Selenium WebDriver Recipes in Node.js

The Problem Solving Guide to Selenium WebDriver



Zhimin Zhan

Selenium WebDriver Recipes in Node.js

The problem solving guide to Selenium WebDriver in JavaScript

Zhimin Zhan

This book is available at http://leanpub.com/selenium-webdriver-recipes-in-nodejs

This version was published on 2024-10-28

ISBN 978-1537328256



This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2016 - 2024 Zhimin Zhan

Also By Zhimin Zhan

Practical Performance and Load Testing

Practical Continuous Testing

Practical Desktop App Test Automation with Appium

API Testing Recipes in Ruby

Selenium WebDriver Recipes in Python

Learn Swift Programming by Examples

Learn Ruby Programming by Examples

Selenium WebDriver Recipes in Java

Selenium WebDriver Recipes in Ruby

Watir Recipes

Practical Web Test Automation with Selenium WebDriver

Contents

Preface . . . . . **i**
Who should read this book ii
How to read this book ii
Recipe test scripts ii
Send me feedback iii

1. Introduction **1**
1.1 Selenium language bindings 1
1.2 Install Node.JS 3
1.3 Write first script with JavaScript editor 5
1.4 Install Selenium WebDriver 6
1.5 Run script 6
1.6 Cross browser testing 7
1.7 Mocha Test Framework 10
1.8 Create package.json file 13
1.9 Run recipe test scripts 14

2. Locating web elements **16**
2.1 Start browser 16
2.2 Find element by ID 17
2.3 Find element by Name 18
2.4 Find element by Link Text 18
2.5 Find element by Partial Link Text 18
2.6 Find element by XPath 19
2.7 Find element by Tag Name 20
2.8 Find element by Class 21
2.9 Find element by CSS 21
2.10 Chain findElement to find child elements 21
2.11 Use locator name as JSON attribute 22

2.12 Find multiple elements 22

3. Hyperlink 23
3.1 Click a link by text 23
3.2 Click a link by ID 23
3.3 Click a link by partial text 24
3.4 Click a link by XPath 24
3.5 Click Nth link with exact same label 25
3.6 Click Nth link by CSS Selector 25
3.7 Verify a link present or not? 25
3.8 Getting link data attributes 26
3.9 Test links open a new browser window 26

4. Resources 28
4.1 Books 28
4.2 Web Sites 29
4.3 Blog 29
4.4 Tools 30

Preface

Selenium WebDriver comes with five core language bindings: Java, C#, Ruby, Python and JavaScript (Node.js). I have written Selenium WebDriver recipes for all other four languages except JavaScript. This is not because I don't know JavaScript, as a matter of fact, I have a long history of using JavaScript and I am still using it to develop dynamic web applications. JavaScript, to me, seems always associated with Web and HTML, rather than being a standalone scripting language. One day, overcoming my prejudice, I gave Selenium WebDriver with Node.js a go. I was very impressed. I found Selenium WebDriver in JavaScript is lightweight, quick setup, and above all, test execution is fast, really fast.

Over the years, JavaScript has evolved far beyond a client-side scripting language, and has become a powerful programming language which can also be used to create server-side applications, or even desktop applications (Microsoft's new programmer's editor Visual Studio Code is built with Node.js). Node.js is a cross-platform runtime environment that uses open-source V8 JavaScript Engine execute JavaScript code as an application. As you have probably noticed, the demand for JavaScript programmers is high (see JavaScript developer salary graph in US [1]). It comes to no surprises that Selenium WebDriver includes JavaScript as one of the five official language bindings.

The purpose of this book is to help motivated testers work better with Selenium WebDriver. The book contains over 170 recipes for web application tests with Selenium WebDriver in JavaScript. If you have read my other book: *Practical Web Test Automation*[2], you probably know my style: practical. I will let the test scripts do most of the talking. These recipe test scripts are 'live', as I have created the target test site and included offline test web pages. With both, you can:

1. **Identify** your issue
2. **Find** the recipe
3. **Run** the test case
4. **See** test execution in your browser

[1] http://www.indeed.com/salary/q-Javascript-Developer-l-United-States.html
[2] https://leanpub.com/practical-web-test-automation

Who should read this book

This book is for testers or programmers who are writing (or want to learn) automated tests with Selenium WebDriver. In order to get the most of this book, basic (very basic) JavaScript skills is required.

How to read this book

Usually, a 'recipe' book is a reference book. Readers can go directly to the part that interests them. For example, if you are testing a multiple select list and don't know how, you can look up in the Table of Contents, then go to the chapter 8. This book supports this style of reading.

If you are new to Selenium WebDriver, I recommend you to try out the recipes from the front to back. The recipes in the first half of the book are arranged according to their levels of complexity, I believe readers can get the pattern of testing with Selenium and gain confidence after going through them.

Recipe test scripts

To help readers to learn more effectively, this book has a dedicated site[3] that contains the recipe test scripts, test web pages and related resources. For access code, please see the Resources section of this book.

As an old saying goes, "*There's more than one way to skin a cat.*" You can achieve the same testing outcome with test scripts implemented in different ways. The recipe test scripts in this book are written for simplicity, and there is always room for improvement. But for many, to understand the solution quickly and get the job done are probably more important.

If you have a better and simpler way, please let me know.

