(一)快速开始

Selenium 1.x 时代已经远去，它理应躺在历史的角落里，靠着壁炉烤着火，抽着旱烟，在袅袅的升起的青烟中回忆那曾经属于自己的美好时代。

不过事实却并非如此，现今原本早应退役的 selenium 1.x 却还是多数人坚定的选择，究其原因不过是 1.x 时代遗留下了大量的文档，代码，教程让人们误以为 1.x 还是这个年代的主流，还应该光鲜亮丽在前台演出属于它的美好。长江后浪推前浪，最为前浪的 selenium 1.x 的宿命应该是死在沙滩上。

好了，直入主题，由于开源社区不再维护 selenium 1.x 再加之更为先进的 selenium 2.0 确实有不少优势之初可以完全取代 1.x，在这里笔者会花一些笔墨，若干篇幅，争取深入浅出的讲解 selenium 2.0 的一些基本知识，常用方法和高级扩展，但由于笔者水平和时间精力等确实有限，文中应该避免不了谬误和臆断之处，还望众位读者多多海涵。

本文中所以代码和示例均由 Ruby 编写，本文介绍的 webdriver api 也主要是 ruby binding。所以首先请确保 ruby 语言在开发环境上正确安装。

教程的第一节从 selenium 2.0 和 webdriver 关系说起。

Selenium 2.0 和 webdriver 之间有什么关系，有什么不可告人的秘密？说来话长，但也简单。Selenium 2.0 其实就是 webdriver。就像张飞就是张翼德，关羽就是关云长一样，叫法不同但内容却是一样的。

安装 selenium webdriver

- 安装 ruby1.8.7 或 1.9.2。注意 selenium-webdriver 只支持 1.8.7 以上的 ruby 版本；
- 使用 gem 安装 selenium-webdriver；打开命令行，输入下列代码完成安装。注意，如果你的开发环境需要 **http proxy** 的话，请注意在 gem 命令中加入--http_proxy 参数;

  gem install selenium-webdriver [--http_proxy]
- 在命令行中输入 gem list，如果 selenium-webdriver 正确安装，则其应该出现在结果列表里。在文本写作时，selenium webdriver 的最新版本应该是 2.2.0;

  gem list selenium-webdriver
- 安装 firefox。本文使用 firefox 作为测试浏览器进行讲解，所以请确保开发环境上正确安装了 firefox。由于 firefox 版本更新较快，我们只需要选择 1 个稳定版本安装既可,本文中笔者使用的版本是 FF 5.0;

简单的 google test

下面我们写几行代码在初次感受一下 webdriver 的魅力，好吧，说老实话原生的 selenium webdriver 并没有什么独特的魅力，相反到是 watir-webdriver 更加的平易近人老少咸宜。这个不是文本讨论的范畴，暂且打住。

view plain

```
1. require 'rubygems'
2. require 'selenium-webdriver'
3.
4. driver = Selenium::WebDriver.for :firefox
5. driver.navigate.to "http://google.com"
6. sleep 3
7.
8. element = driver.find_element(:name, 'q')
9. element.send_keys "Hello WebDriver!"
10. element.submit
11.
12. puts driver.title
13.
14. driver.quit
```

如果一切顺利的话，这几行代码将会打开 firefox 浏览器，然后转跳到 google 首页。等待 3 秒后在搜索框中输入 Hello WebDriver 并提交搜索结果至后台。然后在命令行打印出当前页面的 title，并关闭 ff 浏览器。

如果你的代码不能顺利运行，请从如下几个方面进行检查。

- ruby 的版本是否是 1.8.7 以上并正确安装；
- 代码中是否 **require 'rubygems'**;
- 是否正确安装了 Selenium WebDriver；
- 代码输入是否有误；

