

selenium webdriver (python)

(第三版)

声明:

本文档以收费方式出售, 未经过作者本人允许, 禁止一切非法的传播, 禁止用于任何商业用途。

博客园--虫师
fnngj@126.com

前言

对于大多软件测试人员来讲缺乏编程经验（指项目开发经验，大学的 C 语言算很基础的编程知识）一直是难以逾越的鸿沟，并不是说测试比开发人员智商低，是国内的大多测试岗位是功能测试为主，在工作时间中，我们很难深入的接触和使用编程技术；

笔者认为自动化测试尽管有很多不足，更不能完全替代手工测试，但确实是测试人员发展的一个方向，越来越多的公司在实践自动化，越来越多的项目在尝试自动化；所以对于功能测试人员来讲，掌握项目自动化测试技术自然能提高测试技术水平，能够保持不被淘汰，又能在激烈的竞争中处于优势地位。

为什么选 python，因为他语法简单；如果你有一点 C 语言或 java 语言基础的，将会非常容易地学会并使用 python。自动化脚本本身要比开发程序简单得多，大多人学编程半途而废就是没有实践的机会；那么通过 selenium webdriver python 进行自动化测试，很快就可以学以致用，建立继续学习的信心与动力；可以平滑的过渡到真正的编程经验上。

如果要使用 java 或 ruby 语言通过 selenium webdriver 来实施自动化测试，虽然各种语言的语法有差别，但思路是相通的；相信本文档依然可以提供给你学习的思路。

继续在这里感谢：

感谢购买第二版的同学，谢谢你们对本人劳动成果的支持！也正是你们时常问我还出不出第三版了，也是你们的鼓励，让我继续学习整理本文档。

感谢乙醇前辈，第二版的文档是放在他的淘宝网站上卖的，感谢他的帮忙。

最最感谢的还是兔子（MarkRabbit），好吧！他已经极力抗议叫兔子了，哈哈！本文档中相当多的知识点是他提供的，不过他只提供思路，不提供解决问题的具体代码；我需要把他的截图下来，反复理解，然后找具体的解决代码，因此，我 python 的语言能力提高了不少。

下面要简单说说本文档的内容：

《selenium webdriver python (第三版)》相比第二版增加测试套件，参数化问题，引入 HTMLTestRunner，测试结构的调整，相对来说比较好的构建了测试结构（只能说是“结构”，离“架构”还差得呢！）

本文档仍然有很多不足，毕竟不是以出书的标准来要求的，可能很多知识点解释的不透彻，甚至错误的地方，请提出你的意见给本人。

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>

2013.10.31

目录

一、selenium+python 环境搭建.....	6
1.1 selenium 介绍.....	6
1.2 准备工作.....	6
1.3 安装步骤.....	7
1.4 安装 chrome driver.....	8
1.5 安装 IE driver.....	9
二、开始第一个脚本.....	9
2.1 为什么选 python.....	9
2.2 第一个脚本.....	9
2.3 脚本解析.....	10
三、元素的定位.....	11
3.1 id 和 name 定位.....	12
3.2 tag name 和 class name 定位.....	12
3.3 CSS 定位.....	13
3.4 XPath 定位.....	14
3.5 link 定位.....	15
3.6 Partial link text 定位.....	15
四、添加等待时间.....	15
4.1、添加休眠.....	15
4.2、智能等待.....	16
五、打印信息.....	17
5.1、打印 title.....	17
5.2、打印 URL.....	17
六、浏览器的操作.....	18
6.1、浏览器最大化.....	18
6.2、设置浏览器宽、高.....	19
七、操作浏览器的前进、后退.....	19
八、操作测试对象.....	20
8.1、鼠标点击与键盘输入.....	21
8.2、submit 提交表单.....	21
8.3、text 获取元素文本.....	22
8.4、get_attribute 获得属性值.....	22
九、键盘事件.....	23
9.1、键盘按键用法.....	23
9.2、键盘组合键用法.....	24
9.3、中文乱码问题.....	25
十、鼠标事件.....	25
10.1、鼠标右键.....	26

10.2、鼠标双击.....	27
10.3、鼠标拖放.....	27
十一、定位一组元素.....	28
11.1、第一种定位方法.....	30
11.2、第二种定位方法.....	31
11.3、去掉最后一个勾选.....	31
十二、多层框架/窗口定位.....	32
12.1、多层框架定位.....	32
12.2、多层窗口定位.....	35
十三、层级定位.....	35
十四、上传文件操作.....	38
14.1、操作文件上传例子.....	39
14.2、139 邮箱上传.....	40
十五、下拉框处理.....	41
15.1、操作下拉框例子.....	41
15.2、百度搜索设置下拉框操作.....	43
十六、alert、confirm、prompt 的处理.....	44
十七、对话框的处理.....	45
17.1、div 对话框的处理.....	45
17.2、一般对话框的处理.....	48
十八、调用 js.....	49
18.1、通过 js 隐藏元素.....	49
18.2、通过 js 使输入框标红.....	51
十九、控制浏览器滚动条.....	52
19.1、场景一.....	53
19.2、场景二.....	53
二十、cookie 处理.....	54
20.1、打印 cookie 信息.....	54
20.2、对 cookie 操作.....	55
20.3、博客园登陆分析 cookie.....	56
二十一、webdriver 原理解析.....	57
二十二、引入 unittest 框架.....	65
二十三、unittest 单元测试框架解析.....	70
二十四、批量执行测试集.....	75
二十五、异常捕捉与错误截图.....	77
二十六、生成测试报告(HTMLTestRunner).....	80
二十七、数据驱动测试.....	83
27.1、读取文件参数化.....	83
27.2、用户名密码的参数化（读取文件）.....	85
27.3、用户名的参数化（字典）.....	86
27.4、用户名密码的参数化（函数）.....	87
二十八、测试套件.....	89
28.1、测试套件实例.....	89
28.2、整合 HTMLTestRunner 测试报告.....	93

28.3、更易读的报告.....	95
二十九、结构改进.....	96
29.1、all_tests.py 移出来.....	96
29.2、__init__.py 文件解析.....	97
29.3、调用多级目录的用例.....	98
29.4、改进用例的读取.....	99
29.5、进一步分离用例列表.....	101
三十、UliPad--python 开发利器.....	103

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

一、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

二、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

三、安装 pip，我默认解压在了 C:\pip-1.3.1 目录下

四、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

五、再切换到 C:\Python27\Scripts 目录下输入：

```
C:\Python27\Scripts > easy_install pip
```

六、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。

注意：七、八两步可以暂不进行，如果你要学习第二十一章 webdriver 原理的时候再进行也不迟。

=====

七、下载并安装

(http://www.java.com/zh_CN/download/chrome.jsp?locale=zh_CN), 什么!?! 你没整过 java 虚拟机, 百度一下 java 环境搭建吧。

八、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 port 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，有洁癖用这个。

```
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。

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

```
#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 xpath 方式定位
browser.find_element_by_xpath("//input[@id='kw']").send_keys("selenium")

#####
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("selenium")
#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 ，定位它里面的第2个 td

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)`

隐式地等待一个元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 title

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
- text 用于获取元素的文本信息

8.1、鼠标点击与键盘输入

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").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。

8.2、submit 提交表单

我们把“百度一下”的操作从 click 换成 submit 可以达到相同的效果：

```
#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()

```

8.3、text 获取元素文本

text 用于获取元素的文本信息

下面把百度首页底部的声明打印输出

```

#coding=utf-8
from selenium import webdriver

import time

driver = webdriver.Firefox()
driver.get("http://www.baidu.com")
time.sleep(2)

#id = cp 元素的文本信息
data=driver.find_element_by_id("cp").text
print data #打印信息
time.sleep(3)

driver.quit()

```

输出:

```

>>>
©2013 Baidu 使用百度前必读 京 ICP 证030173号

```

8.4、get_attribute 获得属性值

get_attribute

获得属性值。

这个函数的用法前面已经有出现过，在定位一组元素的时候有使用到它，只是我们没有做过多的解释。

一般用法：

```
select = driver.find_element_by_tag_name("select")

allOptions = select.find_elements_by_tag_name("option")

for option in allOptions:

    print "Value is: " + option.get_attribute("value")

    option.click()

.....
```

具体应用参考第十一节[层级定位](#)例子。

九、键盘事件

本章重点：

- 键盘按键用法
- 键盘组合键用法
- send_keys() 输入中文乱码问题

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("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)

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，还需要填写验证码，那么回车也起不到登陆的效果。

9.2、键盘组合键用法

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

#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。

9.3、中文乱码问题

selenium2 python 在 send_keys() 中输入中文一直报错，其实前面加个小 u 就解决了：

```
coding=utf-8
```

```
send_keys(u"输入中文")
```

