watir-webdriver 主页

来源： http://17test.info/

Watir WebDriver

欢迎莅临 Watir WebDriver 官方中文主页。Watir WebDriver：最优雅的 WebDriver ruby 实现。

Watir Webdriver 的官方主页请点击这里

Watir（音同 Water: 是的，我知道这会使人困惑)是 Web Application Testing in Ruby 这几个英文单词的首字母缩写， 另外 WebDriver 当然是代表 WebDriver 了。

Watir-WebDriver 支持哪些浏览器?

几乎所有的浏览器：比如 Firefox, Chrome 和 IE，除了 Safari。

Watir-WebDriver 好用吗?

我可以说 Watir-WebDriver 好评如潮，这个工具确实很优秀。

快速开始

```
gem install watir-webdriver
```

或者直接将 watir-webdriver 添加到你的 gemfile 文件如果你使用 bundler 的话（如果你没有使用 bundler，那么我建议你立刻试用)。

Watir-WebDriver 是基于 ruby 1.9.2的，但是其也兼容更老的 ruby 版本。

那么继续

准备好了，那么打开 irb。irb 是 Ruby 的解释器，可以逐行解释 ruby 语句。那么在 irb 中逐行敲入下面的代码吧，看看接下来会发生什么神器的事情：

```
require 'watir-webdriver'

b = Watir::Browser.new

b.goto '17test.info/watir_wd_demo.php'

b.text_field(:id => 'entry_0').set 'your name'

b.select_list(:id => 'entry_1').select 'Ruby'

b.select_list(:id => 'entry_1').selected? 'Ruby'

b.div(:class => 'ss-form-entry').button.click

b.text.include? 'Thank you'
```

看，是不是很简单。你已经不错了，喝杯茶犒劳自己一下吧。

天哪，竟然没有 Xpath 选择符！

你可能留意到一件事情，那就是没有出现 xpath 选择符。其实 Watir-WebDriver 是支持 xpath 和 css 选择符的。你会发现 api 是很有点麻烦的，让你感到你不需要使用它。当然在特定的情况下，这些选择符还是很有用的。

天哪，竟然不支持录制和回放

自动化脚本录制工具（比如 Selenium IDE）是给傻瓜用的。严肃的说，在 irb 中解释执行 Watir-WebDriver 代码比使用那些傻乎乎的录制工具要高效和实用的多。再加上你亲手编写的代码要比录制工具自动生成的代码有更好的可读性和稳定性，想像一下，如果你代码里有自动生成的长达100个字符的 css 选择符，这样好吗，你懂的。

Ruby 的威力

Ruby 是一门很神奇很有趣的语言。每天你都会有新的发现，你对 ruby 的爱也会与日俱增。相信我， Watir-WebDriver 也是如此神奇的。

Firefox

来源： http://17test.info/?page_id=486

Firefox

It just works

Firefox 支持是通过一个 JavaScript driver 来实现的，所以其能在所有操作系统上正常运行。

```
b = Watir::Browser.new :firefox
```

Firefox Profiles

默认情况下，Firefox driver 在每次运行的时候都会去创建1个新的 profile 文件，这是推荐的做法。

你可以指定在运行 Firefox 的时候使用一个已存在的 profile 文件，比如你的’default’ profile：

```
b = Watir::Browser.new :firefox, :profile => 'default'
```

你也可以在每次运行测试脚本的时候创建1个新的 Firefox profile，以配置任何你可以在 about:config 面板中能够进行配置的选项。

比如：

```
profile = Selenium::WebDriver::Firefox::Profile.new

profile['browser.download.dir'] = "/tmp/webdriver-downloads"

profile['browser.download.folderList'] = 2
```

```
profile['browser.helperApps.neverAsk.saveToDisk'] = "application/pdf"

b = Watir::Browser.new :firefox, :profile => profile
```

Microsoft Windows 上的本地事件

默认情况下，本地事件（Native Event）在 Windows 上是激活状态的。本地事件的目的是为了提供 webdriver 与操作系统间的底层交互。但是在某些情况下，这些交互会产生一些问题。幸运的是，你可以很容易的使用 Firefox 的 profile 来屏蔽这个功能：

```
profile = Selenium::WebDriver::Firefox::Profile.new

profile.native_events = false

Watir::Browser.new :firefox, :profile => profile
```