Selenium-WebDriver ruby binding 的安装实际上非常的简单和快速，稍微有点 ruby 基础的读者应该能够顺利的自行完成。

下一节将介绍 webdriver 的启动和关闭，欢迎继续关注。

(2)浏览器的简单操作

上一讲我们介绍了如何部署 selenium 2.0 的开发环境，这一讲我们将介绍如何使用 selenium 提供给我们的接口进行浏览器的简单操作。

本文将先介绍适合初级用户的一些常用方法，然后将对一些高级用法和实现源码进行稍微深入一些的分析。

如何打开一个测试浏览器

做自动化测试一般情况下我们都需要首先打开测试浏览器，浏览器开启后我们方可"命令"浏览器去打开新页面，点击特定的链接，判断具体的逻辑等等。因此该操作为"万里长征的第一步"，必须给以重视。具体代码如下。需要注意的是如果使用 chrome 进行测试，那么必须下载安装 **chrome driver**。

view plain

```
require 'rubygems'
require 'selenium-webdriver'
# 打开 firefox
dr = Selenium::WebDriver.for :firefox
dr = Selenium::WebDriver.for :ff
# 打开 ie
dr = Selenium::WebDriver.for :ie
dr = Selenium::WebDriver.for :internet_explorer
# 打开 chrome
dr = Selenium::WebDriver.for :chrome
```

如何打开 1 个具体的 url

打开浏览器后我们需要转到我们的测试 url。下面的代码可以达成这个目的。

view plain

```
require 'rubygems'
require 'selenium-webdriver'
dr = Selenium::WebDriver.for :firefox
# 使用 get 方法
dr.get url
# 使用 navigate 方法，然后再调用 to 方法
dr.navigate.to url
```

如何关闭浏览器

测试结束后往往需要关闭浏览器，下面的代码可以完成这个任务。

view plain

```
1. require 'rubygems'
2. require 'selenium-webdriver'
3. dr = Selenium::WebDriver.for :firefox
4. dr.get url
5. # 使用 quit 方法
6. dr.quit
7. # 使用 close 方法
8. dr.close
```

如何返回当前页面的 url

有时候我们需要返回当前测试页面的 url。比如在使用 soso 进行搜索时，当我们提交了搜索请求后，soso 返回的 url 应该是包含我们所需要搜索的关键字的。

例如如果我们搜索 webdriver，那么提交搜索请求后，页面应当转到 url 为 http://www.soso.com/q?pid=s.idx&cid=s.idx&w=webdriver 的页面，这时候我们取到这个页面的 url，然后通过正则表达式去匹配一下就能够得到我们所搜索的关键字了。具体代码如下。

view plain

```
1. require 'rubygems'
2. require 'selenium-webdriver'
3. dr = Selenium::WebDriver.for :firefox
4. url = 'http://www.soso.com'
5. dr.navigate.to url
6. search_input = dr.find_element :id => 's_input'
7. search_input.send_keys 'webdriver'
8. search_input.submit
9. match = dr.current_url.match(/\b\w+$/)
10. keyword = match[0] if match
```

如何返回当前页面的 title

view plain

```
1. require 'rubygems'
2. require 'selenium-webdriver'
3. dr = Selenium::WebDriver.for :firefox
4. url = 'http://www.soso.com'
5. dr.navigate.to url
6. puts dr.title
```

其他方法

- window_handles：返回当前所有打开浏览器的窗口句柄
- window_handle ： 返回当前的浏览器的窗口句柄
- page_source：返回当前页面的源码
- visible?：当前浏览器是否可见，并不保证支持所有浏览器

深入讨论

操作浏览器的方法主要封装在 lib\selenium\webdriver\common\driver.rb 文件中。

该文件定义了 Selenium::WebDriver::Driver 类。我们启动浏览器就是调用这个类的 for 方法。

(3)如何执行一段 js 脚本

有时候在进行自动化测试时需要在页面上执行一段 js 脚本，这个时候就需要用到 **execute_script** 方法了。

view plain

```
1. require 'rubygems'
2. require 'selenium-webdriver'
3. dr = Selenium::WebDriver.for :ie
4. url = 'http://www.soso.com'
5. dr.navigate.to url
6. sleep 3
7. js = <<JS
8.     q = document.getElementById("tb");
9.     q.style.border = "1px solid red";
10. JS
11.
12. dr.execute_script js
```

上面的代码打开了 SoSo 的首页，并高亮显示了 id 为"tb"的 div。

下面的例子演示了在打开 QQ 首页的时候如何自动 focus 到页面上的 soso 搜索框

view plain

```
require 'rubygems'
require 'selenium-webdriver'
dr = Selenium::WebDriver.for :ie
url = 'http://www.qq.com'
dr.navigate.to url
sleep 3
js = <<JS
    p = document.getElementById("smart_input")
    p.focus()
JS

dr.execute_script js
```