All recipe test scripts are Selenium WebDriver 3 compliant, and can be run against Chrome, Firefox and Internet Explorer on multiple platforms. I plan to keep the test scripts updated with the latest stable Selenium version.

[3] http://zhimin.com/books/selenium-recipes-nodejs

Send me feedback

I would appreciate your comments, suggestions, reports on errors in the book and the recipe test scripts. You may submit your feedback on the book site.

Zhimin Zhan

Brisbane, Australia

1. Introduction

Selenium is a free and open source library for automated testing web applications. Selenium was originally created in 2004 by Jason Huggins, it merged with another test framework WebDriver in 2011 (that's why is named 'selenium-webdriver') led by Simon Stewart at Google (update: Simon now works at FaceBook). As WebDriver is a W3C standard[1], it gains support from all major browser vendors, as a result, Selenium WebDriver quickly become the de facto framework for automated testing web applications.

1.1 Selenium language bindings

Selenium tests can be written in multiple programming languages such as Java, C#, Python, Ruby and JavaScript (the core ones). All examples in this book are written in Selenium with JavaScript binding. As you will see from the examples below, the use of Selenium in different bindings are very similar. Once you master one, you can apply it to others quite easily. Take a look at a simple Selenium test script in five different language bindings: JavaScript, Java, C#, Python and Ruby.

JavaScript:

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();

driver.get('http://www.google.com/ncr');
driver.findElement(webdriver.By.name('q')).sendKeys('webdriver');
driver.findElement(webdriver.By.name('btnG')).click();
driver.wait(webdriver.until.titleIs('webdriver - Google Search'), 1000);
driver.quit();
```

Java:

[1] https://www.w3.org/TR/webdriver/

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class GoogleSearch  {
  public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.google.com");
    WebElement element = driver.findElement(By.name("q"));
    element.sendKeys("Hello Selenium WebDriver!");
    element.submit();
    System.out.println("Page title is: " + driver.getTitle());
  }
}
```

C#:

```
using System;
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.Support.UI;

class GoogleSearch
{
  static void Main()
  {
    IWebDriver driver = new FirefoxDriver();
    driver.Navigate().GoToUrl("http://www.google.com");
    IWebElement query = driver.FindElement(By.Name("q"));
    query.SendKeys("Hello Selenium WebDriver!");
    query.Submit();
    Console.WriteLine(driver.Title);
  }
}
```

Python:

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://www.google.com")

elem = driver.find_element_by_name("q")
elem.send_keys("Hello WebDriver!")
elem.submit()

print(driver.title)
```

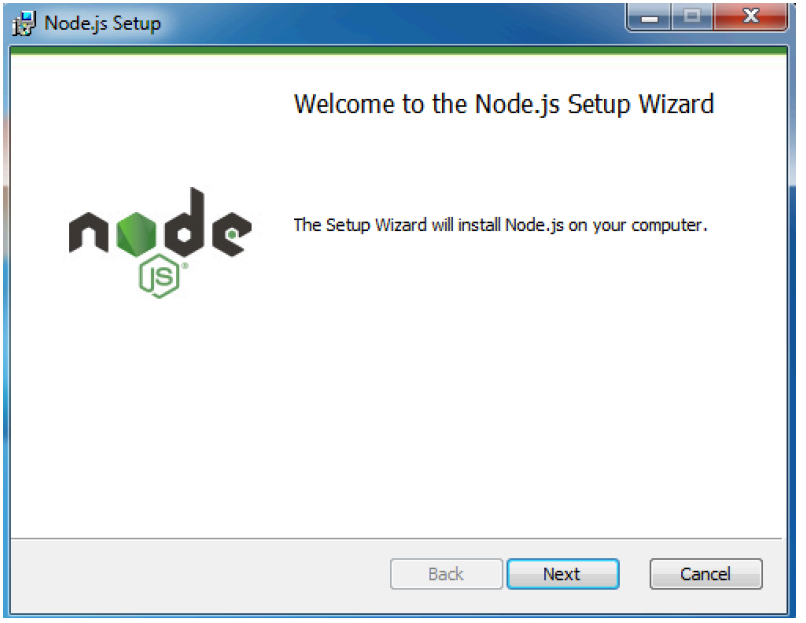
Ruby:

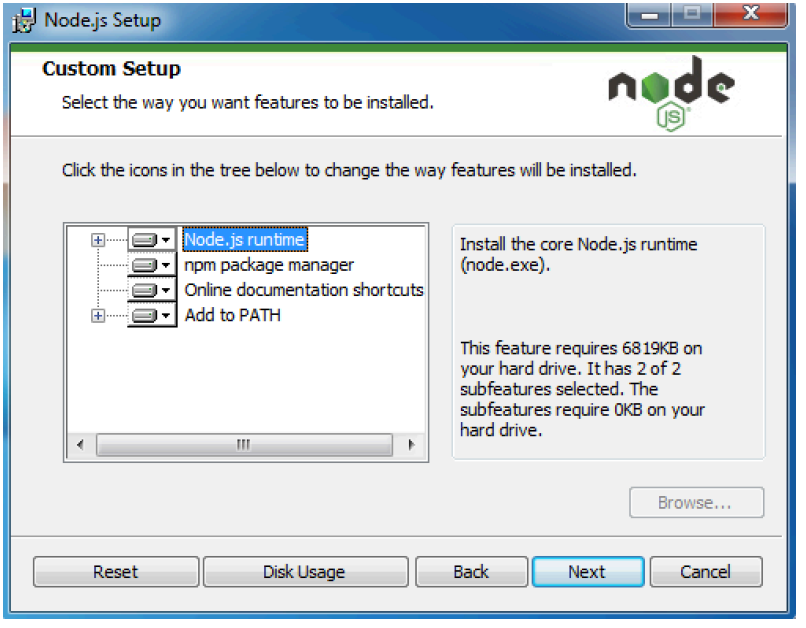
```
require "selenium-webdriver"

