启动浏览器

场景

在使用webdriver进行测试时启动浏览器无疑是必须的前置工作。

代码

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

关闭浏览器

场景

在脚本运行完毕或者测试代码结束的时候关闭浏览器是非常自然的事情，就像在吃完饭后就把餐桌收拾干净一样。

关闭浏览器有两种方式：

- close方法
- quit方法

close方法关闭当前的浏览器窗口，quit方法不仅关闭窗口，还会彻底的退出webdriver，释放与driver server之间的连接。所以简单来说quit是更加彻底的close，quit会更好的释放资源，适合强迫症和完美主义者。

代码

```
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
sleep 2
puts 'browser will be closed'
dr.quit() # or dr.close()
puts 'browser is closed'
```

浏览器最大化

场景

当我们在测试中使用一些基于图像和坐标的辅助测试工具时，我们就会需要使浏览器在每次测试时保存最大化，以便在同一分辨率下进行图像比对和坐标点选。

举例来说，如果在webdriver测试中使用了sikuli来对flash插件进行操作的话，把浏览器最大化无疑是一个比较简单的保证分辨率统一的解决方案。

代码

```
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
sleep 2
puts 'maximize browser'
dr.manage.window.maximize()
sleep 2
puts 'close browser'
dr.quit
```

设置浏览器大小

场景

设置浏览器窗口的大小有下面两个比较常见的用途：

- 在统一的浏览器大小下运行用例，可以比较容易的跟一些基于图像比对的工具进行结合，提升

测试的灵活性及普遍适用性。比如可以跟sikuli结合，使用sikuli操作flash；

- 在不同的浏览器大小下访问测试站点，对测试页面截图并保存，然后观察或使用图像比对工具对被测页面的前端样式进行评测。比如可以将浏览器设置成移动端大小(320x480)，然后访问移动站点，对其样式进行评估；

代码

将浏览器调整成移动端大小，然后访问移动站点，对移动站点的样式进行评估。 ``` require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome

dr.manage.window.resize_to(320,480) dr.get 'http://www.3g.qq.com'

sleep 5 dr.quit ```

讨论

webdriver提供了很多调整浏览器窗口的接口，比如move_to(移动窗口)，position(设置或获取浏览器的位置)。在一般情况下这些功能并不常用。

访问链接

情景

web UI测试里最简单也是最基本的事情就是访问1个链接了。

webdriver的api里有2种访问url的方式，分别是get和navigate.to方法。一般情况下建议使用get，因为其字母比较少，不太容易出错。

代码

```
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
url = 'http://www.baidu.com'
puts "now access #{url}"
dr.get url
sleep 5

dr.quit
```

讨论

navigate方法实际上会产生1个Navigator对象，其封装了与导航相关的一些方法，比如前进后退等。

打印当前页面的title及url

情景

测试中，访问1个页面然后判断其title是否符合预期是很常见的1个用例，所谓用例不够，title来凑就是这个道理。更具体一点，假设1个页面的title应该是'hello world', 那么可以写这样的一个用例：访问该页面，获取该页面的title，判断获取的值是否等于'hello world'。

获取当前页面的url也是非常重要的一个操作。在某些情况下，你访问一个url，这时系统会自动对这个url进行跳转，这就是所谓的'重定向'。一般测试重定向的方法是访问这个url，然后等待页面重定向完毕之后，获取当前页面的url，判断该url是否符合预期。另外的一个常见的测试场景是提交了一个表单，如果表单内容通过了验证，那么则会跳转到一个新页面，如果未通过验证，则会停留在当前页面，此时获取当前页面的url则可以帮助我们判断表单提交的跳转是否符合预期。更具体一点，假如你在测试一个登陆页面，输入正确的登陆信息后，会跳转到系统首页。获取跳转后的url然后判断其是否与系统首页的url相符将是一个很不错的用例。

代码

```
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
url = 'http://www.baidu.com'
puts "now access #{url}"
dr.get url
```

```
puts "title of current page is #{dr.title}"
puts "url of current page is #{dr.current_url}"
sleep 1

dr.quit
```

前进和后退

场景

说实话，这两个功能一般不太常用。所能想到的场景大概也就是在几个页面间来回跳转，省去每次都get url。

代码

```
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
first_url = 'http://www.baidu.com'
puts "now access #{first_url}"
dr.get(first_url)
sleep 1
second_url = 'http://www.news.baidu.com'
puts "now access #{second_url}"
dr.get(second_url)
sleep 1

puts "back to #{first_url}"
dr.navigate.back()
sleep 1
puts "forward to #{second_url}"
dr.navigate.forward()
sleep 1
dr.quit()
```

简单的对象定位

场景

测试对象的定位和操作是webdriver的核心内容，其中操作又是建立在定位的基础之上,因此对象定位就越发显得重要了。

定位对象的目的一般有下面几种

- 操作对象
- 获得对象的属性，如获得测试对象的class属性，name属性等等
- 获得对象的text
- 获得对象的数量

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

代码

html代码 form.html

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8" />
    <title>Form</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>simple login form</h3>
    <form class="form-horizontal">
      <div class="control-group">
        <label class="control-label" for="inputEmail">Email</label>
        <div class="controls">
          <input type="text" id="inputEmail" placeholder="Email" name="email">
        </div>
      </div>
      <div class="control-group">
        <label class="control-label" for="inputPassword">Password</label>
        <div class="controls">
          <input type="password" id="inputPassword" placeholder="Password" name="password">
        </div>
      </div>
      <div class="control-group">
        <div class="controls">
          <label class="checkbox">
            <input type="checkbox"> Remember me
          </label>
          <button type="submit" class="btn">Sign in</button>
          <a href="#">register</a>
        </div>
      </div>
    </form>
  </body>
</html>
```

ruby代码 simple_locate.rb