下一讲的内容是元素定位，欢迎继续关注。

(4)如何定位测试元素

测试对象定位一般都是各种 web 自动化测试框架或工具的核心内容。

selenium-Webdriver 的对象定位方法非常的丰富和强大。一般来说强大的对象定位都会提供如下的一些方法。

- 单个对象的定位方法
- 多个对象的定位方法
- 层级定位

selenium-Webdriver 很好的支持了上述的 3 种定位方式。

定位单个对象

在定位单个对象时,selenium-Webdriver 支持使用如下的一些属性对元素进行定位。

view plain

```
:class             => 'class name',
:class_name        => 'class name',
:css               => 'css selector',
:id                => 'id',
:link              => 'link text',
:link_text         => 'link text',
:name              => 'name',
:partial_link_text => 'partial link text',
:tag_name          => 'tag name',
: xpath            => 'xpath',
```

使用 class 或 class_name 进行定位

当所定位的对象具有 class 属性的时候我们可以通过 class 或 class_name 来定位该对象。

下面的例子定位了 soso 首页上 class 为"new"的 span。

view plain

```
1.  require 'rubygems'
2.  require 'selenium-webdriver'
3.  require 'pp'
4.
5.  url = %q{http://www.soso.com/}
6.
7.  dr = Selenium::WebDriver.for :ie
8.  dr.navigate.to url
9.  sleep 1
10.
11. new_icon = dr.find_element(:class => 'ico_new')
12. puts new_icon.tag_name # ---> span
```

使用 id 属性定位

soso 首页的搜索输入框的 html 代码如下：

view plain

```
1.  <input type="text" name="w" smartpid="sb.idx" smartch="sb.c.idx" autocomplet
    e="off" id="s_input" value="">
```

在进行定位前我们先动态定义 highlight 方法，该方法的作用是高亮显示有 id 属性的页面元素。

随后的代码演示了如何使用 id 属性来定位 soso 首页上的搜索输入框。

view plain

```
1.  require 'rubygems'
2.  require 'selenium-webdriver'
3.  require 'pp'
4.
5.  Selenium::WebDriver::Element.module_eval do
6.      def highlight
7.          e_id = self.attribute('id')
8.          puts "#{e_id} = e_id"
9.          js = <<JS
10.         document.getElementById("#{e_id}").style.border = "3px solid red"
11. JS
```

```
12.         @bridge.executeScript(js) if e_id
13.     end
14. end
15.
16. url = %q{http://www.soso.com/}
17.
18. dr = Selenium::WebDriver.for :ie
19. dr.navigate.to url
20. sleep 1
21.
22. s_input = dr.find_element(:id => 's_input')
23. # 或者使用语法糖衣
24. # s_input = dr['s_input']
25. s_input.highlight
```

当使用 id 定位到正确的元素后，highlight 方法会将该元素以红色高亮显示，借此也可以验证代码是否工作正常。

使用 name 属性定位

soso 首页的搜索输入框的 html 代码如下：

view plain

```
1.  <input type="text" name="w" smartpid="sb.idx" smartch="sb.c.idx" autocomplet
    e="off" id="s_input" value="">
2.
3.  # 同样定位 soso 首页的搜索框
4.  s_input = dr.find_element(:name => 'w')
```

使用 css 属性定位

soso 首页的搜索输入框的 html 代码如下：

view plain

```
1.  <input type="text" name="w" smartpid="sb.idx" smartch="sb.c.idx" autocomplet
    e="off" id="s_input" value="">
```

官方文档上说 Selenium-WebDriver 支持 css3 语法，这对使用 jquery 的同学来说无疑是一个好消息。

view plain

```
1.  # css 属性定位 soso 首页搜索框
2.  s_input = dr.find_element(:css => '#s_input')
```

使用 xpath 定位

soso 首页的搜索输入框的 html 代码如下：

view plain

```
<input type="text" name="w" smartpid="sb.idx" smartch="sb.c.idx" autocomplete="off" id="s_input" value="">
```

```
# 使用 xpath 定位 soso 首页搜索框
s_input = dr.find_element(:xpath => %Q{//input[@id='s_input' and @name='w']})
```

