Rex Jones II

Architecture
Selenium 3 vs Selenium 4

Table of Contents

Video Link 2
Introduction 2
Selenium Architecture Diagram 2
Selenium Client & WebDriver Language Bindings 2
Browser Drivers 3
JSON Wire Protocol 3
Web Browsers 4
Demo Selenium Architecture Components 4
Recap 4

Rex Jones II

W3C WebDriver Protocol 4
Components 5
Direct 5
Remote 7
Advantages 8
Standards 8
Stability 8
Actions API 9
Mobile Automation 9
Cloud Platforms 9

Video Link

https://youtu.be/IPEtkfQryr8

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.

W3C WebDriver Protocol

The W3C WebDriver Protocol has at least 3 advantages. #1 It provides standards #2 It provides stability and #3 It provides an updated Actions API that is supplied with better resources. I will talk about all 3 and get straight to the point.

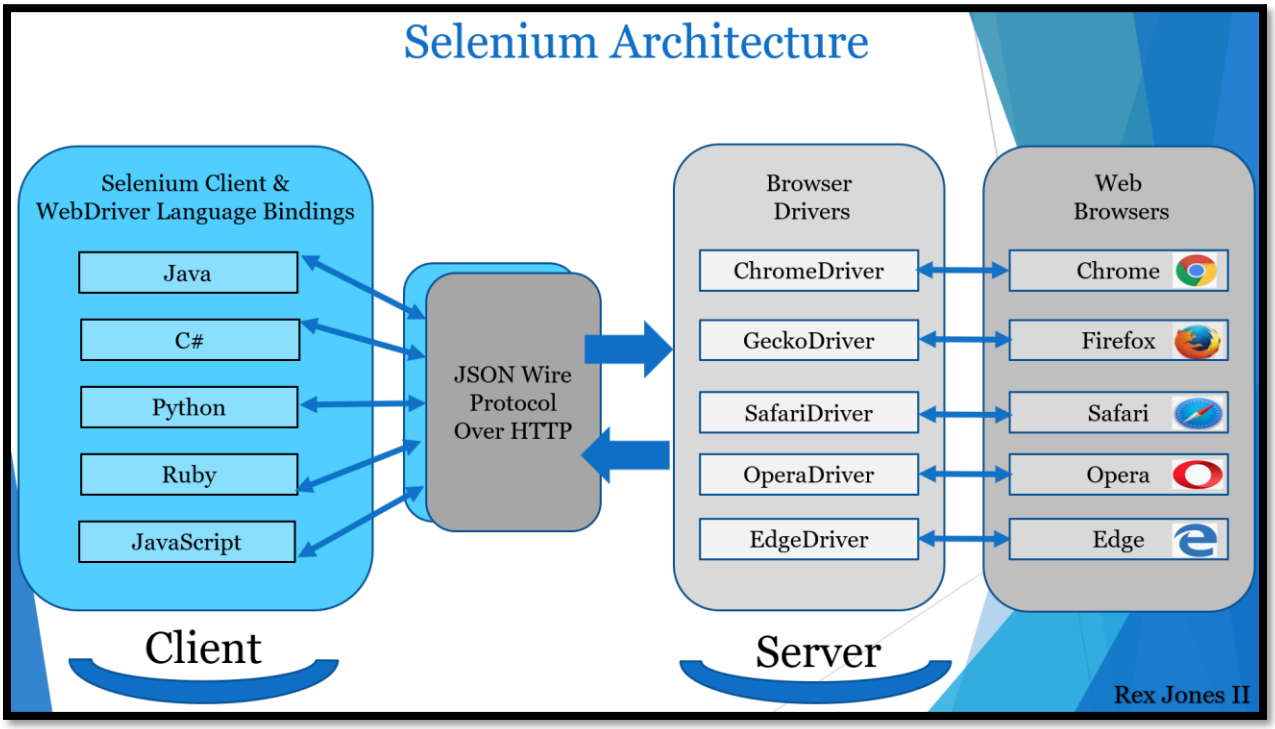
If you are interested in more videos, you can subscribe to my YouTube channel and click the bell icon. You can also follow me on Twitter, connect with me on LinkedIn and Facebook. I will also place the transcript and presentation slides on GitHub.