需要注意的是 utf-8 并不是万能的，我们需要保持脚本、浏览器、程序三者编码之间的转换；如果 utf-8 不能解决，可以尝试 GBK 或修改浏览器的默认编码。

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

```

        <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>

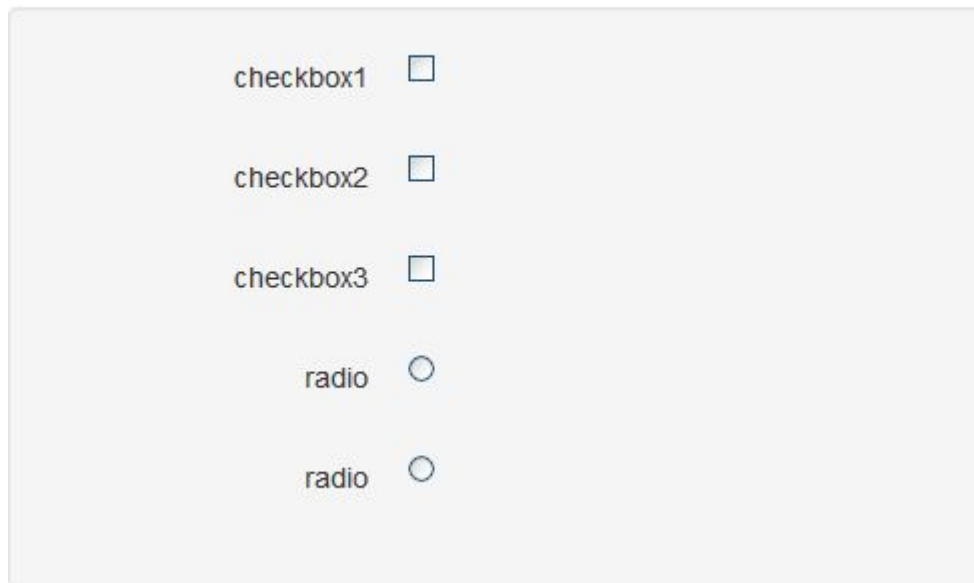
    <div class="control-group">
        <label class="control-label" for="r">radio</label>
        <div class="controls">
            <input type="radio" id="r2" />
        </div>
    </div>
</form>
</div>
</body>
</html>

```

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）

通过浏览器打开，得到下列页面：

checkbox



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()

import os
```

注意：因为我们调用的是本地文件，所以要导入 os 包。

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，selenium 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，通过

浏览器打开，得到下列页面：



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 iframe (id = f1)
browser.switch_to_frame("f1")
#再找到其下面的 iframe (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()
```

12.2、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"></scri
pt>
    <link
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-c
omcombined.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 notepad ++ 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：

Level locate



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。

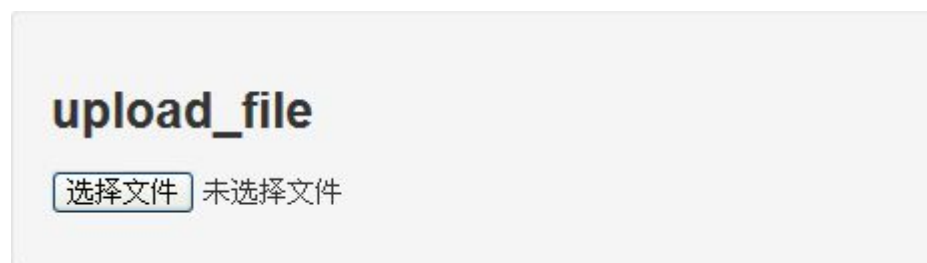
其实，在 `selenium webdriver` 没我们想的那么复杂；只要定位上传按钮，通过 `send_keys` 添加本地文件路径就可以了。绝对路径和相对路径都可以，关键是上传的文件存在。下面通地例子演示。

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>
```

通过浏览器打开，得到下列页面：



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_case\\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>
```

```

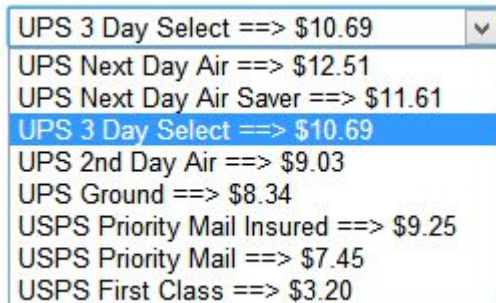
</select>

</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

```
#!/usr/bin/env python
#-*-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、alert、confirm、prompt 的处理

本节重点：

- `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。

在实际的应用中，我们会碰到各种交互的弹窗，在上面百度搜索设置的例子中，我们用 `switch_to_alert()` 处理警告框非常简单；其实，对于原生的 `js alert`、`confirm` 以及 `prompt` 都可以通过 `webdriver` 的 `switch_to_alert()` 方法进行处理。

比较常见下的就是下面这种类型的确认框：



```
selenium.webdriver.remote.webdriver.switch_to_alert()
```

将焦点切换到页面上的警报

Usage:

```
driver.switch_to_alert()
```

由于用法简单，这里就不给具体例子了，在实际应用中：

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

#得到文本信息打印
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
alert = driver.switch_to_alert()
alert.send_keys("xxx")
```

十七、对话框的处理

本节重点：

- 打开对话框
- 关闭对话框
- 操作对话框中的元素
- `current_window_handle` 获得当前窗口
- `window_handles` 获得所有窗口

更多的时候我们在实际的应用中碰到的并不是简单警告框，而是提供更多功能的会话框。

17.1、div 对话框的处理

modal.html

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>modal</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(){
        $('#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">×</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>
</html>
```

```

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>modal</title>
    <script
      type="text/javascript"
      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() {
        $('#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">×</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>
</html>

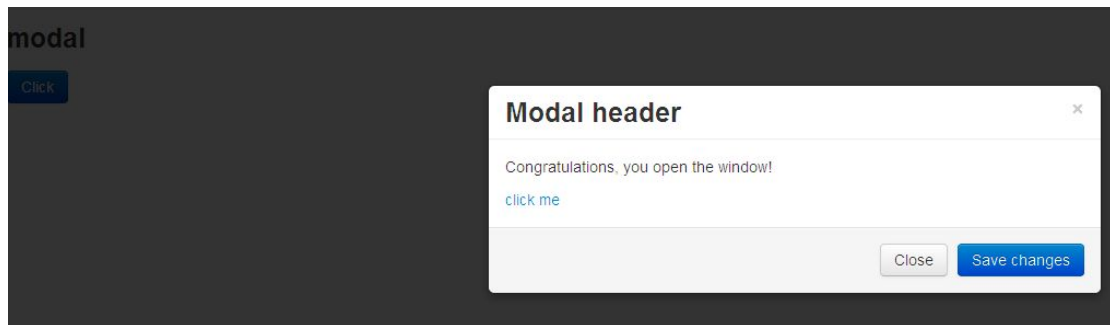
```

```

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

```

代码有点长，你可以直接赋值粘贴到 Notepad++ 中，保存成 html 通过浏览器打开，效果如下：



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

if 'HTTP_PROXY' in os.environ: del os.environ['HTTP_PROXY']

dr = webdriver.Firefox()

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

```



```
# 由于对话框中的元素被蒙板所遮挡，直接点击会报 Element is not clickable 的错误
# 所以使用 js 来模拟 click

link = dr.find_element_by_id('myModal').find_element_by_id('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()

dr.quit()
```

17.2、一般对话框的处理

有些弹出对话框窗，我们可以通过判断是否为当前窗口的方式进行操作。

```
#获得当前窗口

nowhandle=driver.current_window_handle

#打开弹窗

driver.find_element_by_name("xxx").click()

#获得所有窗口

allhandles=driver.window_handles

for handle in allhandles:

    if handle!=nowhandle:    #比较当前窗口是不是原先的窗口

        driver.switch_to_window(handle)    #获得当前窗口的句柄

        dirver.find_element_by_class_name("xxxx").click()    #在当前窗口操作
```

```
#回到原先的窗口

driver.switch_to_window(nowhandle)
```

这里只是操作窗口的代码片段，提供一个思路，能否完成我们想要的结果，还需要我们通过实例去验证。

十八、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')`

18.1、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
```

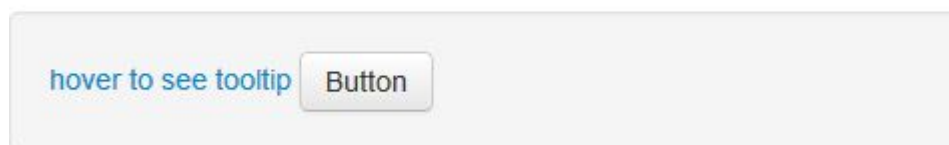
```
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
<script type="text/javascript"> $(document).ready(function() {
    $('#tooltip').tooltip({"placement": "right"});
});
</script>
</head>

<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>
```

(保持 html 文件与执行脚本在同一目录下)

保存并通过浏览器打开，如下：

js



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()
```

18.2、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.kuai
bo.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

十九、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。

19.1、场景一

先来解决场景第一个问题，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)
```

19.2、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()
```

二十、cookie 处理

本节重点:

- driver.get_cookies () 获得 cookie 信息
- add_cookie(cookie_dict) 向 cookie 添加会话信息
- delete_cookie(name) 删除特定(部分)的 cookie
- delete_all_cookies() 删除所有 cookie

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着某个 cookie 信息, webdriver 可以帮助我们读取、添加, 删除 cookie 信息。

20.1、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:

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

```
u'abcUX9zdw0minadIhtvcu', u'secure': False}]
```

20.2、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
```


20.3、博客园登陆分析 cookie

通过博客园登陆来分析 cookie

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

driver = webdriver.Firefox()

driver.get("http://passport.cnblogs.com/login.aspx?ReturnUrl=http://www.cnblogs.com/fnng/admin/EditPosts.aspx")

time.sleep(3)
driver.maximize_window() # 浏览器全屏显示

#通过用户名密码登陆
driver.find_element_by_id("tbUserName").send_keys("fnngj")
driver.find_element_by_id("tbPassword").send_keys("123456")