driver = Selenium::WebDriver.for :firefox
driver.navigate.to "http://www.google.com"

element = driver.find_element(:name, 'q')
element.send_keys "Hello Selenium WebDriver!"
element.submit

puts driver.title
```

1.2 Install Node.JS

Node.js[2] is a JavaScript runtime built on Chrome's V8 JavaScript engine. I use the latest Node.js stable version 10 for the recipes in this book.

[2] https://nodejs.org





Verify **node** and **npm** (Node.js' package manager) are installed:

```
> node --version
v15.9.0
> npm --version
7.5.4
```

1.3 Write first script with JavaScript editor

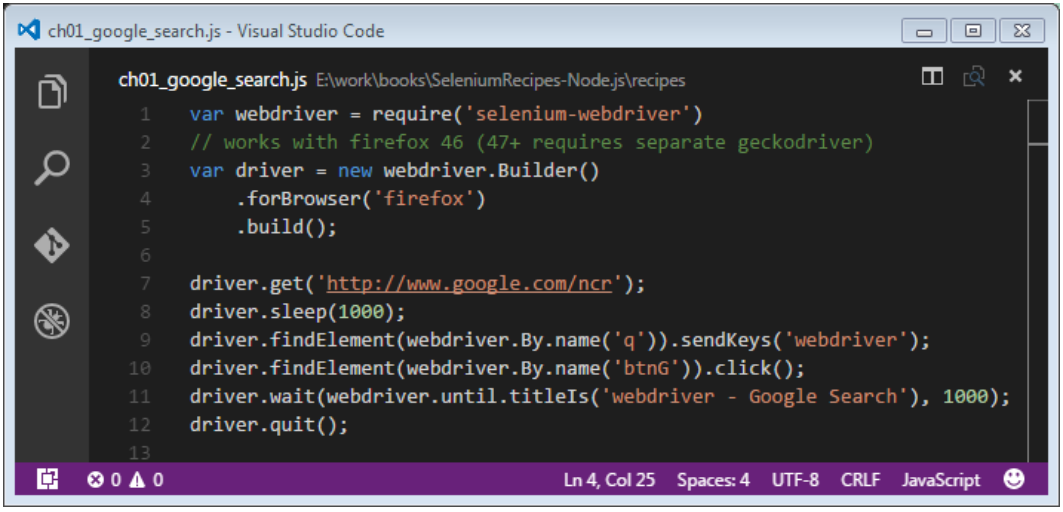
Many programmer's editors (such as Sublime Text) and IDEs (such as WebStorm) support JavaScript, some even provide specific support for Node.js. In this book, I will use Visual Studio Code[3], a free source code editor developed by Microsoft (as a matter of fact, Visual Studio Code is developed with Node.js). Visual Studio Code runs on Windows, Linux and OS X.

Now let's write one Selenium WebDriver script in Node.js. Get your JavaScript editor or IDE ready, type or paste in the script below:

```
var webdriver = require('selenium-webdriver')
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();

driver.get('http://www.google.com/ncr');
driver.sleep(1000);
driver.findElement(webdriver.By.name('q')).sendKeys('webdriver');
driver.findElement(webdriver.By.name('btnG')).click();
driver.wait(webdriver.until.titleIs('webdriver - Google Search'), 1000);
driver.quit();
```

Save the file as *ch01_google_search.js*. It will look like this in Visual Studio Code:

[3] https://code.visualstudio.com



1.4 Install Selenium WebDriver

By default, Node.js scripts load dependent (npm) packages installed locally, i.e, the packages are installed under the directory where the scripts located.

```
> cd YOUR_SCRIPT_DIRECTORY
> npm install selenium-webdriver@4.25.0
```



Previous stable Selenium 3.6.0. You can install it with the command `npm install selenium-webdriver@3.6.0`.

This will create the **node_modules** directory in your current directory, and the packages will be installed into that directory.

If you run into issues on npm package installation, refer NPM doc[4].

1.5 Run script

Prerequisite:

[4] https://docs.npmjs.com/getting-started/installing-npm-packages-locally

- Chrome browser with ChromeDriver installed (see below)

Start a new command (or terminal) window, change to the the script directory, run the command:

```
> node ch01_google_search.js
```

You shall see a new Chrome window open, perform a Google search of 'webdriver' and close the browser.

1.6 Cross browser testing

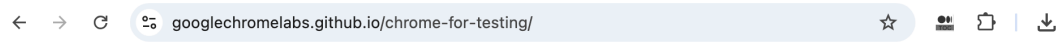
The biggest advantage of Selenium over other web test frameworks, in my opinion, is that it supports all major web browsers: Firefox, Chrome and Internet Explorer/Edge. The browser market nowadays is more diversified (based on the StatsCounter[5], the usage share in May 2021 for Chrome, Firefox, Safari, IE/Edge and are 68.8%, 7.2%, 9.7% and 9.9% respectively). It is logical that all external facing websites require serious cross-browser testing. Selenium is a natural choice for this purpose, as it far exceeds other commercial tools and free test frameworks.

Chrome

To run Selenium tests in Google Chrome, you'll need both the Chrome browser and **ChromeDriver** installed. As of Selenium v4.12 and Chrome v155, the new Selenium Manager handles ChromeDriver installation automatically. However, it's still helpful to know how to install ChromeDriver manually, just in case.

Installing ChromeDriver is easy: go to ChromeDriver site[6].

[5] http://en.wikipedia.org/wiki/Usage_share_of_web_browsers
[6] https://googlechromelabs.github.io/chrome-for-testing/



chromedriver	**linux64**	https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.69/linux64/chromedriver-linux64.zip
chromedriver	**mac-arm64**	https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.69/mac-arm64/chromedriver-mac-arm64.zip
chromedriver	**mac-x64**	https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.69/mac-x64/chromedriver-mac-x64.zip
chromedriver	**win32**	https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.69/win32/chromedriver-win32.zip
chromedriver	**win64**	https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.69/win64/chromedriver-win64.zip

Download the one matching your browser and target platform, unzip it and put **chromedriver** executable in your PATH. To verify the installation, open a command window (terminal for Unix/Mac), execute command *chromedriver*, You should see output similar to the following:

```
Starting ChromeDriver 130.0.6723.69 (...) on port 0
Only local connections are allowed.
ChromeDriver was started successfully on port 64654.
```

The test script below opens a website in a new Chrome browser window and closes it one second later.

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();
```

Firefox

Selenium tests requires Gecko Driver[7] to drive Firefox. The test script below will open a web site in a new Firefox window.

[7] https://github.com/mozilla/geckodriver/releases/

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
    .forBrowser('firefox')
    .build();
```

Edge Chromium

Edge (Chromium) is Microsoft's new and default web browser that works on all common platforms, including macOS. An additional library 'edge-selenium-tools' (besides 'selenium-webdriver') is required.

```
npm install @microsoft/edge-selenium-tools
```

The driver for Edge Chromium is 'Microsoft Edge WebDriver', which you can download at https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/[8].

```
var webdriver = require('selenium-webdriver');
var edge = require('selenium-webdriver/edge');

let options = new edge.Options();
options.setEdgeChromium(true);
let driver = edge.Driver.createSession(options);
driver.get("https://whenwise.agileway.net");
```

Edge Chromium is very similar to Chrome, as they are both based on the Chromium. Unless there is a specific requirement for Edge, I would suggest using Google Chrome for running selenium tests for stability (*ChromeDriver has been around since the inital release of Selenium WebDriver*).

Edge Legacy

To drive Edge (Legacy) with WebDriver, you need download Microsoft WebDriver[9]. After installation, you will find the executable (*MicrosoftWebDriver.exe*) under *Program Files* folder, add it to your PATH.

However, I couldn't get it working after installing a new version of Microsoft WebDriver. One workaround is to specify the driver path in test scripts specifically:

[8] https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
[9] https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/

```
// copy MicrosoftWebDriver.exe to the test script directory
var webdriver = require('selenium-webdriver');
var edge = require('selenium-webdriver/edge');
var service = new edge.ServiceBuilder(__dirname + '/MicrosoftWebDriver.exe')
    .build();
var options = new edge.Options();
var driver = new edge.Driver(options, service);
driver.get("http://www.google.com/ncr");
```

1.7 Mocha Test Framework

The above scripts drive browsers, strictly speaking, they are not tests. To make the effective use of Selenium scripts for testing, we need to put them in a test syntax framework that defines test structures and provides assertions (performing checks in test scripts). There are several popular JavaScript test frameworks, such as Mocha[10], Karma[11], and Jasmine[12].

I use Mocha in this book. You are free to choose any JavaScript test framework, Selenium WebDriver techniques are applicable regardless of syntax frameworks. Here is a Mocha test script for user login:

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();
var assert = require('assert');

describe('User Authentication', async function () {

  it('User can sign in', async function () {
    await driver.get('http://travel.agileway.net');
    await driver.findElement(webdriver.By.name('username')).sendKeys('agileway');
    await driver.findElement(webdriver.By.name('password')).sendKeys('testwise');
    await driver.findElement(webdriver.By.name('commit')).click();
    await driver.getTitle().then(function(the_title) {
      assert.equal("Agile Travel", the_title);
    });
  });

});
```

[10] https://mochajs.org/
[11] https://github.com/karma-runner/karma
[12] http://jasmine.github.io/

The keywords `describe` and `it` define the structure of a Mocha test script.



I/O in JavaScript are asynchronous, which means that a callback will be needed to notify the program when the I/O operation is done. In my opinion, this makes JavaScript not a suitable language for scripting automated tests. `aysnc/await`, in short, handles the asynchronous stuff. For being concise, I omit `aysnc/await` in the book *(the book recipe script files are complete with them)*.