在 Firefox 中使用代理

```
profile = Selenium::WebDriver::Firefox::Profile.new

profile.proxy = Selenium::WebDriver::Proxy.new :http => 'myproxy.com:8080:, :ssl => 'myproxy.com:8080'

b = Watir::Browser.new :firefox, :profile => profile
```

使用 Firebug 配合 Watir-WebDriver 进行测试

首先下载 Firebug xpi 文件(一般来说可以直接在 Firefox 上安装扩展)，然后使用下面的代码:

```
profile = Selenium::WebDriver::Firefox::Profile.new
```

```
profile.add_extension "../path/to/firebug.xpi"

b = Watir::Browser.new :firefox, :profile => profile
```

Headless

来源： http://17test.info/?page_id=488

Headless

headless gem 是 Xvfb 的 ruby 封装。headless 使得在无外设的 Linux 服务器上运行图形化界面的应用变得十分简单。该 gem 非常适合在 headless 设备上使用真实的浏览器运行 Watir-WebDriver。

一个例子

```
require 'watir-webdriver'

require 'headless'

headless = Headless.new

headless.start

b = Watir::Browser.start 'www.google.com'

puts b.title

b.close
```

```
headless.destroy
```

一个 Cucumber 的例子

将下面的代码添加到 env.rb 文件中：

```
if ENV['HEADLESS']
  require 'headless'
  headless = Headless.new
  headless.start
  at_exit do
    headless.destroy
  end
end
```

Chrome

来源： http://17test.info/?page_id=478

Chrome

ChromeDriver

你可以通过下载与平台相关的 ChromeDriver 的二进制文件来获得 Chrome 浏览器的支持。

ChromeDriver 的下载地址见这里

```
b = Watir::Browser.new :chrome
```

Chrome Profiles

```
profile = Selenium::WebDriver::Chrome::Profile.new

profile['download.prompt_for_download'] = false

profile['download.default_directory'] = "/path/to/dir"

b = Watir::Browser.new :chrome, :profile => profile
```

详细的 profile 选项请参见这里

Chrome 的开关

```
b = Watir::Browser.new :chrome, :switches => %w[--ignore-certificate-errors --disable-popup-blocking --disable-translate]
```

详细的开关选项参加这里。

在 Chrome 中使用代理

```
b = Watir::Browser.new :chrome, :switches => %w[--proxy-server=myproxy.com:8080]
```

Internet Explorer

来源： http://17test.info/?page_id=500

Internet Explorer

Windows 上的 IE

Internet Explorer 只支持 Windows 操作系统(靠！)，并且只有当你将所有区域的**保护模式**设置成一致时(也就是说保护模式全部是”开启”或”关闭”)才能正常运行。

```
b = Watir::Browser.new :ie
```

Internet Explorer Config

Watir-WebDriver 使用你标准的浏览器配置，所以你可以在运行测试前手动更改这些设置。

模拟特殊按键

来源： http://17test.info/?page_id=558

模拟特殊按键

使用.send_keys 方法可以模拟特殊的键盘按键(比如 shift)，其参数是你所需要模拟的按键的符号表示(symbolic)。

```
b.send_keys :enter
```

也可以这样做：

```
b.element.send_keys [:control, 'a'], :backspace
```

你还可以修改 click 方法的行为，使得点击可以配合按键一起进行：

```
b.element.click(:shift, :control)
```

支持的按键键名列表如下：

:null

:cancel

:help

:backspace

:tab

:clear

:return

:enter

:shift

:left_shift

:control

:left_control

:alt

:left_alt

:pause

:escape

:space

:page_up

:page_down

:end

:home

:left

:arrow_left

:up

:arrow_up

:right

:arrow_right

:down

:arrow_down

:insert

:delete

:semicolon

:equals

:numpad0

:numpad1

:numpad2

:numpad3

:numpad4

:numpad5

:numpad6

:numpad7

:numpad8

:numpad9

:multiply

:add

:separator

:subtract

:decimal

:divide

:f1

:f2

:f3

:f4

:f5

:f6

:f7

:f8

:f9

:f10

:f11

:f12

:meta

:command

Frames

来源： http://17test.info/?page_id=512

Frames

在 Watir-WebDriver 中处理 frame 是非常简单的，就跟处理其他页面元素一样：

```
b.frame(:id => "content_ifr").send_keys "hello world"
```