#勾选保存密码
driver.find_element_by_id("chkRemember").click()
time.sleep(3)

#点击登陆按钮
driver.find_element_by_id("btnLogin").click()

#获取 cookie 信息并打印
cookie= driver.get_cookies()
print cookie

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

运行打印信息:

```
#第一次执行信息
>>>
[{'u'domain':    u'.cnblogs.com',    u'name':    u'.DottextCookie',    u'value':
u'C709F15A8BC0B3E8D9AD1F68B371053849F7FEE31F73F1292A150932FF09A7B0D4A1B449A3
2A6B24AD986CDB05B9998471A37F39C3B637E85E481AA986D3F8C187D7708028F9D4ED3B326B
46DC43B416C47B84D706099ED1D78B6A0FC72DCF948DB9D5CBF99D7848FDB78324',
```

```
u'expiry': None, u'path': u '/', u'secure': False}}
>>> ===== RESTART =====
#第二次执行信息
>>>
[{'u'domain': u'.cnblogs.com', u'name': u'.DottextCookie', u'value':
u'5BB735CAD62E99F8CCB9331C32724E2975A0150D199F4243AD19357B3F99A416A93B2E803F
4D5C9D065429713BE8B5DB4ED760EDCBAF492EABE2158B3A6FBBEA2B95C4DA3D2EFEADACC324
7040906F1462731F652199E2A8BEFD8A9B6AAE87CF3059A3CAEB9AB0D8B1B7AD2A',
u'expiry': 1379502502, u'path': u '/', u'secure': False}}
>>>
```

第一次注释掉勾选保存密码的操作，第二次通过勾选保存密码获得 cookie 信息；来看两次运行结果的 cookie 的何不同：

u'expiry': None

u'expiry': 1379502502

通过对比发现，不勾选保存密码时 expiry 的值为 none；那么就可以初步判断勾选保存密码的操作在 cookie 中起到了作用。至于是否准确可以再做进一步的分析。

二十一、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，在代码前加上 \$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），来转化浏览器 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=true, browserName=chrome, version=}] at URL: /session)
10:20:38.593 INFO - Creating a new session for Capabilities [{platform=ANY, javascriptEnabled=true, browserName=chrome, version=}]
```

webdriver 通过 GET 方式发送请求

```
[0.921][INFO]: received Webriver request: GET /status
```

向 webdriver 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:\Documents 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 sessionID 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.GetSessionCapabilities@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/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/url)
```

webdriver 正试向服务器请求 youdao 网站

```
[1.820][INFO]: received Webriver request: POST /session/32b33aa585ccbbf7ba78535882852af3/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/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/element)
```

```
[2.905][INFO]: received Webriver request: POST /session/32b33aa585ccbbf7ba78535882852af3/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 {
```



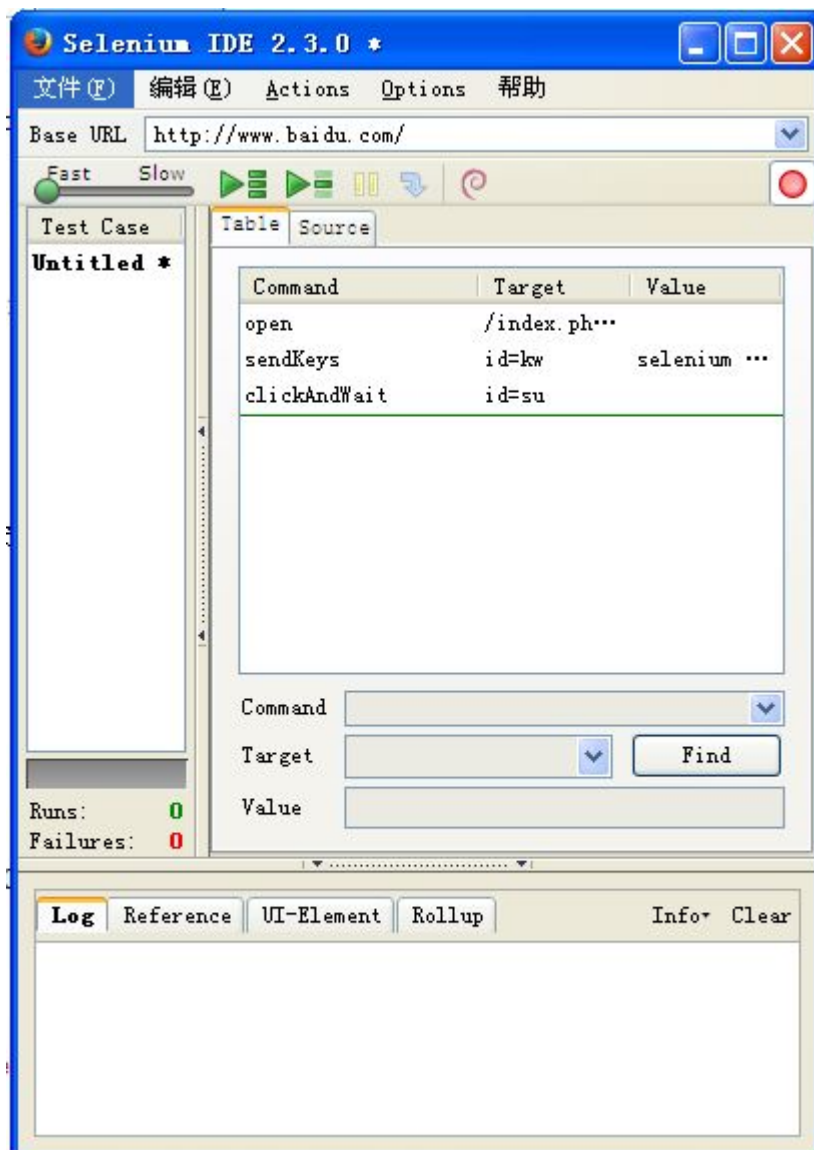
```
"sessionId": "32b33aa585ccbbf7ba78535882852af3",  
"status": 0,  
"value": null}  
10:20:44.156 INFO - Done: /session/ac5b2c71-5b1a-469e-814c-fdd09a2061fc/window
```

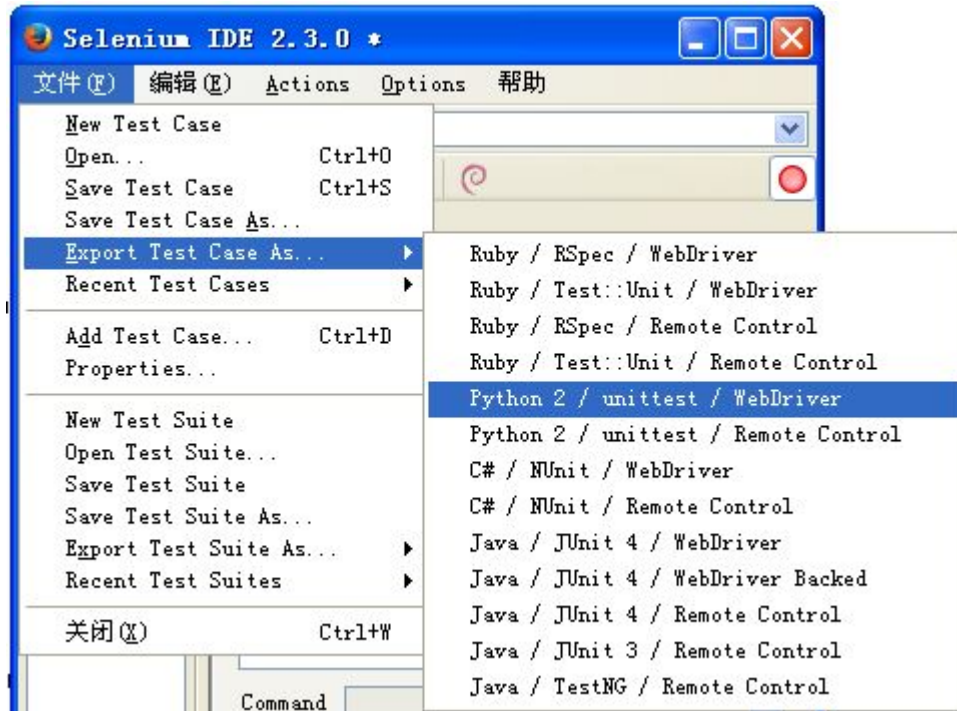
第二部分：框架的力量

二十二、引入 unittest 框架

unittest 框架学习

借助 IED 录制脚本，





将脚本导出，保存为 baidu.py，通过 python IDLE 编辑器打开。如下：

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
import unittest, time, re

class Baidu(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.base_url = "http://www.baidu.com/"
        self.verificationErrors = []
        self.accept_next_alert = True

    def test_baidu(self):
        driver = self.driver
        driver.get(self.base_url + "/")
        driver.find_element_by_id("kw").send_keys("selenium webdriver")
        driver.find_element_by_id("su").click()
        driver.close()

    def is_element_present(self, how, what):
```

```

    try: self.driver.find_element(by=how, value=what)
    except NoSuchElementException, e: return False
    return True

def is_alert_present(self):
    try: self.driver.switch_to_alert()
    except NoAlertPresentException, e: return False
    return True

def close_alert_and_get_its_text(self):
    try:
        alert = self.driver.switch_to_alert()
        alert_text = alert.text
        if self.accept_next_alert:
            alert.accept()
        else:
            alert.dismiss()
        return alert_text
    finally: self.accept_next_alert = True

def tearDown(self):
    self.driver.quit()
    self.assertEqual([], self.verifyErrors)

if __name__ == "__main__":
    unittest.main()

```

加入 `unittest` 框架后，看上去比我们之前见的脚本复杂了很多，除了中间操作浏览器的几行，其它都看不懂，不要急，我们来分析一下~！

