Testing Your Website With Selenium (and Perl)

Tim Vroom - vroom@blockstackers.com

Pittsburgh Perl Workshop - 2014

About Me

- Tim Vroom
- Work at slashdot.org
- Started writing Selenium tests for work after attending the 2013 YAPC
- VROOM on CPAN
- @vroomtim on twitter

GitHub Repo

- https://github.com/tvroom/selenium-testing-with-perl

Short url of (http://tinyurl.com/sel-perl)

(slides / example code / web pages / scripts) — final updates may come in next few days

What is Selenium?

- A tool for web-browser automation & testing
- Allows for functional testing that also runs JavaScript loaded on the page
- Can run tests against multiple web browsers

What we'll cover

- Running your first selenium test
- SeleniumRC vs Selenium Webdriver
- Core Perl libraries to consider using
- Locators & Handling Waits
- Variety of Example Tests

Steps to running your first Selenium Test with Perl

1. Download Selenium Server - selenium-server-standalone-2.44.0.jar (latest currently) at http://www.seleniumhq.org/download/

2. Run the server:

```
java -jar selenium-server-standalone-2.44.0.jar
```

3. Install Selenium::Remote::Driver from CPAN

Sample PPW Test

```
use Selenium::Remote::Driver;
use Test::More;

my $driver = Selenium::Remote::Driver->new(default_finder => 'css');
$driver->get('http://www.google.com');
$driver->find_element('q','name')->send_keys('Pittsburgh Perl Workshop');
$driver->set_implicit_wait_timeout(5000);
#$driver->find_element('//*[@id="rso"]/div[2]/li[1]/div/h3/a','xpath')->click();
$driver->find_element('#rso li h3 a')->click();
my $title = $driver->get_title();
like($title, qr/The Pittsburgh Perl Workshop/, "Title for PPW is what is expected");
$driver->find_element('Talks and Schedule', 'link_text')->click();
$driver->find_element('Schedule','link_text')->click();
$driver->find_element('Sunday','link_text')->click();
$driver->find_element('Selenium','partial_link_text')->click();
$title = $driver->get_title();
like($title, qr/Testing.*Selenium/, "Title seems to be about testing with Selenium");
sleep 5; # For demo purposes wait a bit before closing so we can see the final page
$driver->quit();
done_testing();
```

Versions of Selenium

Remote Control - Vs - WebDriver

Selenium RC (Selenium 1)

- Selenium Remote Control serves as middle layer between code and browser
- When you run a test it pops one window for the Remote Control, and then that window operates your test window via javascript
- Operates browser actions by running Javascript to mimic those actions
- Due to translation in middle layer sometimes did things differently than the browser actually would

Selenium 2 (WebDriver API)

- Talks directly to browser via WebDriver API calls
- Browser makers have built WebDriver support right into the browse
- Typically faster running tests due to more direct nature
- Behaves more consistently with actual browser behavior, because you're executing native code with-in the browser, which matches up with WebDriver API Calls

Perl Modules for Selenium RC tests

- WWW::Selenium
- Test::WWW::Selenium
- WWW::Selenium::Util
- WWW::Selenium::NetworkCapture

Perl Modules for Selenium Webdriver Tests

- Selenium::Remote::Driver
- Test::Selenium::Remote::Driver
- Selenium::Remote::WebElement

Sample code for this presentation

- Will primarily show code interacting with Selenium Webdriver (Selenium::Remote::Driver Perl Module)
- Repo will contain examples with SeleniumRC in most cases for comparison

Key Concepts While Writing Tests

Locating Elements

Locate Via…

css

$driver->find_element('.header','css);

name
<input type="text" name="user" value="" />
$driver->find_element('user','name');

link_text
Schedule
$driver->find_element('Schedule','link_text');

xpath
<a href="/">
$driver->find_element('//*[@id="logo"]/a','xpath');

and more…
class, class_name, id, link, partial_link_text, tag_name

find_element

Takes a search specifier, and finder type
$driver->find_element($search_string, $finder_type)