In Selenium v3, I could use `test.describe` and `test.it`, provided with `selenium-webdriver/testing` module, which transparently waits for the promise to resolve before resuming the function. However, this module is no longer available for Selenium v4.

- **`describe`**

  Description of a collection of related test cases.
- **`it`**

  Individual test case.

I use Node.js' assert module[13] here to perform checks: `assert.equal(...)`.

Mocha hooks

If you worked with xUnit before, you must have known `setUp()` and `tearDown()` fixtures that are run before or after every test. Mocha provides similar hooks `before()`, `after()`, `beforeEach()`, and `afterEach()`, which can be used to set up preconditions and clean up after test scripts.

[13] https://nodejs.org/api/assert.html

```
describe('User Authentication', function () {

  before(function() {
    // run before all test cases, commonly initalize WebDriver
    driver = new webdriver.Builder()
          .forBrowser('chrome')
          .build();
  });

  beforeEach(function() {
    // run before each test case, typically visit home page
    driver.get("http://travel.agilway.net")
  });

  afterEach(function() {
    // run after each test case
  });

  after(function() {
    // run after all test cases, typicaly close browser
    driver.quit();
  });

  it('Test case description', function() {
   // one test case

  });

  it('Another Test case description', function() {
   // another test case

  });

});
```

Install Mocha

Use **npm** to install Mocha package.

```
> npm install -g mocha
```

-g tells npm to install this module globally, i.e, any Node.js project can use this module. When it is done, the output will be like below:

```
+ mocha@9.1.3
```

Run the command below in a new Command window (or Terminal on Mac/Linux) to verify that Mocha is installed.

```
> mocha --version
9.1.3
```

1.8 Create package.json file

A **package.json** file contains metadata about your Node.js app (in our case, test project), usually in the project root. It includes the list of module dependencies to install from npm. While it is not mandatory, it is a good idea to have it so that it will be easy to install or update the project's dependent modules (just run `npm install`) . If you are familiar with Ruby, it is similar to a Gemfile.

To create *package.json*, run the command below in the test project's root folder.

```
> npm init
```

Follow the prompt and answer the questions, the answers will be saved in *package.json* file in the folder. Then install two modules: *selenium-webdriver* and *mocha*. You might have noticed the difference from the previous `npm install` commands: using `--save` instead of `-g`. This tell npm to install modules locally and update the *package.json* file.

```
> npm install --save selenium-webdriver@4.1.0
> npm install --save mocha
```

After installation, you will see folder **node_modules** (locally installed modules) and the following content in the *package.json* file.

```
"dependencies": {
  "mocha": "^9.1.3",
  "selenium-webdriver": "^4.1.0",
}
```

1.9 Run recipe test scripts

Test scripts for all recipes can be downloaded from this book's site. They are all in a ready-to-run state. I include the target web pages/sites as well as Selenium test scripts. There are two kinds of target web pages: local HTML files and web pages on a live site. An Internet connection is required to run tests written for a live site.

One key advantage of open-source test frameworks, such as Selenium WebDriver and Mocha, is FREEDOM. You can edit the test scripts in any text editors and run them from a command line. For example, To run test cases in a test script file (named *ch01_agiletravel_login_spec.js*), run command

```
> mocha ch01_agiletravel_login_spec.js
```

You might get the error output like below:

Timeout error

```
User Authentication
  1) User can sign in

0 passing (2s)
1 failing

1) User Authentication User can sign in:
   Error: timeout of 2000ms exceeded. Ensure the done() callback is being called
   in this test.
     at Timeout.<anonymous> (C:\...\npm\node_modules\mocha\lib\runnable.js:226:19)
```

This is because the default timeout for a test case (or hook) is 2000 ms (i.e. 2 seconds). One common way is to override the timeout value at the beginning of a test case:

```
it('User can sign in', async function() {
  this.timeout(8000);
  await driver.get('http://travel.agileway.net');
  // ...
});
```

Run the test script again, it shall pass. Here is a sample output with two test cases in one test script file.

Sample Success Output

```
User Authentication
  ✓ Invalid user (784ms)
  ✓ User can login successfully (1197ms)

2 passing (11s)
```

2. Locating web elements

As you might have already figured out, to drive an element in a page, we need to find it first. Selenium WebDriver uses what is called locators to find and match the elements on web page. There are 8 locators in Selenium WebDriver:

Locator	Example
ID	`findElement(By.id("user"))`
Name	`findElement(By.name("username"))`
Link Text	`findElement(By.linkText("Login"))`
Partial Link Text	`findElement(By.partialLinkText("Next"))`
XPath	`findElement(By.xpath("//div[@id='login']/input"))`
Tag Name	`findElement(By.tagName("body"))`
Class Name	`findElement(By.className("table"))`
CSS	`findElement(By.css, "#login > input[type='text']"))`

You may use any one of them to narrow down the element you are looking for.

2.1 Start browser

Testing web sites starts with a browser. The test script below launches a Chrome browser window and navigate to a site.