关于本站

来源： http://17test.info/?page_id=572

关于本站

本站的英文版本是由来自 Brisbane, Australia 的 Alister Scott 撰写的。

本站中文版本是由官方正式授权，乙醇翻译的。由于作者水平有限，所以翻译中的一些错误难以避免，希望大家能够指出并谅解。

浏览器 Cookies

来源： http://17test.info/?page_id=540

浏览器 *Cookies*

从 Watir-WebDriver 0.5.2起，作者提供了处理 cookie 的 API,使用起来非常简单。

```
require 'watir-webdriver'

browser = Watir::Browser.new

browser.cookies.clear

browser.cookies.add 'foo', 'bar', :path => "/", :expires => 10.days.from_now, :secure => true

browser.cookies.delete 'foo'

browser.cookies.to_hash
```

浏览器证书

来源： http://17test.info/?page_id=528

浏览器证书

Firefox

默认情况下，Firefox driver 自己会正确的处理不信任的证书。

如有你拥有一个信任的证书，但是仍然提示其他的一些证书错误，比如 hostname 不匹配(也就是说在测试时使用正式环境的证书)，那么你需要做下面的事情：

```
profile = Selenium::WebDriver::Firefox::Profile.new

profile.assume_untrusted_certificate_issuer = false

b = Watir::Browser.new :firefox, :profile => profile
```

The reason this is needed is explained here.

Chrome

Chrome 上只需要在实例化 Browser 时传入相应的开关(switch)就可以忽略无效证书了：

```
Watir::Browser.new :chrome, :switches => ['--ignore-certificate-errors']
```

等待页面加载完毕

来源： http://17test.info/?page_id=509

等待页面加载完毕

当你的测试页面包含一些动态的交互，比如有许多 AJAX 时，等待页面加载完毕往往是一件很头痛的事情。

显示的等待

Watir-Webdriver 提供了4种方法使你的等待体验不再痛苦(将那些丑陋的 sleep 语句从代码中移除掉吧):

- Watir::Wait.until { … }: 等待，直到 block 中的语句为 true
- object.when_present.set: 当 object 出现时去操作 objcet，这个例子里是调用了 object 的 set 方法，当 object 存在的时候
- object.wait_until_present:等待，直到 object 出现
- object.wait_while_present:等待，直到 object 消失

默认情况下，上面的方法会等待30秒，不过你也可以在调用方法时传入1个参数来增加(减少)等待时间，如果需要的话。

```
require 'watir-webdriver'

b = Watir::Browser.start '17test.info/watir_wd_demo.php'

b.select_list(:id => 'entry_1').wait_until_present

b.text_field(:id => 'entry_0').when_present.set 'your name'

b.button(:value => 'Submit').click

b.button(:value => 'Submit').wait_while_present

Watir::Wait.until { b.text.include? 'Thank you' }
```

隐式的等待

另一个选择是使用 WebDriver 的隐式等待。脚本每次在定位测试元素的时候总是会**隐式**的等待一段时间(在这段时间内如果元素无法被定位到，那么脚本就会抛出异常)，你可以通过改变 driver 的属性来自行指定这个**隐式**的等待时间：

```
require 'watir-webdriver'

b = Watir::Browser.new

b.driver.manage.timeouts.implicit_wait = 3 #3 秒
```

注意：使用隐式等待会使得你的测试用例运行速度变慢并且当用例失败时其失败原因会更难定位。

移动设备

来源： http://17test.info/?page_id=503

移动设备

使用 watir-webdriver 测试移动站点有下面3种方法：

在真实设备的内置浏览器中进行测试；

在模拟器的内置浏览中；

或在配置了相同分辨率和 user-agent 的桌面浏览器(也就是将 PC 浏览器”模拟”成移动设备的浏览器)中进行测试；

在真实的设备上(iOS 或者 Android)上进行测试是需要花费点银子的(当然了，要买手机和平板的)，而且其运行速度比起桌面浏览器还是要差上一截。这里是如何配置 iOS 和 Android 测试环境的详细文档。在真正的苹果设备上测试需要苹果注册苹果的开发者账号，大约需要花费99美金。

更加简单高效的方法是使用桌面浏览器，将其”模拟”成移动浏览器进行测试。这样的话，配合使用 webdriver-user-agent gem 将使得一切变得非常简单。

```
require 'watir-webdriver'

require 'webdriver-user-agent'

driver = UserAgent.driver(:browser => :chrome, :agent => :iphone, :orientation =>
 :landscape)

browser = Watir::Browser.new driver

browser.goto 'tiffany.com'

browser.url.should == 'http://m.tiffany.com/International.aspx'
```