If no finder type is specified, it goes with the default of xpath

If you want to override the default_finder type, you can do so at driver initialization with:

Selenium::Remote::Driver->new(‘default_finder’ => ‘css’)

find_element() calls return Selenium::Remote::WebElement objects

Selenium::Remote::WebElement defines methods to take action on the elements or test attributes of them, look at the documentation to see what’s available

This allows you to chain calls like:
$driver->find_element(‘.foo’,’css’)->click();
$driver-find_element(‘username’,’name’)->send_keys(‘bob’);

Managing Waits For Your Tests

Waits

Tests need to properly wait for completion of actions, before looking for an element provided as a result of previous action

When you….

Load a new page

Change current page via Ajax or JS

Range of Waits

Too short/ Non-Existent = failing tests

Too long = (long sleeps), slow test suites

Just right - wait only as long as you need to = passing tests + fast test suites

Selenium::Remote::Driver waits for Page loads

polls and waits for up to ‘page load’ timeout (default 180 secs)

You can change this timeout via:

```
$driver->set_timeout('page load', 10000);
# 10 second timeout in ms
```

Selenium::Remote::Driver waits for in page element changes

poll and wait up to implicit_wait_timeout for
find_element - default is 0 ms

You can change this timeout via:

$driver->set_implicit_wait_timeout(5000)
change to 5 secs in ms

Selenium::Remote::Driver (wait for condition)

Selenium RC has a method, which runs a piece of JS checking for a condition for up to a timeout threshold

```
# wait until a new article is inserted
$sel->wait_for_condition('window.$
(".article").length >'.$orig_count, 10000);
```

You could write your own method to extend Selenium::Remote::Driver to provide the same functionality (sample of this in repo)

Example Tests

Running JS in your tests

$driver->execute_script() method to run JS and return results

```
use Selenium::Remote::Driver;
use Test::More;
my $driver = Selenium::Remote::Driver->new();
$driver->navigate("http://timvroom.com/selenium/js_testing.html");
my $shopping_count = $driver->execute_script(
    'return $(".shopping-list li”).length'
);
ok($shopping_count >= 3, "Shopping list has at least 3 items");

done_testing();
```

Verify JS library inclusion

```
use Selenium::Remote::Driver;
use Test::More;

my $driver = Selenium::Remote::Driver->new();

$driver->navigate("http://timvroom.com/selenium/js_testing.html");

ok(js_defined("window.jQuery"), "jQuery installed on $_[0]");
ok(js_defined("window.jQuery.cookie"), "jQuery cookie plugin installed");
ok(js_defined("window.Handlebars"), "has Handlebars installed");

done_testing();

sub js_defined {
        my $val = shift;
        my $ret = $driver->execute_script("return !(typeof $val === 'undefined')");
        return $ret;
}
```

Capturing Network Data

WWW::Selenium::NetworkCapture - gives you a way of seeing all the HTTP requests, and response information generated by your Selenium session

Webdriver doesn’t offer this capability, but you could run your tests through a webproxy, and then analyze results

Ex) Test for missing failed resources loading (Source HTML)

```
<html>
<head>
	<title>Broken resources test page</title>
</head>
<body>
	<h1>Broken resources test page</h1>
	<style type="text/css">
	#logo {
		background:transparent url(/missing_logo_call_from_css.png) no-repeat top left;
		float:left;
		height: 20px;
		width: 120px;
	}
	</style>
	<div id="logo"></div>
	<img src="broken_image.png" alt="broken_image">
	<script src="broken_script.js" type="text/javascript">
</body>
</html>
```