使用其他方式定位

在定位 link 对象的时候，可以使用 link 和 link_text 属性；

另外还可以使用 tag_name 属性定位任意元素；

定位多个元素

find_elements 方法可以定位一组对象，例如

view plain

```
# 定位页面上所有的 link 对象
all_links = driver.find_elements(:tag_name, 'a')
all_links.each do {|l| puts l.class} # ---> Selenium::WebDriver::Element
```

上面的代码返回页面上所有 link 对象的数组

下面代码演示了如何选取页面上所有的 button 对象：

view plain

```
all_buttons = driver.find_elements(:tag_name, 'input').select do |i|
    i['type'] == 'button'
end
```

层级定位

层级定位的思想是先定位父对象，然后再从父对象中精确定位出其我们需要选取的后代元素。层级定位一般的应用场景是无法直接定位到需要选取的元素，但是其父元素比较容易定位，通过定位父元素再遍历其子元素选择需要的目标元素，或者需要定位某个元素下所有的子元素。

下面的代码演示了如何使用层级定位选取 id 为 bm 的 div 下所有的 link 元素，并打印出所有其 href

view plain

```
links = dr.find_element(:id, 'bm').find_elements(:css => 'a')
links.each do |l|
    puts l['href']
end
```

这一节分享了 Selenium-WebDriver 定位元素的一般方法，下一节将介绍 Selenium-WebDriver 对 **frame** 的处理

(5)如何定位 frame 中的元素

在 web ui 自动化测试中，frame 一直是令人头痛的问题，就像上班必须挤公车坐地铁一般，frame 的问题总是令人气闷纠结为之黯然神伤。

以前在使用 watir 1.6x 的时候，frame 也是颇为棘手的一个问题。不但要照本宣科的进行一系列的设置，而且在进行实际代码编写的过程中会遇到各种奇奇怪怪的问题。frame 就像中国男足的后防线，问题多多难以解决。

selenium webdriver 处理 frame 比较简单，这点比某些测试工具要先进一些，令人身心愉悦。

以下面的 html 代码为例，我们看一下如何定位 frame 上的元素。

view plain

```
frame.html

<html>

    <head>

        <title>Frame</title>

        <style>

            #f_1 {width: 10em; height: 10em; border: 1px solid #ccc; }

            #f_2 {display: none}

        </style>

    </head>

```

```
19.     <body>
20.
21.         <p id = "p">Outside frame</p>
22.
23.         <iframe id = "f_1" f1" src = "part1.htm"></iframe>
24.
25.         <iframe id = "f_2" src = "part2.htm"></iframe>
26.
27.     </body>
28.
29. </html>
30.
31.
32.
33. part1.htm
34.
35. <html>
36.
37.     <head><title>Part1</title></head>
38.
39.     <body>
40.
41.         <p id = "f_p">This is part 1</p>
42.
43.         <input id = "btn" type = "button" value = "click me" onclick = "aler
    t('hello')" />
44.
45.     </body>
46.
47. </html>
```

switch_to 方法会 new1 个 TargetLocator 对象，使用该对象的 frame 方法可以将当前识别的"主体"移动到需要定位的 frame 上去。

view plain

```
1.  require 'rubygems'
2.
3.  require 'selenium-webdriver'
4.
5.
6.
7.  dr = Selenium::WebDriver.for :firefox
8.
```

```
9.  frame_file = 'file:///'+File.expand_path(File.join(File.dirname(__FILE__), '
    frame.html'))
10.
11. dr.navigate.to frame_file
12.
13. # 定位 default content 上的 p 元素
14.
15. p dr.find_element(:id => 'p')
16.
17. # 将当前识别主体移动到 id 为 f_1 的 frame 上去
18.
19. dr.switch_to.frame('f_1')
20.
21. # 点击 frame 上的 button
22.
23. dr.find_element(:id =>'btn').click # --> a alert will popup
24.
25. # 此时再去定位 frame 外的 p 元素将出现错误
26.
27. p dr.find_element(:id => 'p') # --> error
28.
29. # 将识别的主体切换出 frame
30.
31. dr.switch_to.default_content
32.
33. p dr.find_element(:id => 'p') # --> ok
```

webdriver 的 frame 处理方式让人感觉那个不痛越来越轻松，这点进步值得肯定。