```
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'form.html'))
puts file_path
dr.get file_path

# by id
dr.find_element(:id, 'inputEmail').click

# by name
dr.find_element(:name, 'password').click

# by tagname
puts dr.find_element(:tag_name, 'form')[:class]

# by class_name
e = dr.find_element(:class, 'controls')
dr.execute_script('$(arguments[0]).fadeOut().fadeIn()', e)
sleep 1

# by link text
link = dr.find_element(:link_text, 'register')
dr.execute_script('$(arguments[0]).fadeOut().fadeIn()', link)
sleep 1

# by partial link text
link = dr.find_element(:partial_link_text, 'reg')
dr.execute_script('$(arguments[0]).fadeOut().fadeIn()', link)
sleep 1

# by css selector
div = dr.find_element(:css, '.controls')
dr.execute_script('$(arguments[0]).fadeOut().fadeIn()', div)
```

```
sleep 1

# by xpath
dr.find_element(:xpath, '/html/body/form/div[3]/div/label/input').click

sleep 2
dr.quit
```

讨论

上面例子里由于html文件中引用了jquery，所以在执行js时可以使用jquery的$()及fadeIn()等方法。如果你测试的页面没用包含jquery的话，这些方法是无效的。

定位一组对象

场景

从上一节的例子中可以看出，webdriver可以很方便的使用find_element方法来定位某个特定的对象，不过有时候我们却需要定位一组对象，这时候就需要使用find_elements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

代码

checkbox.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>Checkbox</title>
        <script type="text/javascript" async="" src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
        <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
    </head>
    <body>
        <h3>checkbox</h3>
        <div class="well">
            <form class="form-horizontal">
                <div class="control-group">
                    <label class="control-label" for="c1">checkbox1</label>
                    <div class="controls">
                        <input type="checkbox" id="c1" />
                    </div>
                </div>
                <div class="control-group">
                    <label class="control-label" for="c2">checkbox2</label>
                    <div class="controls">
                        <input type="checkbox" id="c2" />
                    </div>
                </div>
                <div class="control-group">
                    <label class="control-label" for="c3">checkbox3</label>
                    <div class="controls">
                        <input type="checkbox" id="c3" />
                    </div>
                </div>
                <div class="control-group">
                    <label class="control-label" for="r">radio</label>
                    <div class="controls">
                        <input type="radio" id="r" />
                    </div>
                </div>
            </form>
        </div>
```

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

find_element.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'checkbox.html'))

dr.get file_path

# 选择所有的checkbox并全部勾上
dr.find_elements(:css, 'input[type=checkbox]').each {|c| c.click}
dr.navigate.refresh()
sleep 1

# 打印当前页面上有多少个checkbox
puts dr.find_elements(:css, 'input[type=checkbox]').size

# 选择页面上所有的input，然后从中过滤出所有的checkbox并勾选之
dr.find_elements(:tag_name, 'input').each do |input|
    input.click if input.attribute(:type) == 'checkbox'
end
sleep 1

# 把页面上最后1个checkbox的勾给去掉
dr.find_elements(:css, 'input[type=checkbox]').last.click

sleep 2
dr.quit
```

讨论

checkbox.html必须与find_elments.rb在同一级目录下

层级定位

场景

在实际的项目测试中，经常会有这样的需求：页面上有很多个属性基本相同的元素，现在需要具体定位到其中的一个。由于属性基本相当，所以在定位的时候会有些麻烦，这时候就需要用到层级定位。先定位父元素，然后再通过父元素定位子孙元素。

代码

下面的代码演示了如何通过层级定位来定位下拉菜单中的某一项。由于两个下拉菜单中每个选项的link text都相同，href也一样，所以在这里就需要使用层级定位了。

具体思路是：先点击显示出1个下拉菜单，然后再定位到该下拉菜单所在的ul，再定位这个ul下的某个具体的link。在这里，我们定位第1个下拉菜单中的Another action这个选项。

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="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" />
    </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>
```

level_locate.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'level_locate.html'))

dr.get file_path

dr.find_element(:link_text, 'Link1').click
wait = Selenium::WebDriver::Wait.new({:timeout => 30})
wait.until { dr.find_element(:id, 'dropdown1').displayed? }
menu = dr.find_element(:id, 'dropdown1').find_element(:link_text, 'Another action')

dr.action.move_to(menu).perform()