该gem目前支持将firefox和chrome”模拟成”iphone, ipad, android移动电话, 和android平板。当然了，横屏和竖屏都是支持的。

浏览器下载

来源： http://17test.info/?page_id=531

浏览器下载

最简单最好的处理文件下载对话框的方式就是完全的避免对话框弹出。

可以在代码里告诉浏览器自动的将文件下载到指定目录，然后在测试用例中访问该目录进行验证。

Firefox

```
download_directory = "#{Dir.pwd}/downloads"

download_directory.gsub!("/", "\\") if Selenium::WebDriver::Platform.windows?

profile = Selenium::WebDriver::Firefox::Profile.new

profile['browser.download.folderList'] = 2 # custom location

profile['browser.download.dir'] = download_directory

profile['browser.helperApps.neverAsk.saveToDisk'] = "text/csv,application/pdf"

b = Watir::Browser.new :firefox, :profile => profile
```

关于 Firefox 的所有配置项可以通过在地址栏中输入’about:config’进行查看。

If you want to know a way to work out the file types (eg. application/pdf) then you can read the following blog post for an step by step guide.

如果你想知道如何处理特定类型的文件，请阅读这篇博文。

Chrome

```
download_directory = "#{Dir.pwd}/downloads"

download_directory.gsub!("/", "\\") if  Selenium::WebDriver::Platform.windows?

profile = Selenium::WebDriver::Chrome::Profile.new

profile['download.prompt_for_download'] = false
```

```
profile['download.default_directory'] = download_directory

b = Watir::Browser.new :chrome, :profile => profile
```

基本的浏览器验证

来源： http://17test.info/?page_id=524

基本的浏览器验证

最简单最优雅的处理浏览器基本验证(basic browser authentication)的方法是在 url 中加入用户名和密码，以绕开验证对话框。

```
require 'watir-webdriver'

b = Watir::Browser.start 'http://admin:password@yourwebsite.com'
```

所见即所得编辑器

来源： http://17test.info/?page_id=561

所见即所得编辑器

有两种方法可以通过 Watir-WebDriver 向所见即所得编辑器(应该指的是富文本编辑器)中输入文字：

- 定位编辑器所在的 iFrame，然后使用.send_keys 方法(缺点是浏览器必须在前台运行)
- 在浏览器上执行 javascript,通过 js 脚本去设置编辑器的值（最可靠的办法）

CKEditor

```
require 'watir-webdriver'

b = Watir::Browser.new :firefox

b.goto 'http://ckeditor.com/demo'

b.execute_script("CKEDITOR.instances['editor1'].setData('hello world');")

b.frame(:title => 'Rich text editor, editor1, press ALT 0 for help.').send_keys 'hello world again'
```

TinyMCE Editor

```
require 'watir-webdriver'

b = Watir::Browser.new

b.goto 'http://tinymce.moxiecode.com/tryit/full.php'

b.execute_script("tinyMCE.get('content').execCommand('mceSetContent',false, 'hello world' );")

b.frame(:id => "content_ifr").send_keys 'hello world again'
```

Safari

来源： http://17test.info/?page_id=625

Safari

Watir-WebDriver 支持 Safari 浏览器了，感谢 SafariDriver 的正式发布。

目前 SafariDriver 的缺点是配置比较烦琐，其需要 Safari 扩展（版本5+）。除非有人在网上发布了该扩展，否则你必须自己去构建它。

构建 extension 的步骤

1. 首先，从 Apple 那里获得创建和安装 Safari 扩展的许可。你必须注册 Safari 开发者账号(免费的)，然后再将许可证书下载到本地。

2. 现在，你需要构建扩展了。第一步， check out selenium 的源码:

```
svn co http://selenium.googlecode.com/svn/trunk selenium3.
```

3. 然后 cd 到这个目录进行构建。

```
cd selenium
./go safari4.
```

4. 最后安装扩展。

- 打开 Safari
- 确保 在设置 Advanced Preferences 时 develop 菜单可以正常显示
- 打开 Extension Builder (Develop > Show Extension Builder)
- 从$SELENIUM*CHECKOUT*LOCATION/build/javascript/safari-driver/SafariDriver.safariextension 添加新的扩展

使用 Watir-WebDriver 和 Safari 进行测试

这跟在其他浏览器中进行测试没什么区别。

```
require 'watir-webdriver'
b = Watir::Browser.new :safari
```

用户须知：

- browser.execute_script 的行为会有些奇怪
- 扩展无法加载本地的 html 文件，也就是说无法打开本地的 html 文件进行测试
- 目前无法实现文件上传的操作
- 无法确定是否能够在代码中使用编程的手段来配置浏览器，比如配置 user agent。

Enjoy!