```
import unittest
```

想使用 `unittest` 框架，首先要引入 `unittest` 包，这个不多解释。

```
class Baidu(unittest.TestCase):
```

`Baidu` 类继承 `unittest.TestCase` 类，从 `TestCase` 类继承是告诉 `unittest` 模块的方式，这是一个测试案例。

```
def setUp(self):
    self.driver = webdriver.Firefox()
```

```
self.base_url = "http://www.baidu.com/"
```

setUp 用于设置初始化的部分，在测试用例执行前，这个方法中的函数将先被调用。这里将浏览器的调用和 URL 的访问放到初始化部分。

```
self.errors = []
```

脚本运行时，错误的信息将被打印到这个列表中。

```
self.accept_next_alert = True
```

是否继续接受下一警告（字面意思，没找到解释！）

```
def test_baidu(self):
    driver = self.driver
    driver.get(self.base_url + "/")
    driver.find_element_by_id("kw").send_keys("selenium webdriver")
    driver.find_element_by_id("su").click()
```

test_baidu 中放置的就是我们的测试脚本了，这部分我们并不陌生；因为我们执行的脚本就在这里。

```
def is_element_present(self, how, what):
    try: self.driver.find_element(by=how, value=what)
    except NoSuchElementException, e: return False
    return True
```

is_element_present 函数用来查找页面元素是否存在，在这里用处不大，通常删除。因为判断页面元素是否存在一般都加在 **testcase** 中。

```
def is_alert_present(self):
    try: self.driver.switch_to_alert()
    except NoAlertPresentException, e: return False
    return True
```

对弹窗异常的处理



```
def close_alert_and_get_its_text(self):
    try:
```

```

        alert = self.driver.switch_to_alert()
        alert_text = alert.text
        if self.accept_next_alert:
            alert.accept()
        else:
            alert.dismiss()
        return alert_text

    finally: self.accept_next_alert = True

```

关闭警告和对得到文本框的处理，如果不熟悉 **python** 的异常处理和 **if** 语句的话，请去补基础知识，这里不多解释。

```

def tearDown(self):
    self.driver.quit()

    self.assertEqual([], self.errors)

```

tearDown 方法在每个测试方法执行后调用，这个地方做所有清理工作，如退出浏览器等。

self.assertEqual([], self.errors)

这个是难点，对前面 **errors** 方法获得的列表进行比较；如查 **errors** 的列表不为空，输出列表中的报错信息。

而且，这个东西，也可以将来被你自己更好的调用和使用，根据自己的需要写入你希望的信息。（**rabbit** 告诉我的）

```

if __name__ == "__main__":
    unittest.main()

```

unittest.main()函数用来测试 类中以 **test** 开头的测试用例

这样一一分析下来，我们对 **unittest** 框架有了初步的了解。运行脚本，因为引入了 **unittest** 框架，所以控制台输出了脚本执行情况的信息。

```

>>> ===== RESTART =====
>>>
.
-----
Ran 1 test in 10.656s

OK
>>>

```

很帅吧!? 后面将以 unittest 为基础, 向新的征程进发~!

二十三、unittest 单元测试框架解析

上一节只是从自动化测试的角度简单分析了一下 unittest , 这一节从 python 的单元测试框架的角度再学习一下 **unittest** 框架 (又名 PyUnit 框架)

(好好学, 这一章整不明白, 后面的技术就别玩了!)

widget.py---被测试类

```
#coding= utf-8

# 将要被测试的类
class Widget:

    def __init__(self, size = (40, 40)):
        self._size = size

    def getSize(self):
        return self._size

    def resize(self, width, height):
        if width < 0 or height < 0:
            raise ValueError, "illegal size"
        self._size = (width, height)

    def dispose(self):
        pass
```

auto.py---测试类

```
#coding= utf-8

from widget import Widget
import unittest

# 执行测试的类
class WidgetTestCase(unittest.TestCase):
```

```
def setUp(self):
    self.widget = Widget()

def testSize(self):
    self.assertEqual(self.widget.getSize(), (40, 40))

def tearDown(self):
    self.widget = None

# 构造测试集
def suite():
    suite = unittest.TestSuite()
    suite.addTest(WidgetTestCase("testSize"))
    return suite

# 测试
if __name__ == "__main__":
    unittest.main(defaultTest = 'suite')
```

- 用 `import` 语句引入 `unittest` 模块
- 让所有执行测试的类都继承于 `TestCase` 类，可以将 `TestCase` 看成是对特定类进行测试的方法的集合
- `setUp()` 方法中进行测试前的初始化工作，`tearDown()` 方法中执行测试后的清除工作。`setUp()` 和 `tearDown()` 都是 `TestCase` 类中定义的方法
- 在 `testSize()` 中调用 `assertEqual()` 方法，对 `Widget` 类中 `getSize()` 方法的返回值和预期值进行比较，确保两者是相等的，`assertEqual()` 也是 `TestCase` 类中定义的方法。
- 提供名为 `suite()` 的全局方法，`PyUnit` 在执行测试的过程调用 `suit()` 方法来确定有多少个测试用例需要被执行，可以将 `TestSuite` 看成是包含所有测试用例的一个容器。

框架分析

软件测试中最基本的组成是单元测试用例（`test case`），我们在实际测试过程中，不可能真对一个功能（类）只写一个用例。`TestCase` 在 `PyUnit` 测试框架中被视为测试单元的运行实体，`Python` 程序员可以通过它派生自定义的测试过程与方法（测试单元），利

用 Command 和 Composite 设计模式，多个 TestCase 还可以组合成测试用例集合。

编写测试用例

采用 PyUnit 提供的动态方法，只编写一个测试类来完成对整个软件模块的测试，这样对象的初始化工作可以在 setUp() 方法中完成，而资源的释放则可以在 tearDown() 方法中完成。

对的 widget.py 被测试类的多方法进行测试

```
# 执行测试的类
class WidgetTestCase(unittest.TestCase):

    def setUp(self):
        self.widget = Widget()

    # 测试 getSize() 方法的测试用例
    def testSize(self):
        self.assertEqual(self.widget.getSize(), (40, 40))

    # 测试 resize() 方法的测试用例
    def testResize(self):
        self.widget.resize(100, 100)
        self.assertEqual(self.widget.getSize(), (100, 100))

    def tearDown(self):
        self.widget.dispose()
        self.widget = None
```

我们可以在一个测试类中，写多个测试用例对被测试类的方法进行测试。

组织用例集

完整的单元测试很少只执行一个测试用例，开发人员通常都需要编写多个测试用例才能对某一软件功能进行比较完整的测试，这些相关的测试用例称为一个测试用例集，在 PyUnit 中是用 TestSuite 类来表示的。

可以在单元测试代码中定义一个名为 suite() 的全局函数，并将其作为整个单元测试

的入口，PyUnit 通过调用它来完成整个测试过程。

```
def suite():
    suite = unittest.TestSuite()
    suite.addTest(WidgetTestCase("testSize"))
    suite.addTest(WidgetTestCase("testResize"))
    return suite
```

如果用于测试的类中所有的测试方法都以 test 开头，Python 程序员甚至可以用 PyUnit 模块提供的 makeSuite() 方法来构造一个。

```
def suite():
    return unittest.makeSuite(WidgetTestCase, "test")
```

TestSuite 类可以看成是 TestCase 类的一个容器，用来对多个测试用例进行组织，这样多个测试用例可以自动在一次测试中全部完成。

运行测试集

PyUnit 使用 TestRunner 类作为测试用例的基本执行环境，来驱动整个单元测试过程。Python 开发人员进行单元测试时一般不直接使用 TestRunner 类，而是使用其子类 TextTestRunner 来完成测试，并将测试结果以文本方式显示出来：

```
runner = unittest.TextTestRunner()
runner.run(suite)
```

对 widget.py 被测试类，下面通过 PyUnit 编写完整的单元测试用例：

text_runner.py

```
#coding=utf-8
from widget import Widget
import unittest
# 执行测试的类
class WidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget()

    def tearDown(self):
        self.widget.dispose()
        self.widget = None
```

```
def testSize(self):
    self.assertEqual(self.widget.getSize(), (40, 40))

def testResize(self):
    self.widget.resize(100, 100)
    self.assertEqual(self.widget.getSize(), (100, 100))
# 测试
if __name__ == "__main__":
    # 构造测试集
    suite = unittest.TestSuite()
    suite.addTest(WidgetTestCase("testSize"))
    suite.addTest(WidgetTestCase("testResize"))

    # 执行测试
    runner = unittest.TextTestRunner()
    runner.run(suite)
```

PyUnit 模块中定义了一个名为 `main` 的全局方法，使用它可以很方便地将一个单元测试模块变成可以直接运行的测试脚本，`main()` 方法使用 `TestLoader` 类来搜索所有包含在该模块中的测试方法，并自动执行它们。如果 Python 程序员能够按照约定（以 `test` 开头）来命名所有的测试方法，那就只需要在测试模块的最后加入如下几行代码即可：

