Rex Jones II

How Selenium Architecture Works For Automation

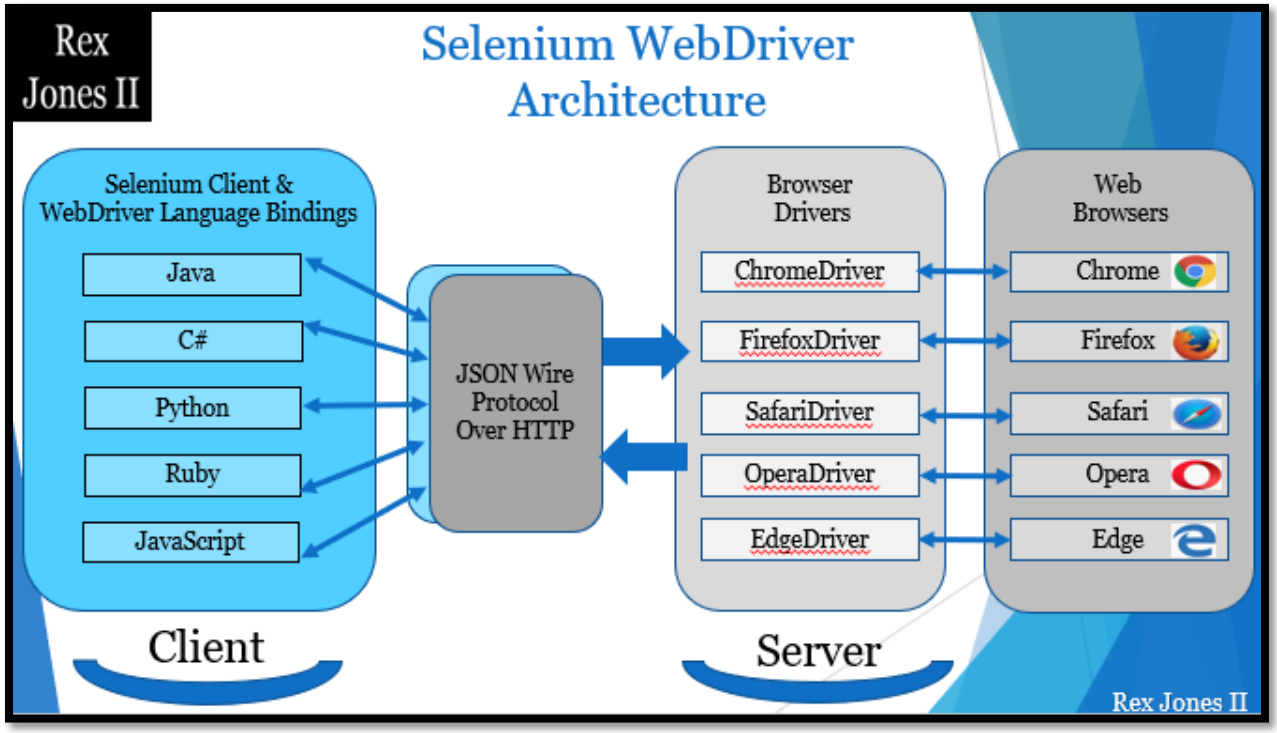

Table of Contents

Rex Jones II

Video Link

https://www.youtube.com/watch?v=V2RKYbVqBwo

Introduction

Hello and Welcome To Selenium 4 Beginners. We are going to discuss How The Selenium Architecture Works For Automation. The Transcript and Presentation Slides will be available on github at RexJonesII/Selenium4Beginners and https://tinyurl.com/SeleniumArchitecture .

In this lesson, I will show you a diagram of the Selenium Architecture, demo the Selenium Architecture Components, then Recap the Selenium Architecture.

The Architecture of Selenium WebDriver is mainly about communication between the client and server. There are 4 components that make up the architecture. The 1st component is Selenium Client and WebDriver Language Bindings. Next, is the JSON Wire Protocol. Third, we have the Browser Drivers, and fourth are the Web Browsers.

Selenium Architecture Diagram

Selenium Client & WebDriver Language Bindings

The 1st component has 2 parts. Selenium Client is 1 part and WebDriver Language Bindings is another part. Selenium Client is responsible for sending out a request to perform a command.

The WebDriver Language Bindings is a code library designed to drive actions on a web browser. A good way to think about a code library is to imagine about a library with a lot of books. Rather than having books, a code library has a lot of code for Java, C#, Ruby, Python, and JavaScript. There are more programming languages that can be used for Selenium but these are the 5 core languages. Each language has their own bindings. Bindings mean the same commands written for Java is also written for C#, Ruby, Python, and JavaScript. For example, Java has a command for navigating to a page and the other languages have a command for navigating to a page.