浏览器代理

来源： http://17test.info/?page_id=537

浏览器代理

实例：在 Firefox 中使用代理

```
profile = Selenium::WebDriver::Firefox::Profile.new

profile.proxy = Selenium::WebDriver::Proxy.new :http => 'my.proxy.com:8080', :ssl => 'my.proxy.com:8080'

browser = Watir::Browser.new :chrome, :profile => profile
```

实例：在 Chrome 中使用代理

```
switches = '--proxy-server=my.proxy.com:8080'

browser = Watir::Browser.new :chrome, :switches => switches
```

新开浏览器窗口

来源： http://17test.info/?page_id=534

新开浏览器窗口

当一个新的浏览器窗口打开时，你可以使用’use’方法来处理这个新窗口。

```
browser.window(:title => "annoying popup").use do

  browser.button(:id => "close").click

end
```

更多的例子请参考这里

Javascript 对话框

来源： http://17test.info/?page_id=546

Javascript 对话框

JavaScript dialogs are fairly common in web applications.

在 web 应用中，JavaScript 对话框是十分常见的。

Watir-WebDriver has an inbuilt library for handling the dialogs, and capturing values from the dialogs. First, require the extension:

Watir-WebDriver 内建了处理这些对话框的方法，并且可以返回对话框中显示的内容。首先，加载这个扩展：

```
require "watir-webdriver/extensions/alerts"
```

JAVASCRIPT ALERTS

```
browser.alert do
  browser.button(:value => 'Alert').click
end #=> 'the alert message'
```

JAVASCRIPT CONFIRMS

```
browser.confirm(true) do
  browser.button(:value => 'Confirm').click
end #=> 'the confirm message'
```

JAVASCRIPT PROMPT

```
browser.prompt('hello') do
  browser.button(:value => 'Prompt').click
end #=> { :message => 'foo', :default_value => 'bar' }
```

可选方法

如果你使用上面的方法时遇到了麻烦，你可以自行覆盖 JavaScript functions，这样一来原来应该显示的对话框就可以在触发时不显示了。

```
# 使 alert 方法返回空

browser.execute_script("window.alert = function() {}")

# 使 prompt 返回特定的字符串，用来模拟用户的输入

browser.execute_script("window.prompt = function() {return 'my name'}")

# 使 prompt 方法返回 null 用来模拟用户点击了 Cancel(取消)按钮

browser.execute_script("window.prompt = function() {return null}")

# 使 confirm 方法返回 true 用来模拟用户点击了 OK(确定)按钮

browser.execute_script("window.confirm = function() {return true}")

# 使 confirm 方法返回 false 用来模拟用户点击了 Cancel(取消)按钮

browser.execute_script("window.confirm = function() {return false}")
```

测试页面性能

来源： http://17test.info/?page_id=549

测试页面性能

Watir-WebDriver-Performance gem 提供在访问页面的同时进行页面性能度量的功能，其使用的是 W3C 页面性能度量指标。这是一个完美的捕获响应性能指标的解决方案，其使用方法非常直观和简单，不过目前只支持 Chrome 和 IE9l 浏览器。

```
require 'watir-webdriver'

require 'watir-webdriver-performance'

b = Watir::Browser.new :chrome

10.times do

  b.goto 'http://17test.info'

  load_secs = b.performance.summary[:response_time]/1000

  puts "Load Time: #{load_secs} seconds."

end
```

其统计结果如下：

```
Load Time: 3.701 seconds.

Load Time: 0.694 seconds.

Load Time: 1.874 seconds.

Load Time: 1.721 seconds.

Load Time: 2.096 seconds.

Load Time: 0.823 seconds.

Load Time: 2.362 seconds.

Load Time: 1.008 seconds.

Load Time: 1.761 seconds.
```

Load Time: 2.066 seconds.

支持的性能指标：

- :summary
- :navigation
- :memory
- :timing

Page Object

来源： http://17test.info/?page_id=552

Page Object

Page Object 设计模式是将测试页面及页面上的测试封装成可重用的类的方法。watir-webdriver 的 wiki 详细阐述了这个设计模式。

下面的 gem 能够帮助你实现 page objects 设计模式：

- Watir Page Helper
- LoadableComponent