```
#coding=utf-8
from widget import Widget
import unittest
# 执行测试的类
class WidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget()

    def tearDown(self):
        self.widget.dispose()
        self.widget = None

    def testSize(self):
        self.assertEqual(self.widget.getSize(), (40, 40))

    def testResize(self):
        self.widget.resize(100, 100)
        self.assertEqual(self.widget.getSize(), (100, 100))
# 测试
```

```
if __name__ == "__main__":
    unittest.main()
```

二十四、批量执行测试集

有了上面对 `unittest` 框架的学习作铺垫，下面我们就可以将多个自动化用例用到一起执行。

```
#coding=utf-8
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
import unittest, time, re

class Baidu(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.base_url = "http://www.baidu.com/"
        self.verificationErrors = []
        self.accept_next_alert = True

    #百度搜索用例
    def test_baidu_search(self):
        driver = self.driver
        driver.get(self.base_url + "/")
        driver.find_element_by_id("kw").send_keys("selenium webdriver")
        driver.find_element_by_id("su").click()
        time.sleep(2)
        driver.close()

    #百度设置用例
    def test_baidu_set(self):
        driver = self.driver
```

```
#进入搜索设置页
driver.get(self.base_url + "/gaoji/preferences.html")

#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()

def tearDown(self):
    self.driver.quit()
    self.assertEqual([], self verificationErrors)

if __name__ == "__main__":
    unittest.main()
```

虽然已经实例了多个用例一起跑，但这样仍然不合理，几个用例一起执行还好，如果几十个、几百个的用例的话，这个文件将变得无比庞大，不利于维护。

所以，做合理做法是一个例一个文件，把所文件放一个文件夹下，通过单独脚本控制所有用例的执行，将脚本的执行结果输出到一个 log 文件中。

初步把框架走通了。



单个用例相信你早就会写了，把他们整理一下放到一个文件夹下，然后编写执行用例集脚本：

```
test_case_.py
#-*-coding=utf-8 -*-
import os
#列出某个文件夹下的所有 case,这里用的是 python, 所在 py 文件运行一次后会生成一个 pyc
的副本
caselist=os.listdir('D:\\selenium_use_case\\test_case')
for a in caselist:
    s=a.split('.')[1:][0] #选取所要执行的用例
    if s=='py':
        #此处执行 dos 命令并将结果保存到 log.txt
        os.system('D:\\selenium_use_case\\test_case\\%s 1>>log.txt 2>&1'%a)
```

查看 log.txt 文件:

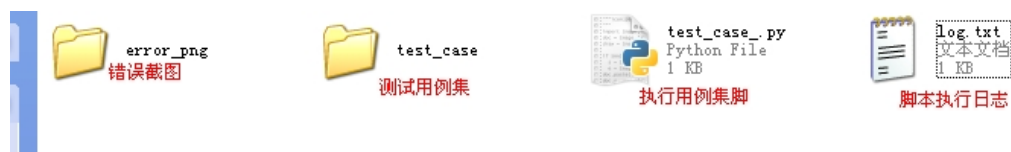
```
..
-----
Ran 2 tests in 32.469s

OK
..
-----
Ran 2 tests in 27.016s

OK
```

二十五、异常捕捉与错误截图

创建错误截图文件夹，目录结果如下：



用例不可能每一次运行都成功，肯定运行时候有不成功的时候，换句话说，我们不需要永远都运行成功的用例，他本身是没有什么意义的。关键是我们捕捉到错误，并以把并错误

截图保存，这将是一个非常棒的功能，也会给我们错误定位带来方便。

baidu.py

```
#coding=utf-8
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
import unittest, time, re

class Baidu(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.base_url = "http://www.baidu.com/"
        self.verificationErrors = []
        self.accept_next_alert = True

    #百度搜索用例
    def test_baidu_search(self):
        driver = self.driver
        driver.get(self.base_url + "/")

        try:
            #kwddd 是一个无法找到的元素 id
            driver.find_element_by_id("kwddd").send_keys("selenium webdriver")
        except:
            driver.get_screenshot_as_file("D:\\selenium_use_case\\error_png\\kw.png")
            #如果没有找到上面的元素就截取当前页面。

        driver.find_element_by_id("su").click()
        time.sleep(2)
        driver.close()

    def tearDown(self):
        self.driver.quit()
        self.assertEqual([], self.verificationErrors)

if __name__ == "__main__":
    unittest.main()
```

这里特意把脚本写错误的，使脚本找不到 id 为 kwddd 的元素，通过 try....except...对

异常进行捕捉；并把结果保存下来。再次执行你的脚本会发现 `error_png` 目录下面产生了错误时候的截图。



截图函数 `get_screenshot_as_file`

`selenium.webdriver.remote.webdriver.get_screenshot_as_file(filename)`

截图当前窗口图片。如果有任何 `IOError` 将返回 `false`，否则将返回 `Ture`。

`filename`: 指定错误截图的存放路径及图片名。

用法:

```
driver.get_screenshot_as_file('/Screenshots/foo.png')
```

我们需要用 `python` 这门语言去调用 `selenium` 的一些工具来操作浏览器，帮助我们实现“web UI”自动化。

=====华丽分割线=====

下面的内容为本文档第三版的内容，后面的学习重点就不是通过 `webdriver` 如何操作页面元素了，我们的关注点将转移到框架上，如何 `python` 语言使我们的框架实现更强大的功能，我在后面的章节学习与整理的过程中，也补充了不少 `python` 知识，建议读者最好掌握一些 `python` 编程基础。

二十六、生成测试报告(HTMLTestRunner)

在脚本运行完成之后，除了在 log.txt 文件看到运行日志外，我们更希望能生一张漂亮的测试报告来展示用例执行的结果。

下面我们就通过 HTMLTestRunner.py 来生成测试报告。

首先要下 HTMLTestRunner.py 文件，下载地址：

<http://tungwaiyip.info/software/HTMLTestRunner.html>

将下载的文件放入...\Python27\Lib 目录下（windows），打开交互模式引入包，如果没有报错，说明添加成功，当然也可以通过 dir() 看看 HTMLTestRunner 包含哪些方法。

```
>>> import HTMLTestRunner
>>> dir(HTMLTestRunner)
['HTMLTestRunner', 'OutputRedirector', 'StringIO', 'Template_mixin', 'TestProgram', 'TestResult',
 '_TestResult', '__author__', '__builtins__', '__doc__', '__file__', '__name__', '__package__',
 '__version__', 'datetime', 'main', 'saxutils', 'stderr_redirector', 'stdout_redirector', 'sys', 'time',
 'unittest']
>>>
```

ok！下面在我们用例中添加可以生成报告的代码：

```
#coding=utf-8

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
import unittest, time, re
import HTMLTestRunner

class Baidu(unittest.TestCase):
```

```

def setUp(self):

    self.driver = webdriver.Firefox()

    self.driver.implicitly_wait(30)

    self.base_url = "http://www.baidu.com"

    self verificationErrors = []

    self.accept_next_alert = True


#测试用例一

def test_baidu_search(self):


#测试用例二

def test_baidu_set(self):


#测试用例三

def test_baidu_xxx(self):


....

def tearDown(self):

    self.driver.quit()

    self.assertEqual([], self.verificationErrors)if __name__ == "__main__":
if __name__ == "__main__":

    testunit=unittest.TestSuite()    #定义一个单元测试容器


    testunit.addTest(Baidu("test_baidu_search"))  #将测试用例加入到测试容器中

    testunit.addTest(Baidu("test_baidu_set"))

    testunit.addTest(Baidu("test_baidu_xxx"))


    filename = 'D:\\result.html'  #定义个报告存放路径，支持相对路径。

```

```
fp = file(filename, 'wb')

runner =HTMLTestRunner.HTMLTestRunner(
    stream=fp,
    title='Report_title',
    description='Report_description')

runner.run(testunit) #自动进行测试
```

代码分析：

用例的部分通过之前的学习已经非常了解了，下面重点分析底部这段代码：

```
testunit=unittest.TestSuite()    #定义一个单元测试容器

testunit.addTest(Baidu("test_baidu_search")) #将测试用例加入到测试容器中
testunit.addTest(Baidu("test_baidu_set"))
testunit.addTest(Baidu("test_baidu_xxx"))

filename = 'D:\result.html' #定义个报告存放路径，支持相对路径。
fp = file(filename, 'wb')

runner =HTMLTestRunner.HTMLTestRunner(
    stream=fp,
    title='Report_title',
    description='Report_description')

runner.run(testunit) #自动进行测试
```

TestSuite 其实并不陌生，在 **23 章** unittest 单元测试框架分析的部分已经介绍，只是为了方便我们使用了 `unittest.main()` 方法，默认会将所有用例执行。因为这里要生成报告，所以要将所有用例列出；

下面的也很容易理解, 创建 result.html 文件, 给以读写权限(wb), 调用 HTMLTestRunner 文件, 并将测试结果以 HTMLTestRunner 规定的格式通过 fp 传递写入到 result.html 文件中。

最后是运行的 testunit , 也就是 TestSuite 中的所有用例。

脚本运行结束, 生成如下报告:

Report_title

Start Time: 2013-10-17 10:19:59

Duration: 0:00:13.062000

Status: Pass 2

Report_description

Show [Summary](#) [Failed](#) [All](#)

Test Group/Test case	Count	Pass	Fail	Error	View
Baidu	2	2	0	0	Detail
test_baidu_search			pass		
test_baidu_set			pass		
Total	2	2	0	0	

问题:

这个报告是根据一个.py 文件生成的, 这样就迫使我们把所有用例都写在一个.py 文件里, 如果我们每一个用例都写在不同的.py 文件里将生成很多个报告, 不便于阅读; 但写在一个.py 文件里, 如果用例非常多的话, 同样不便于维护。

后面, 我们将一起寻求解决办法。

二十七、数据驱动测试

先来理解一下自动化领域的两种驱动, 对象驱动与数据驱动。

数据驱动: 测试数据的改变引起执行结果的改变 叫 数据驱动;

关键字驱动: 测试对象名字的改变引起引起测试结果的改变 叫 关键字驱动。

27.1、读取文件参数化

以百度表搜索为例，我们可以通过脚本循环执行，读取一文件中不同的内容来完成自动化工作，也就是说我们每次取的文件里的搜索关键字不同，而每次百度搜索的结果不同，这也是数据驱动的本质。

代码如下：

d:\abc\data.txt



baidu_read_data.py

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

source = open("D:\\abc\\data.txt", "r")
values = source.readlines()
source.close()

# 执行循环
for serch in values:

    browser = webdriver.Firefox()

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

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

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

    browser.quit()
```