sleep 2
dr.quit
```

讨论

move_to方法实际上是模拟把鼠标移动到某个具体的测试对象上。

操作测试对象

场景

定位到具体的对象后，我们就可以对这个对象进行具体的操作，比如先前已经看到过的点击操作(click)。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

代码

下面的代码演示了如何点击元素，如何往文本框中输入文字以及如何清空文字。

operate_element.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>Level Locate</title>
        <script type="text/javascript" async="" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
    </head>
    <body>
        <h3>Level locate</h3>
        <div class="span3">
            <div class="well">
                <div class="dropdown">
                    <a class="dropdown-toggle" data-toggle="dropdown" href="#">Link1</a>
                    <ul class="dropdown-menu" role="menu" aria-labelledby="dLabel" id="dropdown1" >
                        <li><a tabindex="-1" href="#">Action</a></li>
                        <li><a tabindex="-1" href="#">Another action</a></li>
                        <li><a tabindex="-1" href="#">Something else here</a></li>
                        <li class="divider"></li>
                        <li><a tabindex="-1" href="#">Separated link</a></li>
                    </ul>
                </div>
            </div>
        </div>
        <div class="span3">
            <div class="well">
                <div class="dropdown">
                    <a class="dropdown-toggle" data-toggle="dropdown" href="#">Link2</a>
                    <ul class="dropdown-menu" role="menu" aria-labelledby="dLabel" >
                        <li><a tabindex="-1" href="#">Action</a></li>
                        <li><a tabindex="-1" href="#">Another action</a></li>
                        <li><a tabindex="-1" href="#">Something else here</a></li>
                        <li class="divider"></li>
                        <li><a tabindex="-1" href="#">Separated link</a></li>
                    </ul>
                </div>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

operate_element.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'operate_element.html'))

dr.get file_path
#click
dr.find_element(:link_text, 'Link1').click
sleep(1)
dr.find_element(:link_text, 'Link1').click

#send_keys
element = dr.find_element(:name, 'q')
element.send_keys('something')
sleep(1)

#clear
element.clear()

sleep(2)
dr.quit
```

send keys模拟按键输入

场景

send_keys方法可以模拟一些组合键操作，比如ctrl+a等。另外有时候我们需要在测试时使用tab键将焦点转移到下一个元素，这时候也需要send_keys。在某些更复杂的情况下，还会出现使用send_keys来模拟上下键来操作下拉列表的情况。

代码

下面的代码演示了如何将A多行文本框中的内容清空并复制到B文本框中。

send_keys.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>send keys</title>
        <script type="text/javascript" async="" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
    </head>
    <body>
        <h3>send keys</h3>
        <div class="row-fluid">
        <div class="span3">
            <div class="well">
                <label>A</label>
                <textarea rows="10", cols="10" id="A">I think watir-webdriver is better than selenium-webdriver</textarea>
            </div>
        </div>
        <div class="span3">
            <div class="well">
                <label>B</label>
                <textarea rows="10", cols="10" id="B"></textarea>
            </div>
        </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

send_keys.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'send_keys.html'))
dr.get file_path

# copy content of A
dr.find_element(:id, 'A').send_keys([:control, 'a'])
dr.find_element(:id, 'A').send_keys([:control, 'x'])
sleep(1)

# paste to B
dr.find_element(:id, 'B').send_keys([:control, 'v'])
sleep(1)

# send keys to A
dr.find_element(:id, 'A').send_keys('watir', '-', 'webdriver', :space, 'is', :space, 'better')
sleep(2)

dr.quit()
```

处理button group

场景

button group就是按钮组，将一组按钮排列在一起。处理这种对象的思路一般是先找到button group的包裹(wrapper)div，然后通过层级定位，用index或属性去定位更具体的按钮。

代码

下面的代码演示了如何找到second这个按钮。其处理方法是先找到button group的父div，class为btn-group的div，然后再找到下面所有的div(也就是button)，返回text是second的div。

button_group.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>button group</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(){
                $('.btn').click(function(){
                    alert($(this).text());
                });
            });
        </script>
    </head>
    <body>
        <h3>button group</h3>
        <div class="row-fluid">
            <div class="span3">
                <div class="well">
                    <div class="btn-toolbar">
                        <div class="btn-group">
                            <div class="btn">first</div>
                            <div class="btn">second</div>
                            <div class="btn">third</div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

button_group.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'button_group.html'))
dr.get file_path

# 定位text是second的按钮
second_btn = dr.find_element(:class, 'btn-group').find_elements(:class, 'btn').detect {|btn| btn.text == 'second'}
second_btn.click()
sleep(2)

dr.quit()
```

讨论

自己查资料搞清楚detect方法的作用。

处理button dropdown

场景

button dropdown就是把按钮和下拉菜单弄到了一起。处理这种对象的思路一般是先点击这个按

钮，等待下拉菜单显示出来，然后使用层级定位方法来获取下拉菜单中的具体项。

代码

下面的代码演示了如何找到watir-webdriver这个菜单项。其处理方法是先点击info按钮，然后等到下拉菜单出现后定位下拉菜单的ul元素，再定位ul元素中link text为watir-webdriver的link，并点击之。

button_dropdown.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>button group</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(){
                $('.btn').click(function(){
                    alert($(this).text());
                });
            });
        </script>
    </head>
    <body>
        <h3>button group</h3>
        <div class="row-fluid">
            <div class="span3">
                <div class="well">
                    <div class="btn-toolbar">
                        <div class="btn-group">
                            <div class="btn">first</div>
                            <div class="btn">second</div>
                            <div class="btn">third</div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

button_dropdown.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'button_dropdown.html'))
dr.get file_path

# 定位text是watir-webdriver的下拉菜单
# 首先显示下拉菜单
dr.find_element(:link_text, 'Info').click()
wait = Selenium::WebDriver::Wait.new(timeout: 10)
wait.until { dr.find_element(:class, 'dropdown-menu').displayed? }

