WebDriverManager

Table of Contents

1. Motivation

Selenium WebDriver is a library that allows controlling web browsers programmatically. It provides a cross-browser API that can be used to drive web browsers (e.g., Chrome, Edge, or Firefox, among others) using different programming languages (e.g., Java, JavaScript, Python, C#, or Ruby). The primary use of Selenium WebDriver is implementing automated tests for web applications.

Selenium WebDriver carries out the automation using the native support of each browser. For this reason, we need to place a binary file called **driver** between the test using the Selenium WebDriver API and the browser to be controlled. Examples of drivers for major web browsers nowadays are chromedriver (for Chrome), geckodriver (for Firefox), or msedgedriver (for Edge).

From a practical point of view, we need to carry out a *driver management process* to use Selenium WebDriver. This process consists on:

1. Download. Drivers are platform-specific binary files. Therefore, we need to identify the driver type we need (e.g., chromedriver if we want to use Chrome), the operating system (typically,

Windows, Linux, or Mac OS), the architecture (typically, 32 or 64 bits), and very important, the version. Each driver release is usually compatible with a given browser version(s). For this reason, we need to find out the correct driver version for a specific browser version.

2. Setup. Once we have downloaded the driver to our computer, we need to provide a way to locate this driver from our Selenium WebDriver tests. In Java, this setup can be done in two different ways. First, we can add the driver location to our PATH environmental variable. Second, we can use *Java system properties* to export the driver path. Each driver path should be identified using a given system property, as follows:

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

3. Maintenance. Last but not least, we need to warranty the compatibility between driver and browser in time. This step is relevant since modern browsers automatically upgrade themselves (i.e., they are *evergreen* browsers), and for this reason, the compatibility driver-browser is not warranted in the long run. For instance, when a WebDriver test using Chrome faces a driver incompatibility, it reports the following error message: "*this version of chromedriver only supports chrome version N*." As you can see in StackOverflow, this is a recurrent problem for manually managed drivers (chromedriver in this case).

WebDriverManager is an open-source Java library that carries out this driver management process in a fully automated manner. In addition, as of version 5, WebDriverManager provides other relevant features, such as the ability to find if a browser is installed in the system, build WebDriver objects (such as `ChromeDriver`, `FirefoDriver`, etc.), and the capability to run browsers in Docker containers containers seamlessly.

2. Setup

Using WebDriverManager is very easy. 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>webdrivermanager</artifactId>
    <version>5.0.0</version>
    <scope>test</scope>
</dependency>
```

The dependency (typically *test*) to be declared in a Gradle project is as follows:

```
dependencies {
    testImplementation("io.github.bonigarcia:webdrivermanager:5.0.0")
}
```

Then, you need to declare *Selenium-Jupiter* extension in your JUnit 5 test, simply annotating your test with `@ExtendWith(SeleniumJupiter.class)`. Finally, you need to include one or more parameters in your `@Test` methods (or constructor) whose types implements the `WebDriver` interface (e.g. `ChromeDriver` to use Chrome, `FirefoxDriver` for Firefox, and so for). 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 to drive the browser(s) you want. For example:

```
import static org.assertj.core.api.Assertions.assertThat;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.wdm.WebDriverManager;

class ChromeTest {

    WebDriver driver;

    @BeforeAll
    static void setupAll() {
        WebDriverManager.chromedriver().setup();
    }

    @BeforeEach
    void setupEach() {
        driver = new ChromeDriver();
    }

    @AfterEach
    void teardown() {
        if (driver != null) {
            driver.quit();
        }
    }

    @Test
    void test() {
        // Exercise
        driver.get("https://bonigarcia.org/webdrivermanager/");

        // Verify
        assertThat(driver.getTitle()).contains("WebDriverManager");
    }

}
```

As of JUnit 5.1, extensions can be registered programmatically via `@RegisterExtension` or automatically via Java's `ServiceLoader`. Both mechanisms can be used with *Selenium-Jupiter*.

3. Features

3.1. Driver Management

Under construction.

3.1.1. Resolution Algorithm

Under construction.

3.2. Browser Finder

Under construction.

3.3. WebDriver Builder

Under construction.

3.4. Browsers in Docker

Under construction.

3.4.1. Recordings

Under construction.

3.4.2. Remote Desktop

Under construction.

4. Other Usages

4.1. WDM CLI

Under construction.

4.2. WDM Server

Under construction.

4.3. WDM in Docker

Under construction.

4.4. WDM Agent

Under construction.

5. Examples

Under construction.

6. Configuration

Under construction.

7. Known issues

Under construction.

HTTP response code 403

Under construction.

Testing localhost

Under construction.

Docker and chromedriver 92+

Under construction.

8. Support

There are several ways to get in touch with WebDriverManager:

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

9. Contributing

To do.

10. Further documentation

To do.

11. About

WebDriverManager (Copyright © 2017-2021) is an open-source project created and maintained by Boni García (@boni_gg), licensed under the terms of Apache 2.0 License. This documentation (also available in PDF) is released under the terms of CC BY-NC-SA 2.0.