这里简单说明一下，open方法左以只读方式（r）打开本地的data.txt文件，readlines方法是逐行的读取文件内容。

通过for循环，serch可以每次获取到文件中的一行数据，在定位到百度的输入框后，

将数据传入 send_keys(serch)。这样通过循环调用，直到文件的中的所有内容全被读取。

27.2、用户名密码的参数化（读取文件）

按照上面的方法，对自动化脚本中用户名密码进行参数化应该很简单，其实没有想象的那么简单，从目前我所查到 python 读取方法有，整个文件读取，逐行读取，固定字节读取。

怎样才一次读取用户名和密码两个信息呢，最初的修改是这样的：

创建两个文件，分别存放用户名密码



调用用户名密码登录脚本

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

source = open("D:\\abc\\data2.txt", "r") #用户名文件
user = source.read(5) #用户名长度
source.close()

source2 = open("D:\\abc\\data3.txt", "r") #密码文件
pw = source2.read(6) #密码长度
source2.close()

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(user)
time.sleep(3)
driver.find_element_by_id("user_pwd").clear()
driver.find_element_by_id("user_pwd").send_keys(pw)
time.sleep(3)
```

```
driver.find_element_by_name("Submit").click()
time.sleep(1)
driver.quit()
```

缺点：

虽然目的达到了这，但这样的实现有很多问题：

- 1、用户名密码分别在不同的文件里，这样就要求用户名密码必须一一对应
- 2、必须指定读取的长度，测试 `readlines()` 并不是读取的一行数据。
- 3、无法循环读取。

27.3、用户名的参数化（字典）

- 用户名密码参数化
- 解决循环调用

通过一整天研究，重新补习 python 字典、函数调用，如果固定只是读取用户名，密码两个值，可以通过如下方法实现。

创建 `fun.py` 文件，定义一个字典方法：

```
def zidian():
    d={'fnngj':'a23456','testing360':123456}
    print "suess read username and password!!"
    return d
```

字典的可以方便的存放 k,v 键值对，一个键对应一个值；注意，如果密码中有非数字，需要加单引号。

下面循环调用词典的值：

```
#coding=utf-8
from selenium import webdriver
import os,time
import fun #导入函数
```

```
#循环调用字典里的用户名密码，分别赋值给 k,v
for k,v in fun.zidian().items():
    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(k)
    time.sleep(3)
    driver.find_element_by_id("user_pwd").clear()
    driver.find_element_by_id("user_pwd").send_keys(v)
    time.sleep(3)
    driver.find_element_by_id("dl_an_submit").click()
    time.sleep(1)
    driver.close()
```

脚本这样表设计就稳定了很多，每次取的值非常固定，而且同样实现了参数与脚本分离，如果几百个脚本都调用 fun() 函数，当需要修改用户名密码时，只用修改 fun() 函数里面字典的值就可以了。

27.4、用户名密码的参数化（函数）

其实，在[我的项目](#)中只需要做到参数化就行了，并不需要循环的读取内容。那么通过函数调用就可以很简单的解决。

fun.py

```
def user(k='fnngj',v=123456):
    print "suess read username and password!!"
    return k,v
```

赋默认值，并将结果返回。

调用函数值：

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



```

from selenium import webdriver

import os,time

import fun #导入函数


#通过调用函数获得用户名&密码

k,v = fun.user()

print k,v


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(k)

driver.find_element_by_id("user_pwd").clear()

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

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

time.sleep(3)

driver.close()

```

运行结果：

```

>>> ===== RESTART =====
>>>
suess read username and password!!
fnngj 123456
.
-----
Ran 1 test in 25.484s
OK

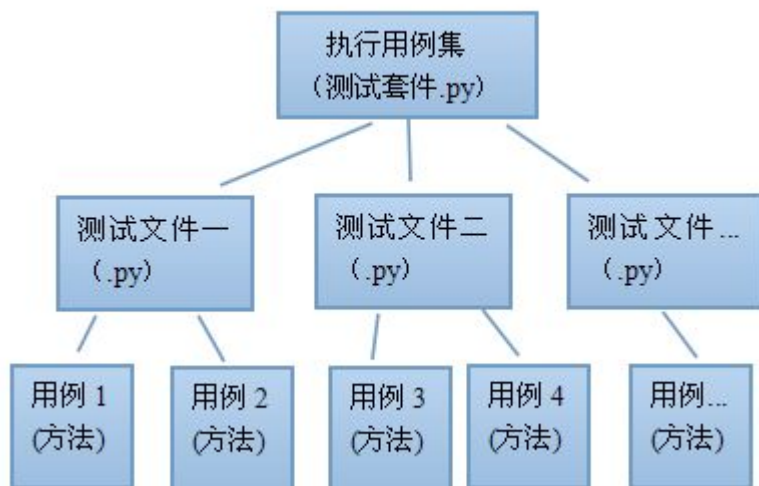
```

如果学好了 python 语言，解决问题的方法是多样的，使用最贴合需求的方法，简单解决问题。这一节写的比较多，对构建自动化框架来说，参数化是非常重要的一个知识点。

二十八、测试套件

在 23 章单元测试框架解析中我提了到“测试套件”，当时只是把一个.py 文件里的多个用例通过测试套件执行。批量执行测试集中 虽然可以批量执行多个.py 文件，但它使用的是读取文件夹下文件的方式，而不是使用的测试套件。这一节就使用测试套件来执行多个.py 测试文件。

最终我们全通过测试套件完成下面的结构：



测试套件的问题解决了，26 章生成测试报告遗留的问题自然也可以解决了。

28.1、测试套件实例

下面通过一个例子来组建我们的测试套件。

test_youdao.py

```
#coding=utf-8
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
```

```

from selenium.webdriver.support.ui import Select

from selenium.common.exceptions import NoSuchElementException

import unittest, time, re

import HTMLTestRunner


class Youdao(unittest.TestCase):

    def setUp(self):

        self.driver = webdriver.Firefox()

        self.driver.implicitly_wait(30)

        self.base_url = "http://www.baidu.com"

        self.verificationErrors = []

        self.accept_next_alert = True


    #百度搜索用例

    def test_youdao_search(self):

        driver = self.driver

        driver.get(self.base_url + "/")

        try:

            driver.find_element_by_id("query").send_keys(u"虫师")

            driver.find_element_by_id("qb").click()

            time.sleep(2)

        except:

            driver.get_screenshot_as_file("D:\\selenium_use_case\\error_png\\kw.png")

            #如果没有找到上面的元素就截取当前页面。


    def tearDown(self):

        self.driver.quit()

```

```

        self.assertEqual([], self verificationErrors)

if __name__ == "__main__":
    suite = unittest.TestSuite()
    suite.addTest(Youdao("test_youdao_search"))
    #这里可以添加更多的用例,如:
    #suite.addTest(Youdao("aaaa"))

    unittest.TextTestRunner().run(suite)

```

test_baidu.py

```

#coding=utf-8

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
import unittest, time, re
import HTMLTestRunner

class Baidu(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.base_url = "http://www.baidu.com"
        self.verificationErrors = []
        self.accept_next_alert = True

```

```

#百度搜索用例

def test_baidu_search(self):

    driver = self.driver

    driver.get(self.base_url + "/")

    try:

        #是一个无法找到的元素 id

        driver.find_element_by_id("kw").send_keys("selenium webdriver")

    except:

        driver.get_screenshot_as_file("D:\\selenium_use_case\\error_png\\kw.png")

        #如果没有找到上面的元素就截取当前页面。

    driver.find_element_by_id("su").click()

    time.sleep(2)

    driver.close()

def tearDown(self):

    self.driver.quit()

    self.assertEqual([], self.verifyErrors)

if __name__ == "__main__":

    suite = unittest.TestSuite()

    suite.addTest(Baidu("test_baidu_search"))

    #同样的，可以在这个文件中添加更多的用例。

    #suite.addTest(Youdao("aaaa"))

    results = unittest.TextTestRunner().run(suite)

```

通过测试套件运行上面两个测试文件，创建 all_tests.py 文件

```
#coding=utf-8

"Combine tests for gnosis.xml.objectify package (req 2.3+)"

import unittest, doctest

import test_baidu, test_youdao  #这里需要导入测试文件

import HTMLTestRunner

suite = doctest.DocTestSuite()