In this session, I am going to speak about the Selenium 4 Components, Advantages, Mobile Automation, and Cloud Platforms.

Rex Jones II

Components

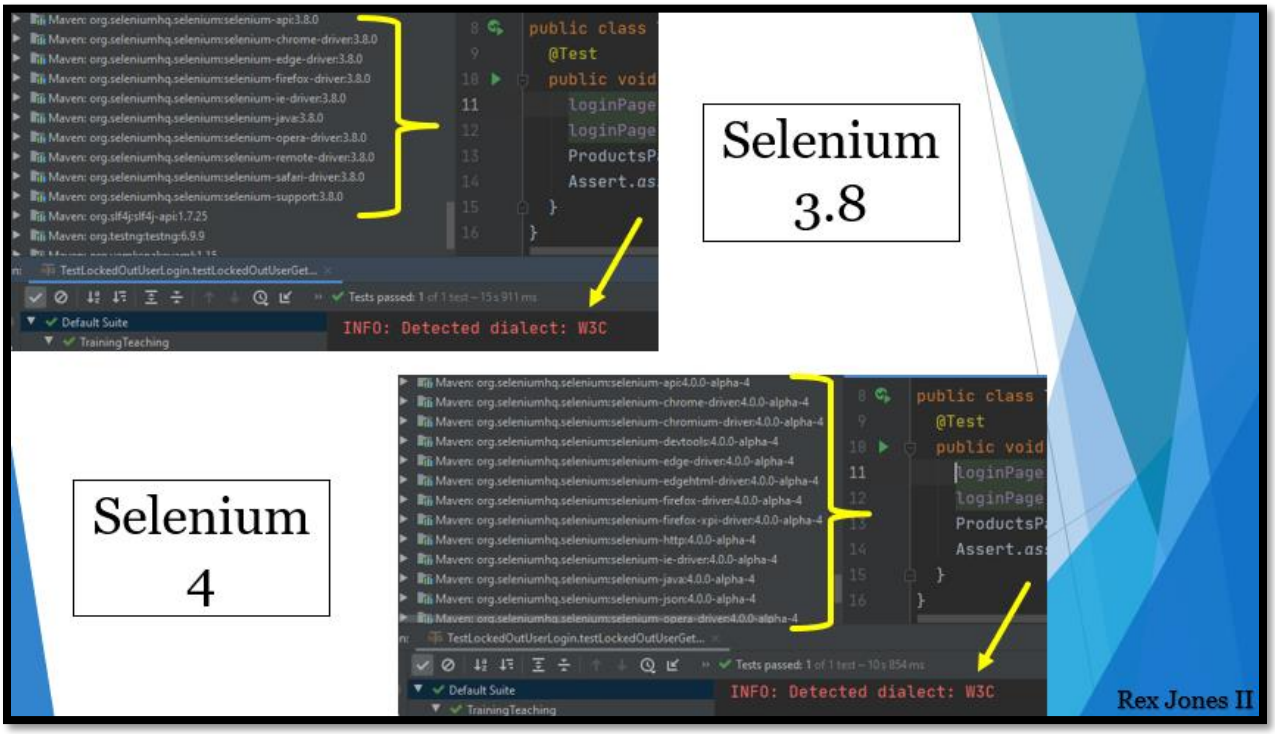
Starting with the components, Selenium has moved from 3 to 4 because of the W3C WebDriver Protocol. This is an example of Selenium 3 which includes the JSON Wire Protocol. The objective of JSON Wire Protocol was to transfer information from the client to the server. That information was processed over HTTP by sending HTTP Requests and receiving HTTP Responses.



