[3.064][INFO]: done waiting for pending navigations
[3.064][INFO]: sending Webriver response: 200 {
   "sessionId": "32b33aa585ccbbf7ba78535882852af3",
   "status": 0,
   "value": null}
    10:20:41.906 INFO - Done: /session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/element/
```

```
0/value
10:20:41.906 INFO - Executing: [close window] at URL:
/session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/window)
[3.068][INFO]: received Webriver request: ELETE /session/32b33aa585ccbbf7ba78535
882852af3/window
[WARNING:chrome_desktop_impl.cc(88)] chrome detaches, user should take care of d
irectory:C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\scoped_dir1808_7550 and C:\DOCUME~1\
ADMINI~1\LOCALS~1\Temp\scoped_dir1808_26821
[5.318][INFO]: sending Webriver response: 200 {
   "sessionId": "32b33aa585ccbbf7ba78535882852af3",
   "status": 0,
   "value": null}
    10:20:44.156 INFO - Done: /session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/window
```

扩展资料：

rt sm eyiselenium 与 webdriver 的关系：
http://v.qq.com/boke/page/j/v/v/j01135krrvv.html

lazyman 快速入门：
http://v.qq.com/boke/page/i/k/a/i0113wompka.html

关于 python 自动化的博客，慢慢研读：
http://www.cnblogs.com/hzhida/archive/2012/08/13/2637089.html

splinter 自动化框架：
http://splinter.cobrateam.info/docs/why.html

http://v.qq.com/boke/page/s/8/3/s0114uu1d83.html。
大家可以了解一下 webdriver guide 的内容

webdriver API 地址：
https://github.com/easonhan007/webdriver_guide

robot framework

自动化测试框架，后序研究。
RF 框架系列文章
http://www.51testing.com/?21116/
http://blog.csdn.net/tulituqi/article/category/897484/2

安装：http://blog.sina.com.cn/s/blog_654c6ec70100tkxn.html

selenium webdriver py 文档
http://selenium.googlecode.com/git/docs/api/py/index.html

seleniumwrapper 0.5.3

https://pypi.python.org/pypi/seleniumwrapper

selenium webdriver 系列教程
http://blog.csdn.net/nbkhic/article/details/6896889

文档
http://selenium.googlecode.com/git/docs/api/py/index.html