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
               .forBrowser('chrome')
               .build();
driver.get("https://agileway.com.au/demo")
```

Use `firefox` and `ie` for testing in Firefox and IE respectively.

Test Pages

I prepared the test pages for the recipes, you can download it at the book's site[a]. Unzip it to a local directory and refer to test pages like this:

```
driver.get("file://" + __dirname + "/../../site/locators.html");
```

`__dirname` is the directory where the test script is.

[a] http://zhimin.com/books/selenium-recipes-nodejs

I recommend, for beginners, to close the browser window at the end of a test case.

```
driver.quit();
```

2.2 Find element by ID

Using IDs is the easiest and probably the safest way to locate an element in HTML. If a web page is W3C HTML conformed[1], the IDs should be unique and identified in web controls on the page. In comparison to texts, test scripts that use IDs are less prone to application changes (e.g. developers may decide to change the label, but are less likely to change the ID).

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
              .forBrowser('chrome')
              .build();
// ...
driver.findElement(webdriver.By.id("submit_btn")).click();
```

As we will use locator statement `webdriver.By` frequently in test scripts, we usually create a shorthand `By` for it as below.

[1] http://www.w3.org/TR/WCAG20-TECHS/H93.html

```
var webdriver = require('selenium-webdriver'),
              By = webdriver.By,
              until = webdriver.until;
var driver = new webdriver.Builder()
              .forBrowser('chrome')
              .build();
//...
driver.findElement(By.id("cancel_link")).click();            // Link
driver.findElement(By.id("username")).sendKeys("agileway");  // Textfield
driver.findElement(By.id("alert_div")).getText();     // HTML Div element
```

2.3 Find element by Name

The name attributes are used in form controls such as text fields and radio buttons. The values of the name attributes are passed to the server when a form is submitted. In terms of least likelihood of a change, the name attribute is probably only second to ID.

```
driver.findElement(By.name("comment")).sendKeys("Selenium Cool");
```

2.4 Find element by Link Text

For Hyperlinks only. Using a link's text is probably the most direct way to click a link, as it is what we see on the page.

```
driver.findElement(By.linkText("Cancel")).click();
```

2.5 Find element by Partial Link Text

Selenium allows you to identify a hyperlink control with a partial text. This can be quite useful when the text is dynamically generated. In other words, the text on one web page might be different on your next visit. We might be able to use the common text shared by these dynamically generated link texts to identify them.

```
// will click the "Cancel" link
driver.findElement(By.partialLinkText("ance")).click();
```

2.6 Find element by XPath

XPath, the XML Path Language, is a query language for selecting nodes from an XML document. When a browser renders a web page, it parses it into a DOM tree or similar. XPath can be used to refer a certain node in the DOM tree. If this sounds too technical for you, don't worry, just remember XPath is the most powerful way to find a specific web control.

```
// clicking the checkbox under 'div2' container
driver.findElement(By.xpath("//*[@id='div2']/input[@type='checkbox']")).click();
```

Some testers feel intimidated by the complexity of XPath. However, in practice, there is only limited scope of XPath to master for testers.



Avoid using copied XPath from Browser's Developer Tool

Browser's Developer Tool (right click to select 'Inspect element' to show) is very useful for identifying a web element in web page. You may get the XPath of a web element there, as shown below (in Chrome):



The copied XPath for the second "Click here" link in the example:

```
//*[@id="container"]/div[3]/div[2]/a
```

It works. However, I do not recommend this approach as the test script is fragile. If developer adds another `div` under `<div id='container'>`, the copied XPath is no longer correct for the element while `//div[contains(text(), "Second")]/a[text()="Click here"]` still works.

In summary, XPath is a very powerful way to locating web elements when `id`, `name` or `linkText` are not applicable. Try to use a XPath expression that is less vulnerable to structure changes around the web element.

2.7 Find element by Tag Name

There are a limited set of tag names in HTML. In other words, many elements share the same tag names on a web page. We normally don't use the `tagName` locator by itself to locate an element. We often use it with others in a chained locators (see the section below). However, there is an exception.

```
driver.findElement(By.tagName("body")).getText();
```

The above test statement returns the text view of a web page, this is a very useful one as Selenium WebDriver does not have built-in method return the text of a web page. However, unique this JavaScript binding, we cannot use this way directly. For example, the script below prints out the body's text.

```
var pageText = driver.findElement(By.tagName("body")).getText();
console.log(pageText);
```

The output is a `ManagedPromise`.

```
ManagedPromise {
  flow_:
   ControlFlow {
     propagateUnhandledRejections_: true,
  ...
}
```

> ### What is a Promise?
>
> A Promise is "an object that represents a value, or the eventual computation of a value". Essentially, a promise is a result of async operation that is not resolved yet. If you are not familiar with JavaScript, it can be quite confusing. Good news is that, for the context of Selenium WebDriver, you will soon find out the pattern when there is need to resolve a promise.

To resolve a promise, use `.then(function(returnedValue){ ... }`.

```
driver.findElement(By.tagName("body")).getText().then(function(body_text){
  console.log(body_text)
});
```

2.8 Find element by Class

The `class` attribute of a HTML element is used for styling. It can also be used for identifying elements. Commonly, a HTML element's class attribute has multiple values, like below.

```
<a href="back.html" class="btn btn-default">Cancel</a>
<input type="submit" class="btn btn-deault btn-primary">Submit</input>
```

You may use any one of them.

```
driver.findElement(By.className("btn-primary")).click(); // Submit button
driver.findElement(By.className("btn")).click();         // Cancel link
```

The `className` locator is convenient for testing JavaScript/CSS libraries (such as TinyMCE) which typically use a set of defined class names.

```
// inline editing
driver.findElement(By.id("client_notes")).click();
driver.sleep(500);
driver.findElement(By.className("editable-textarea")).sendKeys("inline");
driver.sleep(500);
driver.findElement(By.className("editable-submit")).click();
```

2.9 Find element by CSS

You may also use CSS Path to locate a web element.

```
driver.findElement(By.css("#div2 > input[type='checkbox']")).click();
```

However, the use of CSS is generally more prone to structure changes of a web page.