#罗列要执行的文件

suite.addTest(unittest.makeSuite(test_baidu.Baidu))

suite.addTest(unittest.makeSuite(test_youdao.Youdao))

unittest.TextTestRunner(verbosity=2).run(suite)
```

运行结果：

```
>>> ===== RESTART =====
>>>
test_baidu_search (test_baidu.Baidu) ... ok
test_youdao_search (test_youdao.Youdao) ... ok
-----
Ran 2 tests in 15.140s
OK
```

28.2、整合 HTMLTestRunner 测试报告

生成 HTMLTestRunner 报告和前面的方法一样，只是把代码移动 all_tests.py 文件即可；而且只需要在这一个地方生成即可。

下面看加入 HTMLTestRunner 之后的 all_tests.py 文件

```
#coding=utf-8

"Combine tests for gnosis.xml.objectify package (req 2.3+)"

import unittest, doctest

#这里需要导入测试文件（test_baidu.py， test_youdao.py）

import test_baidu, test_youdao

import HTMLTestRunner

suite = doctest.DocTestSuite()

suite.addTest(unittest.makeSuite(test_baidu.Baidu))

suite.addTest(unittest.makeSuite(test_youdao.Youdao))

filename = 'D:\\result20.html'

fp = file(filename, 'wb')

runner = HTMLTestRunner.HTMLTestRunner(

    stream=fp,

    title='Report_title',

    description='Report_description')

runner.run(suite)
```

代码都是前面见过的，这里就不费口舌了再解析了；如果不太理解就多敲几遍，自然就理解。运行测试报告如下，这样再多文件的用例都可以放到一张报告里了。

Report_title

Start Time: 2013-10-30 15:16:49

Duration: 0:00:14.187000

Status: Pass 2

Report_description

Show [Summary](#) [Failed](#) [All](#)

Test Group/Test case
sutie.test_baidu.Baidu
test_baidu_search: baidu test case
sutie.test_youdao.Youdao
test_youdao_search: youdao test case
Total

Count	Pass	Fail	Error	View
1	1	0	0	Detail
			pass	
1	1	0	0	Detail
			pass	
2	2	0	0	

28.3、更易读的报告

报告已经接近完美了，唯一的一点小瑕疵，这报告如果给领导看的话，哪知道什么是什
么，经过 MarkRabbit 的指点，我们可以给每一个用例加个中文注释。

.....
#百度搜索用例

```
def test_baidu_search(self):
    u"""百度搜索用例"""
    driver = self.driver
    driver.get(self.base_url + "/")
```

.....

每个用例（方法）下面都可以加这个一行注释信息，小 u 是避免中文引起的乱码问题。

再来跑一下用例，找开生成的报告，是不是完美了，傻瓜都知道是干嘛的。

Test Group/Test case
sutie.test_baidu.Baidu
test_baidu_search: 百度搜索用例
sutie.test_youdao.Youdao
test_youdao_search: 有道搜索用例
sutie.sogou.test_sogou.Sogou
test_sogou_search: 搜狗搜索用例

二十九、结构改进

到目前为止问题已经解决了，通过测试套件执行所有用例，测试报告整合，所有文件都在一个目录下面，估计用例写多了，不方便管理，这一章试着调整一下结构。

29.1、all_tests.py 移出来

all_tests.py 是调用例的程序，而不是执行用例的，所以应该把它移出来。结构上会更为合理。

/selenium_use_case/test_case/untie/test_baidu.py

/unite/test_baidu.py

/unite/__init__.py

/unite/...

/all_tests.py

目录结构应该是这样的，untie 文件夹下存放具体的执行用例，all_tests.py 应该与 untie 文件夹同级。

另外，需要在 unite 文件夹下放一个 __init__.py 文件，文件内容可以为空。

那么直接移出来后再运行 all_tests.py 文件会提示不到测试文件，所以，我们要对代码做调整，把文件夹加到 sys.path 下就可以找到了。在 all_tests.py 头部添加以下内容。

```
.....
```

```
import sys

sys.path.append("/selenium_use_case/test_case")

from sutie import test_youdao

from sutie import test_baidu

.....
```

29.2、__init__.py 文件解析

`all_tests.py` 是移出来了，但是还有个问题，导入包（用例文件）也是个问题，假如几个用例可以通过“`from sutie import test_xxx`”的方式导入，假如成几百条呢，这样罗列几百条，做法确实太二；那有没有不那么二的方式呢。

还记得上面提到的 `__init__.py` 文件吧，这文件是干嘛的，为什么要在引用的目录下加这个文件？

要弄明白这个问题，首先要知道，python 在执行 `import` 语句时，到底进行了什么操作，按照 python 的文档，它执行了如下操作：

第 1 步，创建一个新的，空的 `module` 对象（它可能包含多个 `module`）；

第 2 步，把这个 `module` 对象插入 `sys.module` 中

第 3 步，装载 `module` 的代码（如果需要，首先必须编译）

第 4 步，执行新的 `module` 中对应的代码。

在执行第 3 步时，首先要找到 `module` 程序所在的位置，搜索的顺序是：

当前路径（以及从当前目录指定的 `sys.path`），然后是 `PYTHONPATH`，然后是 python 的安装设置相关的默认路径。正因为存在这样的顺序，如果当前路径或 `PYTHONPATH` 中存在与标准 `module` 同样的 `module`，则会覆盖标准 `module`。也就是说，如果当前目录下存在 `xml.py`，那么执行 `import xml` 时，导入的是当前目录下的 `module`，而不是系统标准的 `xml`。

了解了这些，我们就可以先构建一个 `package`，以普通 `module` 的方式导入，就可以直接访问此 `package` 中的各个 `module` 了。python 中的 `package` 必须包含一个 `__init__.py` 的文件。

-----以上引用“老王 python”

其实 `__init__.py` 文件中可以有内容；我们在导入一个包时，实际上导入了它的 `__init__.py` 文件。

在 `__init__.py` 文件中添加导入包

```
import test_baidu
import test_youdao
```

然后，`all_tests.py` 文件可是这样修改：

```
.....
import sys
sys.path.append("/selenium_use_case/test_case")
from sutie import *
.....
```

“*” 星号表示导入 `sutie` 目录下的所有文件；在 `sutie` 目录下创建测试用例文件，只用在 `__init__.py` 文件下罗列就可以了。而对于 `all_tests.py` 文件来说不需要做任何调整。

29.3、调用多级目录的用例

当测试用例达到一定量级的时候，为了便于管理，必定需要在目录下面再分目录。假设我们有这样一个结构：

```
/selenium_use_case/test_case/untie/test_baidu.py
                               /unite/test_baidu.py
                               /unite/sogou/test_sogou.py  ----二级测试用例目录
                               /unite/sogou/__init__.py
                               /unite/sogou/...
                               /unite/__init__.py
                               /unite/...
                               /all_tests.py
```

其实，这个问题也很好处理，接着分析 `__init__.py` 文件，它处了能导入当前目录下的

文件，是不是还可以导入其它目录下包，或者模块。假设在/unite/sogou/目录下创建了 test_sogou.py 测试文件，修改 unite 目录__init__.py 文件：

```
#coding=utf-8

import sys

sys.path.append("/selenium_use_case/test_case/sutie")

from sogou import *

import test_baidu

import test_youdao
```

别忘了/unite/sogou/ 目录下也要加__init__.py 文件，并且加入包。掌握的这个技巧，再也不用担心多级目录的问题了。

29.4、改进用例的读取

你以为这样就算完了么？ 还有个更严峻的问题需要处理，如果你够警觉一定注意 all_tests.py 的这段代码：

```
....

suite = doctest.DocTestSuite()

suite.addTest(unittest.makeSuite(test_baidu.Baidu))

suite.addTest(unittest.makeSuite(test_youdao.Youdao))

suite.addTest(unittest.makeSuite(test_sogou.Sogou))

.....
```

对的，这也是无法回避的一个硬伤，跟导入包一样无法避免，想想成百的用例罗列到这里是多么痛的领悟。。

最先想到的是能不能通过一个循环来解决掉这个问题，循环的读取某个目录下的所有文件；如果你还记得本文档的 第24章 有一个叫 test_case_.py 的文件的话，读取某个文件夹下的所有文件是一件很简单的事情；但是如何将结果生成到报告里呢。解决这个问题还是稍微有那么一点儿难度的。

经过改进的新 all_tests.py 代码如下：

```
#coding=utf-8
import sys ,re ,os,math

sys.path.append("/selenium_use_case/test_case")
from sutie import *
import unittest, doctest ,site
import HTMLTestRunner

#将用例组建成数组
alltestnames = [
    'sutie.test_baidu.Baidu',
    'sutie.test_youdao.Youdao',
    'sutie.sogou.test_sogou.Sogou',  #注意这个用例是二级目录下的
]

suite = unittest.TestSuite()

if __name__ == '__main__':

    # 这里我们可以使用 defaultTestLoader.loadTestsFromNames(),
    # 但如果不提供一个良好的错误消息时，它无法加载测试
    # 所以我们加载所有单独的测试，这样将会提高脚本错误的确定。
    for test in alltestnames:
        try:
            #最关键的就是这一句，循环执行数据数的里的用例。
            suite.addTest(unittest.defaultTestLoader.loadTestsFromName(test))
        except Exception:
            print 'ERROR: Skipping tests from "%s".' % test
            try:
                __import__(test)
            except ImportError:
                print 'Could not import the test module.'
            else:
                print 'Could not load the test suite.'
            from traceback import print_exc
            print_exc()

    print
    print 'Running the tests...'