# 通过ul再层级定位
dr.find_element(:class, 'dropdown-menu').find_element(:link_text, 'watir-webdriver').click()
sleep(1)

dr.quit()
```

处理navs

场景

navs可以看作是简单的类似于tab的导航栏。一般来说导航栏都是ul+li。先定位ul再去层级定位li中的link基本就能解决问题。

代码

navs.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>Navs</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(){
                    $('.nav').find('li').click(function() {
                        $(this).parent().find('li').removeClass('active');
                        $(this).addClass('active');
                    });
                }
            );
        </script>
    </head>
    <body>
        <h3>Navs</h3>
        <div class="row-fluid">
            <div class="span3">
                <ul class="nav nav-pills">
                    <li class="active">
                        <a href="#">Home</a>
                    </li>
                    <li><a href="#">Content</a></li>
                    <li><a href="#">About</a></li>
                </ul>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

navs.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'navs.html'))
dr.get file_path

# 方法1：层级定位，先定位ul再定位li
dr.find_element(:class, 'nav').find_element(:link_text, 'About').click()
sleep(1)

# 方法2：直接定位link
dr.find_element(:link_text, 'Home').click()

dr.quit()
```

处理面包屑

场景

在实际的测试脚本中，有可能需要处理面包屑。处理面包屑主要是获取其层级关系，以及获得当前的层级。一般来说当前层级都不会是链接，而父层级则基本是以链接，所以处理面包屑的思路就很明显了。找到面包屑所在的div或ul，然后再通过该div或ul找到下面的所有链接，这些链接就是父层

级。最后不是链接的部分就应该是当前层级了。

代码

breadcrumb.html

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

                }
            );
        </script>
    </head>
    <body>
        <h3>breadcrumb</h3>
        <div class="row-fluid">
            <div class="span3">
                <ul class="breadcrumb">
                    <li><a href="#">Home</a> <span class="divider">/</span></li>
                    <li><a href="#">Library</a> <span class="divider">/</span></li>
                    <li class="active">Data</li>
                </ul>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

breadcrumb.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'breadcrumb.html'))
dr.get file_path

# 获得其父层级
anstors = dr.find_element(:class, 'breadcrumb').find_elements(:tag_name, 'a').map { |link| link.text }
p anstors
sleep(1)

# 获取当前层级
# 由于页面上可能有很多class为active的元素
# 所以使用层级定位最为保险
puts dr.find_element(:class, 'breadcrumb').find_element(:class, 'active').text

dr.quit()
```

处理分页

场景

对分页来说，我们最感兴趣的是下面几个信息

- 总共有多少页
- 当前是第几页
- 是否可以上一页和下一页

代码

下面的代码演示了如何获取分页的总数以及当前是第几页

pagination.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>Pagination</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(){
                $('.pagination').find('li').click(function(){
                    $(this).parent().find('li').removeClass('active');
                    $(this).addClass('active');
                });
            });
        </script>
    </head>

    <body>
        <h3>Pagination</h3>
        <div class="row-fluid">
            <div class="span6">
                <div class="pagination pagination-large">
                    <ul>
                        <li><a href="#">Prev</a></li>
                        <li class="active"><a href="#">1</a></li>
                        <li><a href="#">2</a></li>
                        <li><a href="#">3</a></li>
                        <li><a href="#">4</a></li>
                        <li><a href="#">5</a></li>
                        <li><a href="#">Next</a></li>
                    </ul>
                </div>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

pagination.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'pagination.html'))
dr.get file_path

# 获得所有分页的数量
# -2是因为要去掉上一个和下一个
total_pages = dr.find_element(:class, 'pagination').find_elements(:tag_name, 'li').size - 2
puts "total page is #{total_pages}"

# 获取当前页面的url以及当前页面是第几页
current_page = dr.find_element(:class, 'pagination').find_element(:class, 'active')
puts "current page is #{current_page.text}"
dr.quit()
```

处理对话框

场景

页面上弹出的对话框是自动化测试经常会遇到的一个问题。很多情况下这个弹出的对话框是一个iframe，处理起来有点麻烦，需要进行switch_to操作。但现在很多前端框架的对话框都是div形式的，这就让我们的处理变得十分简单了。

处理对话框一般会做下面的一些事情

- 打开对话框
- 关闭对话框
- 操作对话框中的元素

代码

下面的代码演示了如何打开、关闭以及点击对话框中的链接

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('try watir-webdriver right now!');
                });
            });
        </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>watir-webdriver is better than slenium-webdriver</p>
                    <a href="#" id="click">click me</a>
                  </div>
                  <div class="modal-footer">
                    <button class="btn" data-dismiss="modal" aria-hidden="true">Close</button>
                    <button class="btn btn-primary">Save changes</button>
                  </div>
                </div>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

modal.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'modal.html'))
dr.get file_path
```

```
# 打开对话框
dr.find_element(:id, 'show_modal').click

wait = Selenium::WebDriver::Wait.new(timeout: 10)
wait.until { dr.find_element(id: 'myModal').displayed? }

# 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
# 在watir-webdriver中只需要fire_event(:click)就可以了
link = dr.find_element(id: 'myModal').find_element(id: 'click')
dr.execute_script('$(arguments[0]).click()', link)
sleep(2)

