selenium-jupiter

A JUnit 5 extension for Selenium WebDriver

Boni García

Version 1.0.0-SNAPSHOT

Table of Contents

Quick reference 1
Motivation 3
Support 4
About 5

JUnit 5 is the next generation of the well-known testing framework JUnit. *Jupiter* is the name given to the new programming and extension model provided by JUnit 5. Regarding the extension model of JUnit 5, it allows to incorporate extra capabilities for Jupiter tests. On the other hand, Selenium WebDriver is another testing framework which allows to control browsers (e.g. Chrome, Firefox, and so on) programmatically to carry out automated testing of web applications. This documentation presents ***selenium-jupiter***, a JUnit 5 extension aimed to provide seamless integration of Selenium WebDriver within Jupiter tests. It is open source (Apache 2.0 license) and its source code is hosted on GitHub.

Quick reference

selenium-jupiter has been built using the dependency injection capability provided by the extension model of JUnit 5. Thank to this feature, parameters can be injected in JUnit 5 in `@Test` methods. Concretely, *selenium-jupiter* allows to inject subtypes of the **WebDriver** interface (e.g. *ChromeDriver*, *FirefoxDriver*, etc.) as parameters.

Using *selenium-jupiter* it's easy as pie. First, you need to import the dependency in your project (typically as *test* dependency). In Maven, it is done as follows:

```
<dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>selenium-jupiter</artifactId>
    <version>1.0.0</version>
    <scope>test</scope>
</dependency>
```

Then, you need to declare *selenium-jupiter* extension in your JUnit 5 test, simply annotating your test with `@ExtendWith(SeleniumExtension.class)`. Finally, you need to include one or more parameters in your `@Test` methods whose types implements the `WebDriver` interface. That's it. *selenium-jupiter* control the lifecycle of the `WebDriver` object internally, and you just need to use the `WebDriver` object in your test case to automatically control the browser(s) you want. For example:

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ChromeAndFirefoxJupiterTest {

    @Test
    public void testWithOneChrome(ChromeDriver chrome) {
        // using Chrome in this test
    }

    @Test
    public void testWithFirefox(FirefoxDriver firefox) {
        // using Firefox in this test
    }

    @Test
    public void testWithChromeAndFirefox(ChromeDriver chrome,
            FirefoxDriver firefox) {
        // using Chrome and Firefox in this test
    }

}
```

The `WebDriver` subtypes supported by *selenium-jupiter* are the following:

- `ChromeDriver`: Used to control Google Chrome browser.
- `FirefoxDriver`: Used to control Firefox browser.
- `EdgeDriver`: Used to control Microsoft Edge browser.
- `OperaDriver`: Used to control Opera browser.
- `SafariDriver`: Used to control Apple Safari browser (only possible in OSX El Capitan or greater).
- `HtmlUnitDriver`: Used to control HtmlUnit (headless browser).
- `PhantomJSDriver`: Used to control PhantomJS (headless browser).
- `InternetExplorerDriver`: Used to control Microsoft Internet Explorer. Although this browser is supported, Internet Explorer is deprecated (in favor of Edge) and its use is highly discouraged.
- `RemoteDriver`: Used to control remote browsers (*Selenium Grid*).

WARNING	The browser to be used must be installed in the machine running the test beforehand (except in the case of `RemoteWebDriver`, in which the requirement is to known a Selenium Server URL).

Motivation

Selenium WebDriver allows to control different types of browsers (such as Chrome, Firefox, Edge, and so on) programmatically using different programming languages. This is great to implement automated tests for web applications. Nevertheless, in order to use WebDriver, we need to pay a prize. For security reasons, the automated manipulation of a browser can only be done using native features of the browser. In practical terms, it means that a binary file must be placed in between the test using the WebDriver API and the actual browser. The general schema should is depicted as follows:

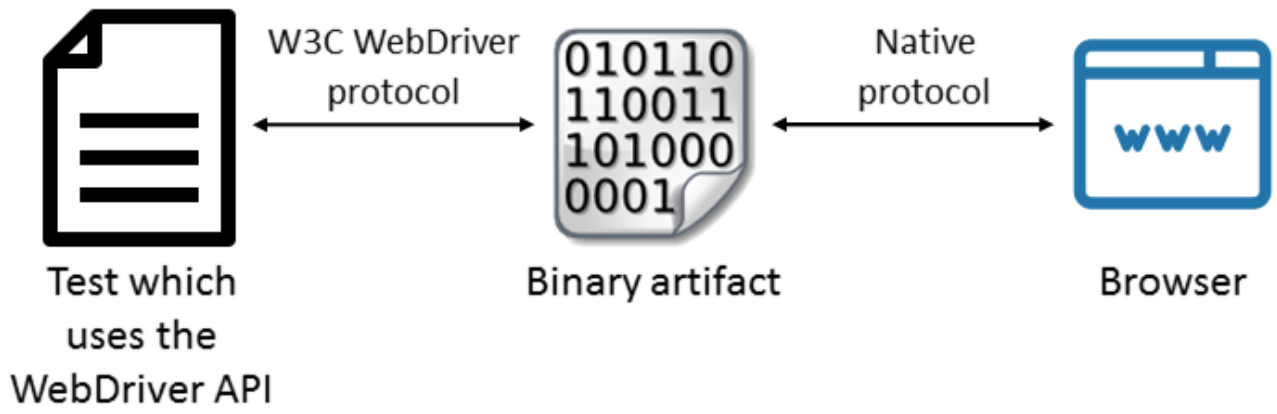

Figure 1. WebDriver general scenario

From a tester point of view, this is a cumbersome process, since it is required to download the proper binary for the platform running the test (i.e. Windows, Linux, Mac). Moreover, the versioning is also a headache, since browsers evolve quite fast, and the corresponding binary file required by WebDriver needs to be constantly updated:

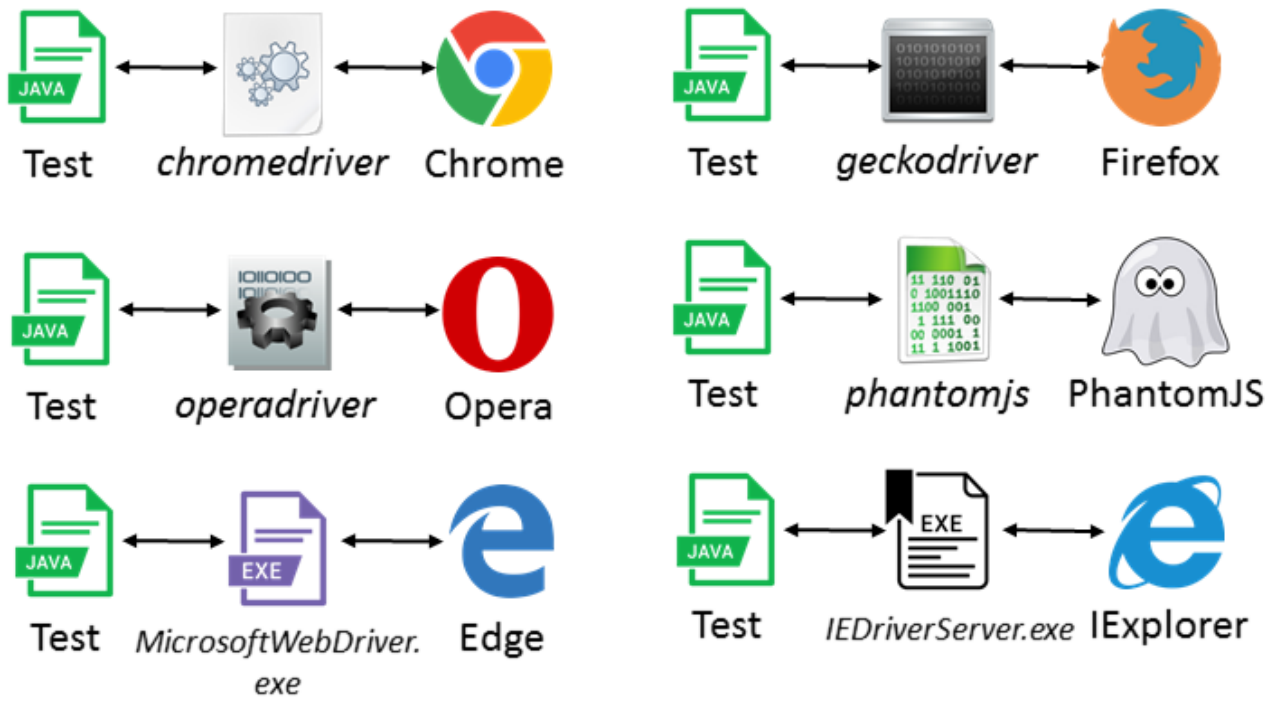

Figure 2. WebDriver scenario for Chrome, Firefox, Opera, PhantomJS, Edge, and Internet Explorer

Focusing on Java, before creating a WebDriver instance, the absolute path of the binary controlling the browser should be exported in a given environment variable, as follows:

```
System.setProperty("webdriver.chrome.driver", "/absolute/path/to/binary/chromedriver"
);
System.setProperty("webdriver.opera.driver", "/absolute/path/to/binary/operadriver");
System.setProperty("webdriver.ie.driver",
"C:/absolute/path/to/binary/IEDriverServer.exe");
System.setProperty("webdriver.edge.driver",
"C:/absolute/path/to/binary/MicrosoftWebDriver.exe");
System.setProperty("phantomjs.binary.path", "/absolute/path/to/binary/phantomjs");
System.setProperty("webdriver.gecko.driver", "/absolute/path/to/binary/geckodriver");
```

In order to simplify the life of Java WebDriver users, in March 2015 the utility WebDriverManager was first released. WebDriverManager is a library which automates all this process (download the proper binary and export the proper variable) for Java in runtime. The WebDriverManager API is quite simple, providing a singleton object for each of the above mentioned browsers:

```
ChromeDriverManager.getInstance().setup();
FirefoxDriverManager.getInstance().setup();
OperaDriverManager.getInstance().setup();
PhantomJsDriverManager.getInstance().setup();
EdgeDriverManager.getInstance().setup();
InternetExplorerDriverManager.getInstance().setup();
```

The solution implemented by WebDriverManager was latter ported in equivalent tools for other languages, such as webdriver-manager for **Node.js** or WebDriverManager.net for **.NET**.

On August 2017, a new major version of the well-know testing JUnit framework was released. This leads to ***selenium-jupiter***, which can be seen as the natural evolution of *WebDriverManager* for **JUnit 5** tests. Internally, *selenium-jupiter* is built using two foundations:

1. It uses *WebDriverManager* to manage the binaries requires by WebDriver.
2. It uses the *dependency injection* feature of the extension model of JUnit 5 to inject WebDriver objects within `@Test` methods.

Support

There are several ways to get in touch with *selenium-jupiter*:

- Questions about *selenium-jupiter* are supposed to be discussed in StackOverflow, using the tag *selenium-jupiter*.
- Comments, suggestions and bug-reporting should be done using the GitHub issues.
- If you think *selenium-jupiter* can be enhanced, consider contribute to the project by means of a pull request.

About

selenium-jupiter (Copyright © 2017) is a project created by Boni Garcia (@boni_gg) licensed under Apache 2.0 License. This documentation is released under the terms of CC BY 3.0.