Let's take a look at Selenium's HQ website. We see the Selenium WebDriver API Commands and Operations. API stands for Application Programming Interface which allow communication between applications. All of these languages have a command for navigating to a page but Java, Ruby, Python, Perl, and JavaScript use get as their method. Here's the method for C#.

We can download the Selenium Client and Bindings from their website. After downloading the APIs for Selenium, we can add them to our IDE such as Eclipse, NetBeans, IntelliJ, or Visual Studio. If we scroll down to the Third-Party Drivers section, we see our next component which is Browser Drivers. GeckoDriver, ChromeDriver, Opera, EdgeDriver, and SafariDriver are some of the drivers.

Browser Drivers

In this diagram, the Browser Drivers have 2 functions. The first function is to receive a request and the second function is to return a response.

Both of these components illustrate a client-server model. What is a client-server model? A client-server model is when the client makes a request and the server executes that request. After executing that request, the server sends back a response to complete their connection. Their connection is built around a

JSON Wire Protocol

JSON Wire Protocol Over HTTP. JSON stands for JavaScript Object Notation and HTTP stands for Hyper Text Transfer Protocol. The objective of a JSON Wire Protocol is to transfer information from the client to the server. That information is processed over HTTP by sending HTTP Requests and receiving HTTP Responses.

Let's take a look at JSON Wire Protocol on github. The Introduction states "This wire protocol defines a RESTful web service". REST is an acronym for REpresentational State Transfer and it is used create APIs for a web application. Recall Selenium is an API. Therefore, the JSON Wire Protocol has an API call for every Selenium command. If we go to the Command Reference, there is a column for HTTP Method, Path, and Summary. This Command Summary list 3 of 5 HTTP Methods: GET, POST, and DELETE but PUT and PATCH are not here. We see a Path and Summary for Timeouts. Implicit Wait, Get Window Handle, navigate to a new URL, get the current page title.

We see a GET HTTP Method request and a session API call. sessionId is the URL parameter. It's an ID of the session to route the command to. The page title is returned as a string.

Let's go back to our diagram and review the flow. A Selenium request is sent from the Selenium Client and WebDriver Language Bindings component. That request is sent to the JSON Wire Protocol which defines a REST API. The REST API is sent to the Browser Drivers in the form of a URL. ChromeDriver, FirefoxDriver, SafariDriver, OperaDriver, and EdgeDriver all have their own HTTP Server.

Web Browsers

Finally, we have our last component which is the Web Browsers. All Selenium commands are performed on the Web Browser. Notice, there is a 2 directional arrow between the Brower Drivers and Web Browsers. This is one of the reasons why Selenium executes our Test Script so fast. The Browser Drivers receive a request and immediately the Web Browser executes that request. Whether our Test Script Pass or Fail, the Web Browser returns a response back to the Browser Driver. The Browser Driver sends that response back to the JSON Wire Protocol and eventually to the client. If the action Fails then an exception shows up in our IDE.

Demo Selenium Architecture Components

I am going to use Java as the programming language and Eclipse as the IDE. We start by downloading Selenium from Selenium HQ website to my Download folder, then add Selenium to Eclipse Libraries tab as External JARs. JAR stands for Java ARchive which allows us to write our Selenium Commands using

Java. WebDriver driver = new ChromeDriver (); ChromeDriver is our Browser Driver. next is System.setProperty to set the property as ChromeDriver. The key is webdriver.chrome.driver and value is the path of chromedriver.exe from my Drivers folder.

Now, we load the JSON Wire Protocol page using driver.get then get the title using driver.getTitle. Know what, let's also print the title. The page landed on click. Click on an element. Eclipse shows Json Wire Protocol as the page title and our Test Script PASSED.

Recap

Let's recap the Selenium Architecture using our Test Script. With the 1st component, we wrote our Selenium Commands using Java. A request was sent to set up ChromeDriver, load the web page, and print the page title. The JSON Wire Protocol received the request then changed that request to a format so ChromeDriver can understand the command. ChromeDriver sent a direct HTTP Request to the Chrome Web Browser. The Chrome Browser performed each command then sent back an HTTP response to the ChromerDriver that eventually showed up in Eclipse. I hope that helps and Thank You for watching How The Selenium Architecture Works For Automation.