# 关闭对话框
dr.find_element(:class, 'modal-footer').find_elements(:tag_name, 'button').first.click

dr.quit()
```

获取测试对象的属性及内容

场景

获取测试对象的内容是前端自动化测试里一定会使用到的技术。比如我们要判断页面上是否显示了一个提示，那么我们就需要找到这个提示对象，然后获取其中的文字，再跟我们的预期进行比较。在webdriver中使用element.attribute()方法可以获取dom元素(测试对象)的属性。

获取测试对象的属性能够帮我们更好的进行对象的定位。比如页面上有很多class都是'btn'的div，而我们需要定位其中1个有具有title属性的div。由于selenium-webdriver是不支持直接使用title来定位对象的，所以我们只能先把所有class是btn的div都找到，然后遍历这些div，获取这些div的title属性，一旦发现具体title属性的div，那么返回这个div既可。在webdriver中，使用element.text()方法可以返回dom节点的内容(text)。

代码

下面的代码演示了如何获取测试对象的title属性和该对象的文字内容

attribute.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>attribute</title>
        <script type="text/javascript" async="" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
        <script type="text/javascript">
            $(document).ready(function(){
                $('#tooltip').tooltip({"placement": "right"});
            });
        </script>
    </head>

    <body>
        <h3>attribute</h3>
        <div class="row-fluid">
            <div class="span6">
                <a id="tooltip" href="#" data-toggle="tooltip" title="watir-webdriver better than selenium-webdriver">hover to see tooltip</a>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

attribute.rb

```
#encoding: utf-8
require 'selenium-webdriver'
```

```
dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'attribute.html'))
dr.get file_path

link = dr.find_element(id: 'tooltip')

# 获得tooltip的内容
puts link.attribute('data-original-title')

# 获取该链接的text
puts link.text()

dr.quit()
```

获取测试对象的css属性

场景

当你的测试用例纠结细枝末节的时候，你就需要通过判断元素的css属性来验证你的操作是否达到了预期的效果。比如你可以通过判断页面上的标题字号以字体来验证页面的显示是否符合预期。当然，这个是强烈不推荐的。因为页面上最不稳定的就是css了，css变动频繁，而且通过属性也不能直观的判断页面的显示效果，还不如让人为的去看一眼，大问题一望即知。

代码

下面的代码演示了如何获取测试对象的css属性。

css.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>attribute</title>
        <script type="text/javascript" async="" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
        <script type="text/javascript">
            $(document).ready(function(){
                $('#tooltip').tooltip({"placement": "right"});
            });
        </script>
    </head>

    <body>
        <h3>attribute</h3>
        <div class="row-fluid">
            <div class="span6">
                <a id="tooltip" href="#" data-toggle="tooltip" title="watir-webdriver better than selenium-webdriver">hover to see tooltip</a>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

css.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'css.html'))
dr.get file_path

link = dr.find_element(id: 'tooltip')
puts link.css_value(:color)

puts dr.find_element(:tag_name, 'h3').css_value('font')

dr.quit()
```

获取测试对象的状态

场景

在web自动化测试中，我们需要获取测试对象的四种状态

- 是否显示。使用element.displayed?()方法；
- 是否存在。使用find_element方法，捕获其抛出的异常，如果是NoSuchElement异常的话则可以确定该元素不存在；
- 是否被选中。一般是判断表单元素，比如radio或checkbox是否被选中。使用element.selected?()方法；
- 是否enable，也就是是否是灰化状态。使用element.enabled?()方法；

代码

status.html

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>attribute</title>
        <script type="text/javascript" async="" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
        <link href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/css/bootstrap-combined.min.css" rel="stylesheet" />
        <script type="text/javascript">
            $(document).ready(function(){
                $('#tooltip').tooltip({"placement": "right"});
            });
        </script>
    </head>

    <body>
        <h3>attribute</h3>
        <div class="row-fluid">
            <div class="span6">
                <a id="tooltip" href="#" data-toggle="tooltip" title="watir-webdriver better than selenium-webdriver">hover to see tooltip</a>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

status.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'status.html'))
dr.get file_path

text_field = dr.find_element(:name, 'user')
puts text_field.enabled?

# 直接用enabled?方法去判断该button的话返回的会是true
# 这是因为button是使用css方法去disabled的，并不是真正的disable
# 这时候需要判断其class里是否有disabled这值来判断其是否处于disable状态
puts dr.find_element(:class, 'btn').enabled?

# 隐藏掉text_field
# 判断其是否显示
dr.execute_script('$(arguments[0]).hide()', text_field)
puts text_field.displayed?

# 使用click方法选择raido
radio = dr.find_element(name: 'radio')
radio.click()
```

```
puts radio.selected?

# 判断元素是否存在
begin
    dr.find_element(id: 'none')
rescue Selenium::WebDriver::Error::NoSuchElementError
    puts 'element does not exist'
end

dr.quit()
```

讨论

在这里我们遇到了一种情况，那就是测试对象看上去是disabled，但是使用enabled方法却返回true。这时候一般思路是判断该对象的css属性或class，通过这些值去进一步判断对象是否disable。

获取测试对象的状态

场景

表单对象的操作比较简单，只需要记住下面几点

- 使用send_keys方法往多行文本框和单行文本框赋值；
- 使用click方法选择checkbox
- 使用click方法选择radio
- 使用click方法点击button
- 使用click方法选择option，从而达到选中select下拉框中某个具体菜单项的效果

代码

form.html

form.rb

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8" />
        <title>form</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(){
                    $('input[type=submit]').click(function(){
                        alert('watir-webdriver is better than selenium webdriver');
                    });
            });
        </script>
    </head>

    <body>
        <h3>form</h3>
        <div class="row-fluid">
            <div class="span6 well">
                <form>
                    <fieldset>
                        <legend>Legend</legend>
                        <label class="checkbox">
                            <input type="checkbox"> Check me out
                        </label>

                        <label class="radio">
                            <input type="radio"> select me
                        </label>

                        <label class="select">
                            <select>
```

```
                                <option>0</option>
                                <option>1</option>
                                <option>2</option>
                            </select> select one item
                        </label>

                        <input type="submit" class="btn" value="submit" />
                    </fieldset>
                </form>
            </div>
        </div>
    </body>
    <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

