3. Selenium Webdriver

Selenium Webdriver

WebDriver is a tool for writing automated tests of websites. It aims to mimic the behavior of a real user, and as such interacts with the HTML of the application. WebDriver is part of Selenium and it's main contribution is its API and the native drivers.

Selenium RC and WebDriver

Selenium RC

Selenium Remote Control (RC) is a test tool that allows you to write automated web application UI tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser.

Webdriver vs RC

The primary new feature of Selenium 2 is the integration of WebDriver, a rival web application testing framework to Selenium 1 (a.k.a. Selenium RC).

While Selenium RC runs a JavaScript application within the browser, WebDriver controls the browser directly using native browser support or browser extensions.
WebDriver addresses the shortcomings of Selenium 1, mainly the ability to step outside the JavaScript sandbox and the ability to provide a fast, lightweight, headless browser emulator. Simon Stewart, the creator of WebDriver, explained why the projects were merging in a joint email to the WebDriver and Selenium community back in August 2009.

```
Why are the projects merging? Partly because WebDriver addresses some
shortcomings in Selenium (by being able to bypass the JS sandbox, for
example. And we've got a gorgeous API),
partly because Selenium addresses some shortcomings in WebDriver (such as
supporting a broader range of browsers) and partly because the main
Selenium contributors and I
felt that it was the best way to offer users the best possible framework.
```

WebDriver is developed for each browser and replaces the JavaScript that was embedded on web applications being tested. This tighter integration with the browser allows for the creation of more advanced tests, and removes the restrictions placed by the JavaScript security model. Besides support from browser vendors, WebDriver also provides emulation of user inputs using OS level calls. WebDriver is supported on Firefox (FirefoxDriver), IE (InternetExplorerDriver), Opera (OperaDriver) and Chrome (ChromeDriver). Support for Safari is not included in this release due to technical constraints, but can be emulated using the SeleneseCommandExecutor. It also works on Android (AndroidDriver) and iPhone (IPhoneDriver) for mobile web application testing. It also has a headless implementation based on HtmlUnit called HtmlUnitDriver. The WebDriver APIs can be accessed from Python, Ruby, Java and C#,
allowing developers to create tests using their preferred programming language.

Main differences:

Selenium RC	Webdriver
Works on almost all browsers.Does not work on latest version of firefox/IE	Works on latest versions of almost all browsers - Firefox, IE(6,7,8), OPera, Chrome
Server is required to start	No server required to start
Core engine is Javascript based	Interacts natively with browser application
Its a simple and small API	Complex and a bit large API as compared to RC
Less Object orinted API	Purely Object oriented API

Cannot move mouse with it	Can move mouse cursor
Full xpaths have to be appended with 'xapth=\\' syntax	No need to append 'xpath=\\'
No Listeners	Implementaiton of Listeners is provided
Cannot test iphone/Android applications	Can test iphone/Android applications
predefined wait	implicit and explicit wait

Webdriver's drivers

HTMLUnit driver

Currently the fastest implementation of WebDriver. It is based on HTMLUnit, a java based implementation of a WebBrowser without a GUI. HTMLUnit acts a 'headless' browser - a browser without a renderer - which lets you test the functionality of a website without necessarily testing the visual aspects.
One of the main advantages are:

- fastest Webdriver implementation
- platform independent
- supports JS
- support for HTTP and HTTPS protocols
- support for cookies
- support for submit methods POST and GET (as well as HEAD, DELETE ...)
- support for HTML responses (submitting forms, clicking links)

This driver is the most used driver when we deal with the automated tests integrated in a CI environment. This is because it's platform independent and is a browser emulator. Also, all those advantages take to one disadvantage that is caused by the fact that it is not a real browser, it'a a browser emulator that emulates other browser's JavaScript behavior.

Using HTMLUnit

```
WebDriver driver = new HtmlUnitDriver();
```

Enabling Javascript

```
HtmlUnitDriver driver = new HtmlUnitDriver();
driver.setJavascriptEnabled(true);
```

Enabling a specific browser

```
HtmlUnitDriver driver = new HtmlUnitDriver(BrowserVersion.FIREFOX_3);
```

Firefox driver

Controls the Firefox browser using a Firefox plugin. The Firefox Profile that is used is stripped down from what is installed on the machine to only

include the Selenium WebDriver.xpi (plugin). Firefox Driver is capable of being run and is tested on Windows, Mac, Linux.
One of the main advantages of Firefox driver is that runs as a real browser and supports JavaScript. It appears to be the second driver from the speed point of view (slower than HTMLUnit and faster than IEDriver).