2.10 Chain findElement to find child elements

For a page containing more than one elements with the same attributes, like the one below, we could use `XPath` locator.

```
<div id="div1">
  <input type="checkbox" name="same" value="on"> Same checkbox in Div 1
</div>
<div id="div2">
  <input type="checkbox" name="same" value="on"> Same checkbox in Div 2
</div>
```

There is another way: chain `findElement` to find a child element.

```
driver.findElement(By.id("div2")).findElement(By.name("same")).click();
```

2.11 Use locator name as JSON attribute

You may use locator as JSON attribute as below.

```
driver.findElement({name: 'comment'}).sendKeys("JSON")   // a text box
driver.findElement({css: "#div1 > input[type='checkbox']"}).click();
driver.findElement({className: "btn-primary"}).click();  // Submit button
```

2.12 Find multiple elements

As its name suggests, `findElements` return a list of matched elements back. Its syntax is exactly the same as `findElement`, i.e. can use any of 8 locators.

The test statements will find two checkboxes under `div#container` and click the second one.

```
let xpathStr = "//div[@id='container']//input[@type='checkbox']";
var checkboxElems = driver.findElements(By.xpath(xpathStr))
driver.sleep(500)
// console.log("XXX: " + checkboxElems.then.length + "\n"); // 2
driver.findElements(By.xpath(xpathStr)).then(function(checkboxElems) {
  checkboxElems[1].click();  // second one
});
```

Sometimes `findElement` fails due to multiple matching elements on a page, which you were not aware of. `findElements` will come in handy to find them out.

3. Hyperlink

Hyperlinks (or links) are fundamental elements of web pages. As a matter of fact, it is hyperlinks that makes the World Wide Web possible. A sample link is provided below, along with the HTML source.

Recommend Selenium

HTML Source

```
<a href="index.html" id="recommend_selenium_link" class="nav" data-id="123"
   style="font-size: 14px;">Recommend Selenium</a>
```

3.1 Click a link by text

Using text is probably the most direct way to click a link in Selenium, as it is what we see on the page.

```
// driver.get("file://" + __dirname + "/../../site/link.html");
driver.findElement(By.linkText("Recommend Selenium")).click();
```

3.2 Click a link by ID

```
driver.findElement(By.id("recommend_selenium_link")).click();
```

Furthermore, if you are testing a web site with multiple languages, using IDs is probably the only feasible option. You do not want to write test scripts like below:

```
if (isItalian()) {
  driver.findElement(By.linkText("Accedi")).click();
} else if (isChinese()) { // a helper function determines the locale
  driver.findElement(By.linkText, "□□").click();
} else {
  driver.findElement(By.linkText("Sign in")).click();
}
```

3.3 Click a link by partial text

```
driver.findElement(By.partialLinkText("Recommend Seleni")).click();
```

3.4 Click a link by XPath

The example below is finding a link with text 'Recommend Selenium' under a <p> tag.

```
driver.findElement(By.xpath( "//p/a[text()='Recommend Selenium']")).click();
```

Your might say the example before (find by `linkText`) is simpler and more intuitive, that's correct. but let's examine another example:

First div Click here
Second div Click here

On this page, there are two 'Click here' links.

HTML Source

```
<div>
  First div
  <a href="link-url.html">Click here</a>
</div>
<div>
  Second div
  <a href="link-partial.html">Click here</a>
</div>
```

If test case requires you to click the second 'Click here' link, `findElement(By.linkText("Click here"))` won't work (as it clicks the first one). Here is a way to accomplish using XPath:

```
driver.findElement(By.xpath('//div[contains(text(), "Second")]/a[text()="Click here"]')).\
click();
```

3.5 Click Nth link with exact same label

It is not uncommon that there are more than one link with exactly the same text. By default, Selenium will choose the first one. What if you want to click the second or Nth one?

The web page below contains two 'Click here" links,



To click the second one,

```
driver.findElements(By.linkText("Click here")).then(function(the_same_links){
  assert.equal(2, the_same_links.length);
  the_same_links[1].click(); // second link
});
```

`findElements` return a list (also called array) of web controls matching the criteria in appearing order. Selenium (in fact JavaScript) uses 0-based indexing, i.e., the first one is 0.

3.6 Click Nth link by CSS Selector

You may also use CSS selector to locate a web element.

```
driver.findElement(By.css("p > a:nth-child(3)")).click(); // 3rd link
```

However, generally speaking, the stylesheet are more prone to changes.