Test Script

```
use WWW::Selenium;
use WWW::Selenium::NetworkCapture;
use Data::Dumper;
use Test::More;

my $sel=WWW::Selenium->new(
        browser_url => "http://timvroom.com/selenium/ga_test.html"
);
$sel->start();
$sel->{session_id} = $sel->get_string(
        "getNewBrowserSession",
        $sel->{browser_start_command},
        $sel->{browser_url},
        undef,
        'captureNetworkTraffic=true'
);

$sel->open("http://timvroom.com/selenium/broken_resources.html");
$sel->wait_for_page_to_load(10000);

my $traffic_xml = $sel->get_string('captureNetworkTraffic', 'xml');

my $netcap = WWW::Selenium::NetworkCapture->new($traffic_xml);
my @stat_code_urls = map { [$_->{statusCode}, $_->{url}]} grep { $_->{url} !~ /favicon.ico/ } grep { $_->{statusCode}
=~/^(4|5)/  }  @{$netcap->{dom}{entry}};

ok(!@stat_code_urls, "Resources loaded w/o 40x or 50x errors for $key $url\n") or diag Dumper(@stat_code_urls);
$sel->stop();
done_testing();
```

Results

```
not ok 1 - Resources loaded w/o 40x or 50x errors for
#
#   Failed test 'Resources loaded w/o 40x or 50x errors for
# '
#   at missing_resources.t line 27.
# $VAR1 = [
#           [
#             '404',
#             'http://timvroom.com/selenium/broken_image.png'
#           ],
#           [
#             '404',
#             'http://timvroom.com/selenium/broken_script.js'
#           ],
#           [
#             '404',
#             'http://timvroom.com/missing_logo_call_from_css.png'
#           ]
#         ];
1..1
# Looks like you failed 1 test of 1.
```

```
use WWW::Selenium;
use WWW::Selenium::NetworkCapture;
use Data::Dumper;
use Test::More;

my $sel=WWW::Selenium->new(
        browser_url => "http://timvroom.com"
);
$sel->start();
$sel->{session_id} = $sel->get_string(
        "getNewBrowserSession",
        $sel->{browser_start_command},
        $sel->{browser_url},
        undef,
        'captureNetworkTraffic=true'
);

$sel->open("http://timvroom.com/selenium/ga_test.html");
$sel->wait_for_page_to_load(10000);

my $traffic_xml = $sel->get_string('captureNetworkTraffic', 'xml');

my $netcap = WWW::Selenium::NetworkCapture->new($traffic_xml);

my @requests = @{$netcap->{dom}{entry}};

my @ga_track_pv = grep { $_->{url} =~ m|www.google-analytics.com/r/collect\?.*t=pageview| } @requests;
my @non_200_ga_track = grep { !$_->{statusCode} == 200 } @ga_track_pvok(@ga_track_pv == 1, "Got exactly 1 GA track
pageview request");
ok(!@non_200_ga_track, "All GA track pageview requests got a 200 statusCode");
```

Check for Google Analytics Tracking

Results:

```
ok 1 - Got exactly 1 GA track pageview request
ok 2 - All GA track pageview requests got a 200
statusCode
1..2
```

Questions?

Talk repo at:

https://github.com/tvroom/selenium-testing-with-perl

Short url of (http://tinyurl.com/sel-perl)

Bonus Slides

(things that might be of interest, or interesting but didn’t fit in the time of the talk)

Other Testing Recipes / Ideas

Problem: How to automatically test captcha based flows

Have captcha solving API available - (only in test mode) so we can test using correct captchas, and error messages for bad captchas

Problem: find url of a page to set whose data or characteristics matches a certain set of criteria

Example) - figure out the url of a story to test of a certain age, with a certain number of comments.

Use an API, redirect script, or generated config file to give you the url to navigate to

Problem: posting limits from repeated posting tests

Have an API to clear posting limits as part of tests (or a setup / cleanup API) for other types of tasks that need it

Additional Reading

- JsonWireProtocol - learn about the protocol (and capabilities provided) in the Protocol used to power WebDriver

- https://code.google.com/p/selenium/wiki/JsonWireProtocol

- Particularly - the command reference section: https://code.google.com/p/selenium/wiki/JsonWireProtocol#Command_Reference