```
    #encoding: utf-8
    require 'selenium-webdriver'

    dr = Selenium::WebDriver.for :chrome
    file_path = 'file:///' + File.expand_path(File.join('.', 'form.html'))
    dr.get file_path

    # 选中checkbox
    dr.find_element(:css, 'input[type=checkbox]').click()
    sleep(1)

    # 选中radio
    dr.find_element(:css, 'input[type=radio]').click()
    sleep(1)

    # 选择下拉菜单中的最后一项
    dr.find_element(:tag_name, 'select').find_elements(:tag_name, 'option').last.click()
    sleep(1)

    # 点击提交按钮
    dr.find_element(:css, 'input[type=submit]').click()
    sleep(1)

    alert = dr.switch_to.alert
    puts alert.text
    alert.accept()

    dr.quit()
```

执行js

场景

如果你熟悉js的话，那么使用webdriver执行js就是一件很高效的事情了。在webdriver脚本中直接执行js的好处很多，这里就不一一枚举了。

webdriver提供了execute_script()接口来帮助我们完成这一工作。在实际的测试脚本中，以下两种场景是经常遇到的

- 在页面直接执行一段js
- 在某个已经定位的元素的上执行js

代码

下面的代码演示了如何在页面以及在已经定位的元素上执行js

js.html

```
  <html>
    <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8" />
      <title>js</title>
      <script type="text/javascript" async="" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
```

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

  <body>
    <h3>js</h3>
    <div class="row-fluid">
      <div class="span6 well">
        <a id="tooltip" href="#" data-toggle="tooltip" title="watir-webdriver better than selenium-webdriver">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>
```

js.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'js.html'))
dr.get file_path

# 在页面上直接执行js
dr.execute_script('$("#tooltip").fadeOut();')
sleep(1)

# 在已经定位的元素上执行js
button = dr.find_element(class: 'btn')
dr.execute_script('$(arguments[0]).fadeOut()', button)
sleep(1)

dr.quit()
```

处理alert/confirm/prompt

场景

webdriver中处理原生的js alert confirm 以及prompt是很简单的。具体思路是使用switch_to.alert()方法定位到alert/confirm/prompt。然后使用text/accept/dismiss/send_keys按需进行操做

- text。返回alert/confirm/prompt中的文字信息
- accept。点击确认按钮
- dismiss。点击取消按钮，如果有的话
- send_keys。向prompt中输入文字

代码

下面代码简单的演示了如何去处理原生的alert

alert.html

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

```
        $('#tooltip').tooltip({"placement": "right"});
        $('#tooltip').click(function(){
            alert('watir-webdriver better than selenium-webdriver')
        });
      });
    </script>
  </head>

  <body>
    <div class="row-fluid">
      <div class="span6 well">
        <h3>alert</h3>
        <a id="tooltip" href="#" data-toggle="tooltip" title="watir-webdriver better than selenium-webdriver">hover to see tooltip</a>
      </div>
    </div>
  </body>
  <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

alert.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'alert.html'))
dr.get file_path

# 点击链接弹出alert
dr.find_element(:id, 'tooltip').click()

alert = dr.switch_to.alert
alert.accept()

sleep(1)
dr.quit()
```

---

wait

场景

Wait类的使用场景是在页面上进行某些操作，然后页面上就会出现或隐藏一些元素，此时使用Wait类的until方法来等待这些效果完成以便进行后续的操作。另外页面加载时有可能会执行一些ajax，这时候也需要去Wait的until的等待ajax的请求执行完毕。

具体一点的例子前面也曾出现过，点击一个链接然后会出现一个下拉菜单，此时需要先等待下拉菜单出现方可进行点击菜单项的操作。

在实例化Wait类时，可以传入以下的一些参数

- timeout。总共等待多久,默认5s
- interval。每隔多久检查一次代码块里的值，默认0.2秒
- message。如果超时则显示message
- ignored。代码块中忽略的异常。也就是说如果代码块中抛出了这个异常，那么webdriver将忽略这个异常，继续进行等待，直到满足下面所列举的退出条件为止。默认情况下NoSuchElement异常是被忽略的。

until方法会一直等下去，直到

- 代码块中的内容为true(不为false或没有抛出异常)
- 超时,也就是超过了timeout设置的时间

代码

下面代码演示了点击按钮后如何等待label出现。这个例子其实没有前面的下拉菜单例子实用。

wait.html

```
<html>
  <head>
```

```
    <meta http-equiv="content-type" content="text/html;charset=utf-8" />
    <title>wait</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(){
          $('#btn').click(function(){
            $('<p><span class="label label-info">waitr-webdriver</span></p>').css('margin-top', '1em').insertAfter($(this));
            $(this).addClass('disabled').unbind('click');
          });
      });
    </script>
  </head>

  <body>
    <div class="row-fluid">
      <div class="span6 well">
        <h3>wait</h3>
        <button class="btn btn-primary" id="btn" >Click</button>
      </div>
    </div>
  </body>
  <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

wait.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'wait.html'))
dr.get file_path