Using Firefox driver

```
WebDriver driver = new FirefoxDriver()
```

;

Running with firebug
Download the firebug xpi file from mozilla and start the profile as follows:

```
File file = new File("firebug-1.8.1.xpi");
FirefoxProfile firefoxProfile = new FirefoxProfile();
firefoxProfile.addExtension(file);
firefoxProfile.setPreference("extensions.firebug.currentVersion", "1.8.1"); //
Avoid startup screen
WebDriver driver = new FirefoxDriver(firefoxProfile);
```

Beta - load fast preferences
There is beta feature to make firefox not wait for the full page to load after calling .get or .click. This may cause immediate find's to break, so please be sure to use an implicit or explicit wait too. This is only available for Firefox and not other browsers.

```
FirefoxProfile fp = new FirefoxProfile();
fp.setPreference("webdriver.load.strategy", "unstable"); // As of 2.19. from 2.9 -2.18
use 'fast'
WebDriver driver = new FirefoxDriver(fp);
```

IE driver

This driver is controlled by a .dll and is thus only available on Windows OS.
One of the main advantages of Firefox driver is that runs as a real browser and supports JavaScript.
Unfortunately, this driver presents a set of disadvantages:

- works only on windows
- a little bit slow
- XPath is not natively supported in most versions
- CSS is not natively supported in version 6 and 7
- problems occurs at: browser focus, hover over elements, submitting forms and alert()

Using IE driver

```
WebDriver driver = new InternetExplorerDriver();
```

Chrome driver

Chrome Driver is maintained / supported by the Chromium project iteslf. WebDriver works with Chrome through the chromedriver binary (found on

the chromium project's download page). You need to have both chromedriver and a version of chrome browser installed. chromedriver needs to be placed somewhere on your system's path in order for WebDriver to automatically discover it. The Chrome browser itself is discovered by chromedriver in the default installation path. These both can be overridden by environment variables. Please refer to the wiki for more information.

The main advantages:

- Runs in a real browser and supports JavaScript
- Because Chrome is a Webkit-based browser, the Chrome Driver may allow you to verify that your site works in Safari. Note that since Chrome uses its own V8 JavaScript engine rather than Safari's Nitro engine, JavaScript execution may differ.

It still needs to have some speed improvements in the future.

Chrome driver usage

```
WebDriver driver = new ChromeDriver();
```

iPhone driver

The iphone driver allows testing on a UIWebView (a webkit browser accessible for 3rd party applications) on the iphone. It works through the use of an iphone application running on your iphone, ipod touch or iphone simulator.

IPhone driver usage

```
WebDriver driver = new RemoteWebDriver(new URL("http://localhost:3001/wd/hub"),
DesiredCapabilities.iphone());
// or use the convenience class which uses localhost:3001 by default
WebDriver driver = new IPhoneDriver();
```

Android driver

Android WebDriver allows to run automated end-to-end tests that ensure your site works correctly when viewed from the Android browser. Android WebDriver supports all core WebDriver APIs, and in addition to that it supports mobile specific and HTML5 APIs. Android WebDriver models many user interactions such as finger taps, flicks, finger scrolls and long presses. It can rotate the display and interact with HTML5 features such as local storage, session storage and application cache.

Android driver usage

```
WebDriver driver = new AndroidDriver();
```

RemoteWebdriver

Remote WebDriver controls a browser by sending commands to a remote server. This server is expected to be running the WebDriver wire protocol .
The remote webdriver comes in two flavours:

- Client mode: where the language bindings connect to the remote instance. This is the way that the FirefoxDriver, OperaDriver and the RemoteWebDriver client normally work.
- Server mode: where the language bindings are responsible for setting up the server, which the driver running in the browser can connect to. The ChromeDriver works in this way.

Advantages

- Separates where tests run from where the browser is
- Allows tests to use browsers not available on the current OS

Disadvantages

- Requires an external server
- Introoduces extra latency to tests

Usage

```
// 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");
One nice feature of the remote webdriver is that exceptions often have an attached
screen shot, encoded as a
Base64 PNG. In order to get this screenshot, you need to write code similar to:
public String extractScreenShot(WebDriverException e) {
Throwable cause = e.getCause();
if (cause instanceof ScreenshotException) {
return ((ScreenshotException) cause).getBase64EncodedScreenshot();
}
return null;
}
```