Direct

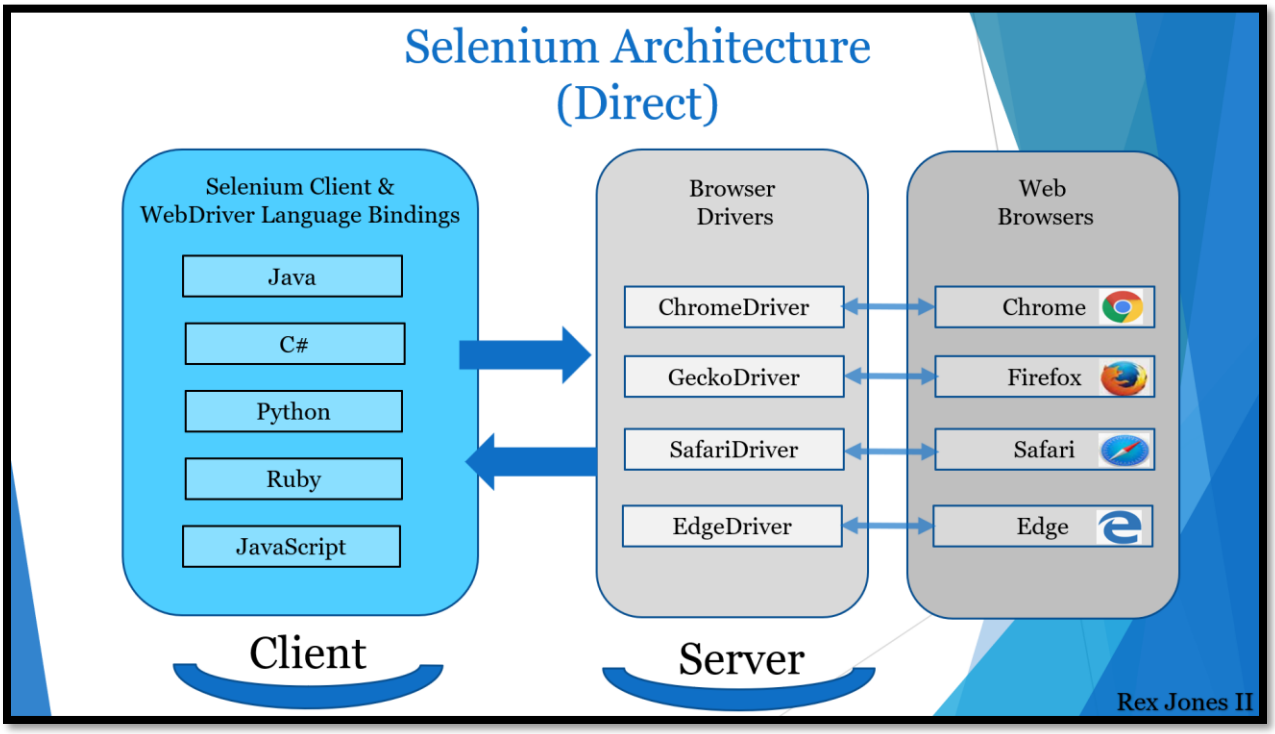
With Selenium 4, the JSON Wire Protocol has been removed from the new architecture. Now, there is direct communication between the Browser Drivers and Selenium Client & WebDriver Language Bindings. The 1st component has 2 parts combined into one. Selenium Client is a separate part and WebDriver Language Bindings is a different part. Selenium is an API that have commands for automating our browser. WebDriver has only 1 job and that job is talk to the browser through a driver.

Each language has their own bindings. Bindings mean the same commands written for Java is also written for C#, Python, Ruby, and JavaScript. You may have noticed that Selenium added support for the W3C protocol starting at version 3.8. According to Simon Stewart, in this Selenium 4 Webinar with BrowserStack, he mentioned the versions of Selenium since 3.8 have spoken to both JSON Wire Protocol and W3C Protocol. After running your Test Script, look for INFO: Detected dialect: W3C to see if your Selenium version is speaking to W3C. Both of these screenshots show W3C for Selenium 3.8 and Selenium 4. If not W3C then it will show OSS which means Open Source Software.