# 点击按钮
dr.find_element(:id, 'btn').click()

wait = Selenium::WebDriver::Wait.new()
wait.until { dr.find_element(class: 'label').displayed? }

sleep(2)
dr.quit()
```

定位frame中的元素

场景

处理frame需要用到2个方法，分别是switch_to.frame(name_or_id)和switch_to.default_content()

switch_to.frame方法把当前定位的主体切换了frame里。怎么理解这句话呢？我们可以从frame的实质去理解。frame中实际上是嵌入了另一个页面，而webdriver每次只能在一个页面识别，因此才需要用switch_to.frame方法去获取frame中嵌入的页面，对那个页面里的元素进行定位。

switch_to.default_content方法的话则是从frame中嵌入的页面里跳出，跳回到最外面的原始页面中。

如果页面上只有1个frame的话那么这一切都是很好理解的，但如果页面上有多个frame，情况有稍微有点复杂了。

代码

下面的代码中frame.html里有个id为f1的frame，而f1中又嵌入了id为f2的frame，该frame加载了百度的首页。

frame.html

```
<html>
  <head>
```

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

  <body>
    <div class="row-fluid">
      <div class="span10 well">
        <h3>frame</h3>
        <iframe id="f1" src="inner.html" width="800", height="600"></iframe>
      </div>
    </div>
  </body>
  <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

inner.html

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8" />
    <title>inner</title>
  </head>

  <body>
    <div class="row-fluid">
      <div class="span6 well">
        <h3>inner</h3>
        <iframe id="f2" src="http://www.baidu.com" width="700" height="500"></iframe>
        <a href="javascript:alert('watir-webdriver better than selenium webdriver;')">click</a>
      </div>
    </div>
  </body>
</html>
```

frame.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'frame.html'))
dr.get file_path

# 先到f1再到f2
dr.switch_to.frame('f1')
dr.switch_to.frame('f2')
# 往f2中的百度关键字文本框中输入内容
dr.find_element(id: 'kw').send_keys 'watir-webdriver'

# 直接跳出所有frame
dr.switch_to.default_content

# 再到f1
dr.switch_to.frame('f1')
dr.find_element(link_text: 'click').click

sleep(2)
dr.quit()
```

讨论

假设页面上有A、B两个frame，其中B在A内，那么定位B中的内容则需要先到A，然后再到B。如果是定位A中的内容，那么直接switch_to.frame('A')就可以了;

switch_to.frame的参数问题。官方说name是可以的，但是经过实验发现id也可以。所以只要frame中id和name，那么处理起来是比较容易的。如果frame没有这两个属性的话，你可以直接加上，这对整个页面影响不大;

页面中使用frame会影响页面渲染速度，如果你遇到页面中有多个frame的情况，你完全可以提出1个页面前端性能的缺陷;

如果实在搞不定页面上的frame，送你一句歌词：也许放弃才能靠近你。那么及时放弃跟此frame相关的用例才是明智之举;

action

场景

由于webdriver是要模拟真实的用户操作，因此webdriver的Action类中提供了很多与操作有关的方法。

下面列举一下Action类的一些主要方法

- key_down。模拟按键按下
- key_up。模拟按键弹起
- click
- send_keys
- double_click。鼠标左键双击
- click_and_hold。鼠标左键点击住不放
- release。鼠标左键弹起，可以与click_and_hold配合使用
- move_to。把鼠标移动到元素的中心点
- content_click。鼠标右键点击
- drag_and_drop。拖拽

代码

```
driver.action.key_down(:shift).
              click(element).
              click(second_element).
              key_up(:shift).
              drag_and_drop(element, third_element).
              perform
```

讨论

具体使用方法可以参考api文档。action的api文档算是比较全面了。

上传文件

场景

上传文件的方法是找到上传文件的对象，通常是 `Choose File` No file selected 的对象。然后直接往这个对象send_keys，传入需要上传文件的正确路径。绝对路径和相对路径都可以，但是上传的文件必须存在，否则会报错。

代码

upload_file.html

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

    <body>
      <div class="row-fluid">
```

```
      <div class="span6 well">
        <h3>upload_file</h3>
        <input type="file" name="file" />
      </div>
    </div>
  </body>
  <script src="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.3.2/js/bootstrap.min.js"></script>
</html>
```

upload_file.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
file_path = 'file:///' + File.expand_path(File.join('.', 'upload_file.html'))
dr.get file_path

dr.find_element(name: 'file').send_keys('./upload_file.md')

sleep(2)
dr.quit()
```

下载

场景

webdriver允许我们设置默认的文件下载路径。也就是说文件会自动下载并且存在设置的那个目录中。

下面会给出chrome和firefox浏览器的具体设置方法。

代码

```
# for chrome
profile = Selenium::WebDriver::Chrome::Profile.new
# 设置自动下载
profile['download.prompt_for_download'] = false
# 设置具体路径
profile['download.default_directory'] = "/path/to/dir"

driver = Selenium::WebDriver.for :chrome, :profile => profile

# for firefox
profile = Selenium::WebDriver::Firefox::Profile.new

profile['browser.download.dir'] = "/tmp/webdriver-downloads"
profile['browser.download.folderList'] = 2
# 设置哪些文件自动下载，这里设置的是pdf文件
profile['browser.helperApps.neverAsk.saveToDisk'] = "application/pdf"

driver = Selenium::WebDriver.for :firefox, :profile => profile
```

超时设置

场景

webdriver中可以设置很多的超时时间

- implicit_wait。识别对象时的超时时间。过了这个时间如果对象还没找到的话就会抛出NoSuchElement异常
- script_timeout。异步脚本的超时时间。webdriver可以异步执行脚本，这个是设置异步执行脚本脚本返回结果的超时时间
- page_load。页面加载时的超时时间。因为webdriver会等页面加载完毕在进行后面的操作，所以如果页面在这个超时时间内没有加载完成，那么webdriver就会抛出异常

代码

