**Why and When JavascriptExecutor is used in Selenium**

* **The Problem**

Sometimes, in Web UI Automation, we find that our regular Selenium commands fail to interact with the web elements. One such scenario is given below.

|  |
| --- |
| 1. driver.findElement(By.id("click-button")).click(); 2. String result = driver.findElement(By.id("result-box")).getText(); |

In line 1, the script is supposed to find the web element and click it.

If it passes, then the script executes line 2 where it will get the resultant text and store it in the String variable result. All seems to be pretty basic stuff!!!

However, during execution, we find that the script is able to find the button that it is supposed to click and does not throw any exception. But, without clicking the button, it proceeds with executing the next line of code which will of course lead to a test **FAILURE!!!**

In real time applications, many a times you might end up facing a scenario like this. One possible reason, an extra layer of security on the web elements to prevent hackers creating mayhem on the application. Especially, in critical sectors like finance and healthcare, where application security is a huge concern, test engineers often complain about their automation scripts not able to interact with the web elements.

* **The Solution**

Selenium comes here to the rescue, by providing us with an interface called JavascriptExecutor.

JavascriptExecutor executes a piece of pure Javascript code which helps us to interact with the HTML DOM elements. This is irrespective of the Selenium language binding.

Not only that, JavascriptExecutor also helps us in performing certain other actions like scrolling to a certain position in the web page or to a certain web element, etc.

As per the official documentation of JavascriptExecutor, the interface is being implemented by the below classes.

* ChromeDriver
* FirefoxDriver
* InternetExplorerDriver
* SafariDriver
* OperaDriver

This means that JavascriptExecutor is supported by all known browsers.

* **How to use JavascriptExecutor**
* **Click a button**

Below is a code snippet on how to click a button on a web page using JavascriptExecutor.

|  |
| --- |
| 1. WebElement button = driver.findElement(By.id("click-button")); 2. String script = "arguments[0].click();"; 3. ((JavascriptExecutor)driver).executeScript(script, button); |

The executeScript() method defined by JavascriptExecutor is used to execute the Javascript code on the web element passed into the method as an argument.

*Note: Here, arguments[0] means the first occurrence of the element on the web page.*

* **Typing text into a textbox**

Just like the above problem, where Selenium click() method fails to click a button on a web page, sometimes, the Selenium sendKeys() command might fail to enter text in a text box.

Below code demonstrates that shows how to enter text to a text box using JavascriptExecutor.

|  |
| --- |
| 1. WebElement textBox = driver.findElement(By.id("web-text-box")); 2. String script = "arguments[0].value='JavascriptExecutor Tutorial';"; 3. ((JavascriptExecutor)driver).executeScript(script, textBox); |

Here, we are passing the text ‘JavascriptExecutor Tutorial’ into the value property of the textbox. This will result in the text getting typed into the textbox.



* **Scrolling a web page**

Generally, scrolling is not needed in Selenium as it directly works on the DOM of the web page. But, at times, scrolling is needed as an element might get loaded only after the user scrolls to the bottom of the page. A very familiar example will be the News Feed page of Facebook. JavascriptExecutor is useful in such scenarios.

Below is a code snippet on how to scroll down to the bottom of the web page.

|  |
| --- |
| 1. String script = "window.scrollTo(0, document.body.scrollHeight);"; 2. ((JavascriptExecutor)driver).executeScript(script); |

Here, we pass the x and y coordinates of the location on the web page where we intend to scroll to into the scrollTo() method. scrollHeight denotes the total length of the web page.

Hence, by executing the above code, we intend to scroll to the bottom left corner of the web page.

Again, below is a code snippet that demonstrates how to scroll to a specific web element before we can interact with the same. This element can be positioned anywhere on the web page.

|  |
| --- |
| 1. WebElement webBanner = driver.findElement(By.id("web-banner")); 2. String script = "arguments[0].scrollIntoView();"; 3. ((JavascriptExecutor)driver).executeScript(script, webBanner); |