Back to our diagram, we see the 2nd component is Browser Drivers and it have 2 functions. The first function is to receive a request from Selenium Client & WebDriver Language Bindings then pass that request to the browser. A driver also known as a proxy is responsible for controlling the browser. The second function is to return a response from the browser back to the Selenium Client & WebDriver Language Bindings. All of the drivers use the W3C WebDriver Protocol and most of them are created by the browser vendors.

When it comes to the 3rd component Web Browsers. This is where all of the Selenium Commands are performed. The browser receives a request, performs the request, and sends back a response to the driver. Also, notice how Opera is not available as a driver or browser. In reality, they are still available but I did not add them because the WebDriver implementations are no longer under development so native support has been removed for Opera and Phantom JS.

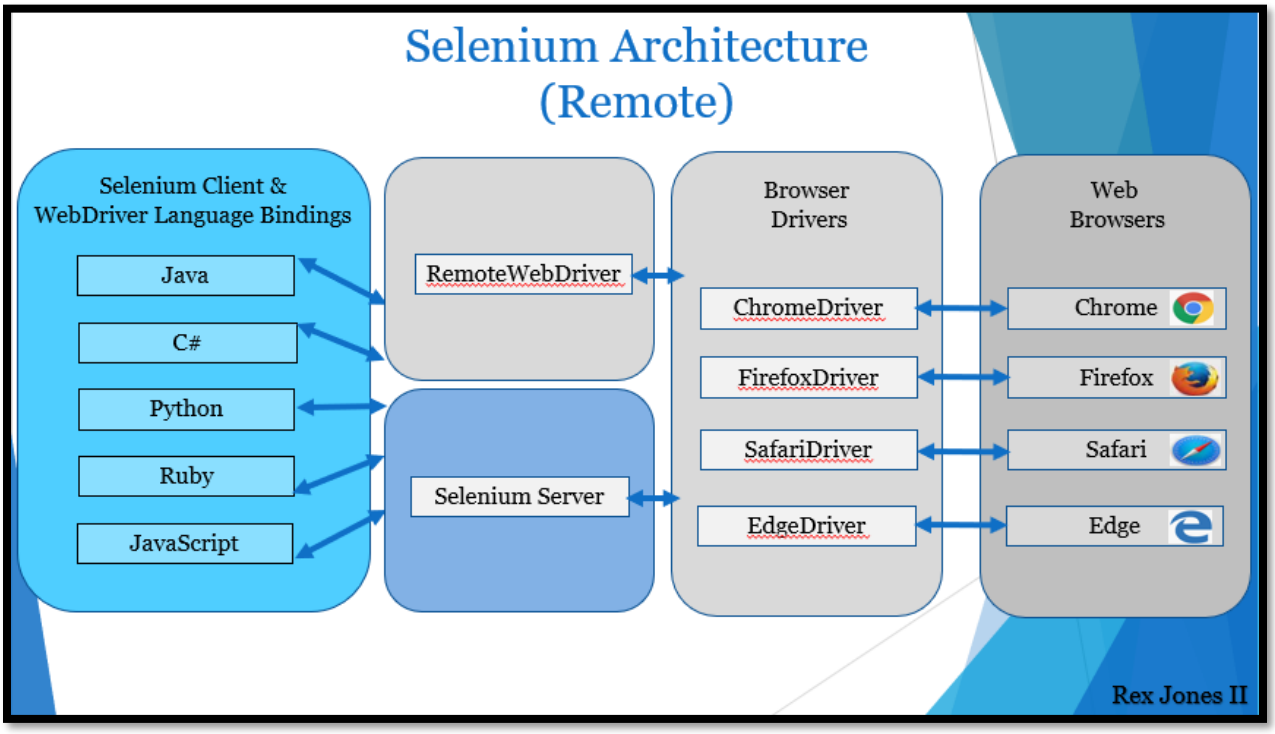


Remote

Remote is another form of communication to the browser. It can happen through the RemoteWebDriver or Selenium Server. The RemoteWebDriver is a class that implements the WebDriver Interface. With Selenium 4, FirefoxDriver and SafariDriver continue to extend RemoteWebDriver. However, ChromeDriver and EdgeDriver no longer extend RemoteWebDriver but they extend ChromiumDriver. In the introduction, I showed how ChromiumDriver, FirefoxDriver, InternetExplorerDriver, OperaDriver, and SafariDriver all extend RemoteWebDriver. We see 3 of the boxes (RemoteWebDriver, Browser Drivers, and Web Browsers) are gray because they all run on the same system.

The Selenium Server is different. It's a way to communicate remotely when talking to the driver but not on the same system as the driver. That's why Selenium Server is a different color. We start the server by using the Selenium Standalone jar file. After it starts, the server directs our Test Scripts to a remote web browser.

Rex Jones II



Advantages

Standards

For W3C WebDriver Protocol, the advantages are Standards, Stability, and Actions. W3C stands for World Wide Web Consortium which is an international group of people that create long term standards for the web. With that 1st advantage of standards, our Test Scripts run consistently on each browser. There were times with Selenium 3 that some commands performed offbeat on different browsers. Since Selenium 4 is compliant with W3C WebDriver, there is no more required encoding and decoding of the API request.