filename = 'D:\\result21.html'
```

```
fp = file(filename, 'wb')

runner =HTMLTestRunner.HTMLTestRunner(
    stream=fp,
    title='Report_title',
    description='Report_description')

runner.run(suite)
```

代码解析，为了做到只解决当前面的问题，上面的代码做了很多简化。其实，我们可以在这里完成更多的功能。

首先我们以“目录.用例文件.用例类”的格式将用例放到一个数组中，可以这样做的前提是我们导入了测试用例文件；然后组成了 `alltestnames` 数组。

通过一个 for 循环来读取数组的内容；读取的方法是：

```
suite.addTest(unittest.defaultTestLoader.loadTestsFromName(test))
```

紧接的 try...except... 是对异常的处理，如果不理解，可以暂无视或删除异常捕捉的相关代码，使代码更清爽。

下面的代码已经见过好多次了，是用于生成 HTMLTestRunner 报告的。

29.5、进一步分离用例列表

都到进一步分，下面知道该怎么做了吧！？ 翻一下 **第27.3节** 参数化中的字典，应该能找到方法。都把用例组成数组了，我们要做的就是把它放到一个单独的文件里。

创建 `allcase_list.py` 文件，与 `all_tests.py` 在同一级目录下。把数组放到一个方法里，`allcase_list.py` 内容如下：

```
def caselist():
    alltestnames = [
        'sutie.test_baidu.Baidu',
        'sutie.test_youdao.Youdao',
        'sutie.sogou.test_sogou.Sogou',
    ]
    print "suess read case list success!!"
```

```
return alltestnames
```

在 all_tests.py 中进行调用：

```
#coding=utf-8

....

import allcase_list #调用数组文件

#获取数组方法
alltestnames = allcase_list.caselist()
...

suite = unittest.TestSuite()

if __name__ == '__main__':

    for test in alltestnames:
        suite.addTest(unittest.defaultTestLoader.loadTestsFromName(test))

.....
```

现在现在优雅多了，把需要的执行的用例往 allcase_list.py 的数组是罗列就行了。
用例的调整，all_tests.py 文件不需要做任何的修改。

最后再回顾一下我们有目前测试的目录结构：

```
/selenium_use_case/test_case/untie/test_baidu.py      -----一级目录测试用例

    /unite/test_baidu.py

    /unite/sogou/test_sogou.py  ----二级目录测试用例目录

    /unite/sogou/__init__.py

    /unite/sogou/...

    /unite/__init__.py

    /unite/...

/all_tests.py      ----调用所有脚本执行

/allcase_list.py    -----罗列要执行的用例

/test_result/result1.html  ----测试报告的存入目录
```

目前看上去还不错的样子~！（得意笑），但是我们项目不同，需求不同，或者当用例达到一定量级后，还会有很多问题暴露出来，需要我们一一的去解决；好吧~！第三版的内容就到这里了。

三十、UliPad--python 开发利器

工欲善其事，必先利其器

有时候往往选择太多，变得无从选择。如果你在 python 开发中已经找到了趁手的 IDE 这一节可以无视。

其实，python 下面能找到一款不错的开发工具是不太容易的。

IDLE 写写单个小程序很好，但一个程序文件与执行信息是两个窗口，程序开多了就分不清哪个了。

pythonWin 也用过，窗口有些老土，窗口布局我不会设置，所以觉得也不好。

notepad++ 这种小巧的万能编辑器，偶尔用用还行。

linux 会有一些非常不错的交互式 python IDE，如 ipython、bpython 等。

vim 肯定是开发神器，但一般也只有高手才会运用自如，体会它的奥妙。

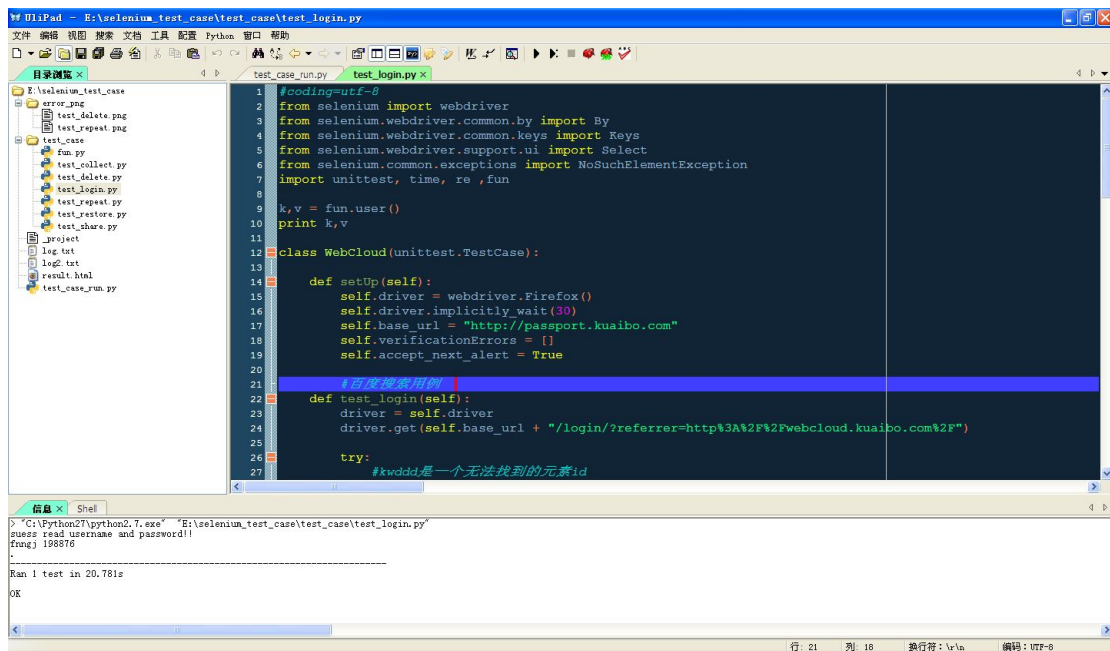
UliPad 是找到的写 python 最舒服的一个 IDE。

地址：<https://code.google.com/p/ulipad/>

免费，可以免费获得并使用它的所有功能。

支持 windows、MAC、linux 等平台。

小巧，内存占用很少，10MB 左右。



```

1 #coding=utf-8
2 from selenium import webdriver
3 from selenium.webdriver.common.by import By
4 from selenium.webdriver.common.keys import Keys
5 from selenium.webdriver.support.ui import Select
6 from selenium.common.exceptions import NoSuchElementException
7 import unittest, time, re, fun
8
9 k,v = fun.user()
10 print k,v
11
12 class WebCloud(unittest.TestCase):
13
14     def setUp(self):
15         self.driver = webdriver.Firefox()
16         self.driver.implicitly_wait(30)
17         self.base_url = "http://passport.kuaibo.com"
18         self.verificationErrors = []
19         self.accept_next_alert = True
20
21     #可被浏览器识别
22     def test_login(self):
23         driver = self.driver
24         driver.get(self.base_url + "/login?referrer=http%3A%2F%2Fwebcloud.kuaibo.com%2F")
25
26         try:
27             #kwddd是一个无法找到的元素id

```

Run 1 test in 20.781s

具体，的安装使用，这里就不介绍了，不是本文档的主题。有兴趣使用可以参考我的博客：

<http://www.cnblogs.com/fnng/p/3393275.html>

另外，还有一些非常棒的收费 python IDE

Wing IDE4.1

<http://wingware.com/>

pycharm

<http://www.jetbrains.com/pycharm/>

希望这一节没影响到文档的和谐。呵呵~！

后记:

都在谈自动化测试，自动化测试是“部分”功能测试的一种替代技术（它们比例肯定在逆转）。通过学习自动脚本也可以使测试人员突破不懂代码的限制；而自动化脚本入门简单。我觉得自动化是方向。

关于自动化又帮了你一段路，但是，依然还有很多问题没有解决；比如，测试用例的多线程处理。目前的结构还不够完美，在脚本运行中，我们可以捕捉更多的信息，更容易的定位问题；使我们的结构更灵活的适应需求的变化；路还很长，任重道远，一起加油吧！

这些问题依然不是一份学习文档可以解决的，如果你掌握了本文档的所有内容，建议从以下几个方面来提高自己的自动化测试水平：

python 语言：兔子（它不让叫兔子了，叫 **MarkRabbit**）的话清晰的说明了学习自动化测试的思路：我们需要用 python 这门语言去调用 selenium 的一些工具来操作浏览器，帮助我们实现“web UI”自动化。所以，我们的重心应该放在语言本身的学习。后面这几章解决问题用的也是 python 技术。

Javascript 语言：在实际的自动化测试过程中，我们会遇到各种问题，有时候 webdriver 提供的方法不能帮我们解决问题，那么需要借助 Javascript 来解决问题。

xpath \css 定位：不能操作一个元素，很多情况下是我们没办法定位这个元素；所以要深入了解 xpath \css 定位的用法。

扩展资料:

rt sm eyiselenium 与 webdriver 的关系:

<http://v.qq.com/boke/page/j/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>

phantomJS

<http://www.cnblogs.com/ziyunfei/archive/2012/09/28/2706061.html>