```
driver = Selenium::WebDriver.for :chrome
# 定位对象时给3s的时间
# 如果3s内还定位不到则抛出异常
driver.manage.timeouts.implicit_wait = 3 # seconds

# 页面加载超时时间设置为5s
driver.manage.page_load = 5 #seconds

# 异步脚本的超时时间设置成3s
driver.manage.script_timeout = 3 #seconds
```

讨论

由于webdriver是通过给driver发送http请求来进行每步操作的，因此就可以设置http请求的超时时间。默认ruby binding的http client超时时间是60s，你可以通过下面的代码来改变这一设置。

```
client = Selenium::WebDriver::Remote::Http::Default.new
client.timeout = 120 # seconds
driver = Selenium::WebDriver.for(:chrome, :http_client => client)
```

Remote Webdriver

场景

简单来说，我们可以把remote webdriver理解成在远程机器上运行webdriver脚本。

想像一下最简单的一个应用场景：你和你的同事两人一起开发一段webdriver脚本，然后你们需要在一个公共的环境去运行这段脚本。为什么要在公共的环境运行？那是因为每个人的开发机器是有差异的，但是如果用同一台测试机的话，那么环境差异的因素就可以基本排除。我们应该经常听到开发说这样的话:"这个bug在我的环境上是好的啊！"。因为运行环境不同而造成的bug比比皆是，因此我们需要一个统一的运行环境来消除差异。

在这样的应用场景下，我们就需要使用remote webdriver，我们在本地开发脚本，然后调用remote webdriver，在测试机器上执行我们的测试。

安装

Remote Webdriver的安装很简单。

首先下载selenium-server-standalone-LAST-VERSION.jar。

然后运行`java -jar selenium-server-standalone.jar`命令。如果没有错误出现的话，这台机器已经被配置成远程机器了，以后webdriver就会在这台机器上启动浏览器，执行脚本。

启动driver

下面的代码可以启动远程机器上的driver，默认情况下这会打开localhost也就是本机上的firefox浏览器

```
    driver = Selenium::WebDriver.for(:remote)
```

如果你的remote webdriver不在本地运行，而且你又想指定除firefox以外的浏览器，那么使用下面的代码 `driver = Selenium::WebDriver.for(:remote, :url => "http://myserver:4444/wd/hub", :desired_capabilities => :chrome)`

通常情况下myserver可以是192.168.x.x之类的ip地址。

另外还可以通过配置`Selenium::WebDriver::Remote::Capabilities`来实现更加定制化的浏览器配置，这个超出本文范围，不做描述。

使用watir-webdriver启动driver

可以使用下面的代码让watir-webdriver也使用remote webdriver模式 `browser = Watir::Browser.new(:remote, {desired_capabilities: :chrome, url: "http://myserver:4444/wd/hub"})`

java版本

```
    // We could use any driver for our tests...
    DesiredCapabilities capabilities = new DesiredCapabilities();

    // ... but only if it supports javascript
```

```
    capabilities.setJavascriptEnabled(true);

    // Get a handle to the driver. This will throw an exception
    // if a matching driver cannot be located
    WebDriver driver = new RemoteWebDriver(capabilities);

    // Query the driver to find out more information
    Capabilities actualCapabilities = ((RemoteWebDriver) driver).getCapabilities();

    // And now use it
    driver.get("http://www.google.com");
```

注意，java版本的代码我没有时间去调试，这里只是把wiki上的代码放出来而已。另外remote server在发生错误时会自动截图，下面是获得截图的代码 `public String extractScreenShot(WebDriverException e) { Throwable cause = e.getCause(); if (cause instanceof ScreenshotException) { return ((ScreenshotException) cause).getBase64EncodedScreenshot(); } return null; }`

python版本

```
    c = webdriver.DesiredCapabilities.CHROME
    driver = webdriver.Remote(command_executor='http://127.0.0.1:4444/wd/hub', desired_capabilities=c)
```

注意，python binding的wiki中使用的启动remote webdriver的代码跟我上面给出的不太相同，可能是因为我的selenium版本较低(30)，最新版本的同学可以试试wiki上的代码。

---

cookie

场景

webdriver可以读取并添加cookie。有时候我们需要验证浏览器中是否存在某个cookie，因为基于真实的cookie的测试是无法通过白盒和集成测试完成的。

另外更加常见的一个场景是自动登陆。有很多系统的登陆信息都是保存在cookie里的，因此只要往cookie中添加正确的值就可以实现自动登陆了。什么图片验证码、登陆的用例就都是浮云了。

webdriver读写cookie的接口有以下一些

- add_cookie。添加cookie，必须有name, value这2个key
- delete_all_cookies。删除所有cookie
- all_cookies。返回所有的cookie
- delete_cookie(name)。删除name这个cookie
- cookie_named。返回特定name的cookie值

代码

下面的代码演示了如何自动登陆百度。其中敏感信息我使用了xxxx来代替。

cookie.rb

```
#encoding: utf-8
require 'selenium-webdriver'

dr = Selenium::WebDriver.for :chrome
url = 'http://www.baidu.com'
dr.get url

p dr.manage.all_cookies
dr.manage.delete_all_cookies
dr.manage.add_cookie(name: 'BAIDUID', value: 'xxxxxx')
dr.manage.add_cookie(name: 'BDUSS', value: 'xxxxxx')

# 重新访问该页面就可以发现已经登陆了
# 当然也可以刷新该页面
dr.get url

sleep(3)
dr.quit()
```