Stability

The second advantage is stability. I believe backward compatibility is the main benefit of stability. Per Simon Stewart, they are fully aware that some people will want to use the old JSON Wire Protocol. So, the Java Bindings and the Selenium Server will provide mechanisms for people to use the old JSON Wire Protocol. They know, we have spent time, we have spent effort, and we have been dedicated to building up our Test Suite. Therefore, our Test Suite will remain smooth for Selenium 4. No changes to our Test Scripts unless an API has been marked deprecated. Deprecated API's like the FindsBy Interfaces have been removed from Selenium 4. As a result, the WebDriver API's are going to continue working like there was never a change to Selenium.

Rex Jones II

Actions API

The updated Actions API is the third advantage. With this API, we can handle keyboard and mouse events like double clicking an element. It's an advantage because Selenium 4 offers a way to perform more than 1 action at the same time like pressing 2 keys. That's an advantage for UI Automation.

Mobile Automation

For Mobile Automation, Appium is the tool used for automating mobile applications. We see on Appium.io site, the introduction shows native, mobile web, and hybrid application on iOS mobile, Android mobile, and Windows desktop platforms. The Appium client library have implemented elements of the W3C Protocol.

Cloud Platforms

Also, Cloud Platforms such as SauceLabs and BrowserStack support W3C WebDriver. However, the format of their capabilities will change. You can go to their site to see what capabilities need to be updated. Here's the page for BrowserStack with an Introduction section, talk about Why changes are being made, What does it mean to me, Updating the Selenium tests, and an Example section that show how capabilities are passed with W3C Protocol. SauceLabs have a page that shows W3C Capabilities Support with sections: What You Will Need, Verifying the Capabilities, W3C WebDriver-Compliant, Instantiating WebDriver with W3C, and Common Test Script Configuration Errors To Avoid. That's it for the W3C WebDriver Protocol. Don't forget to Connect and Subscribe. Next, I will demo the Relative Locators which locate elements based on their relationship to another element.

Contact

- ✔ YouTube https://www.youtube.com/c/RexJonesII/videos
- ✔ Facebook http://facebook.com/JonesRexII
- ✔ Twitter https://twitter.com/RexJonesII
- ✔ GitHub https://github.com/RexJonesII/Free-Videos
- ✔ LinkedIn https://www.linkedin.com/in/rexjones34/