3.7 Verify a link present or not?

```
assert(driver.findElement(By.linkText("Recommend Selenium")).isDisplayed())
assert(driver.findElement(By.id("recommend_selenium_link")).isDisplayed())
```

3.8 Getting link data attributes

Once a web control is identified, we can get its other attributes of the element. This is generally applicable to most of the controls.

```
driver.findElement(By.linkText("Recommend Selenium")).getAttribute("href").then(
  function(the_href) {
    assert(the_href.contains("/site/index.html"))
});
driver.findElement(By.linkText("Recommend Selenium")).getAttribute("id").then(function(th\
e_id) {
  assert.equal("recommend_selenium_link", the_id)
});
driver.findElement(By.id("recommend_selenium_link")).getText().then(
  function(elemtext){
    assert.equal("Recommend Selenium", elemtext)
});
driver.findElement(By.id("recommend_selenium_link")).getTagName().then(
  function(tagname) {
    assert.equal("a", tagname)
});
```

Also you can get the value of custom attributes of this element and its inline CSS style.

```
driver.findElement(By.id("recommend_selenium_link")).getAttribute("style").then(
  function(the_style){
    assert.equal("font-size: 14px;", the_style)
});

driver.findElement(By.id("recommend_selenium_link")).getAttribute("data-id").then(
  function(the_data_id){
    assert.equal("123", the_data_id)
});
```

3.9 Test links open a new browser window

Clicking the link below will open the linked URL in a new browser window or tab.

```
<a href="https://agileway.com.au/demo" target="_blank">Open new window</a>
```

While we could use `switchTo()` method (see chapter 9) to find the new browser window, it will be easier to perform all testing within one browser window. Here is how:

```
var currentUrl = driver.getCurrentUrl();
new_window_url = driver.findElement(By.linkText("Open new window")).getAttribute("href")
driver.get(new_window_url)
// ... testing on new site
driver.findElement(By.name("name")).sendKeys("sometext")
driver.get(currentUrl) // back
```

In this test script, we use a local variable ‘currentUrl’ to store the current URL.

4. Resources

Recipe test scripts
http://zhimin.com/books/bought-selenium-recipes-nodejs[1]

Username: `agileway`
Password: `LOADWISE10`

Log in with the above, or scan QR Code to access directly.

4.1 Books

- **Practical Web Test Automation**[2] by Zhimin Zhan

  Solving individual selenium challenges (what this book is for) is far from achieving test automation success. *Practical Web Test Automation* is the book to guide you to the test automation success, topics include:

  - Page object model
  - Functional Testing Refactorings
  - Cross-browser testing against IE, Firefox and Chrome
  - Strategies on team collaboration and test automation adoption in projects and organizations

- **Practical Continuous Testing**[3] by Zhimin Zhan

  Continuous Testing (CT) is the key process of DevOps. This books guides you to manage executing your Selenium tests in a CT server.

- **Selenium WebDriver Recipes in Java**[4] by Zhimin Zhan

  Sometimes you might be required to write Selenium WebDriver tests in Java. Master Selenium WebDriver in Java quickly by leveraging this book.

[1] http://zhimin.com/books/bought-selenium-recipes-nodejs
[2] https://leanpub.com/practical-web-test-automation
[3] https://leanpub.com/practical-continuous-testing
[4] https://leanpub.com/selenium-recipes-in-java

- **Selenium WebDriver Recipes in C#, 2nd Edition**[5] by Zhimin Zhan

  Selenium WebDriver recipe tests in C#, another popular language that is quite similar to Java.
- **Selenium WebDriver Recipes in Ruby**[6] by Zhimin Zhan

  Selenium WebDriver tests can also be written in Ruby, a beautiful dynamic language very suitable for scripting tests. Master Selenium WebDriver in Ruby quickly by leveraging this book.
- **Selenium WebDriver Recipes in Python**[7] by Zhimin Zhan

  Selenium WebDriver recipes in Python, a popular script language that is similar to Ruby.
- **API Testing Recipes in Ruby**[8] by Zhimin Zhan

  The problem solving guide to testing APIs such as SOAP and REST web services in Ruby language.

4.2 Web Sites

- **Selenium JavaScript API** https://www.selenium.dev/selenium/docs/api/javascript/[9]
- **Selenium JavaScript Github**[10]
- **Selenium Home**[11]

4.3 Blog

- **TestWisely Blog** (https://zhiminzhan.medium.com[12])

  I share my experience and views on Test Automation and Continuous Testing there.

[5] http://www.apress.com/9781484217412
[6] https://leanpub.com/selenium-recipes-in-ruby
[7] https://leanpub.com/selenium-recipes-in-python
[8] https://leanpub.com/api-testing-recipes-in-ruby
[9] https://www.selenium.dev/selenium/docs/api/javascript/
[10] https://github.com/SeleniumHQ/selenium/tree/master/javascript/node/selenium-webdriver
[11] https://www.selenium.dev
[12] https://zhiminzhan.medium.com

4.4 Tools

- **Visual Studio Code**[13]

  Free and flexible code editor from Microsoft.
- **WebStorm IDE**[14]

  Commerical JavaScript IDE from JetBrains.
- **BuildWise** (http://testwisely.com/buildwise[15])

  AgileWay's free and open-source continuous testing server, purposely designed for running automated UI tests with quick feedback.

[13] https://code.visualstudio.com/
[14] https://www.jetbrains.com/webstorm/
[15] http://testwisely.com/buildwise