Perform Actions Using JavaScript in Python Selenium WebDriver

by Arunkumar Velusamy · Dec. 13, 18 · Web Dev Zone · Tutorial

In this tutorial, let's analyze the least used but most powerful feature of Selenium WebDriver. Yes, I am going to discuss the JavaScript executor, and show you a few different ways to execute JavaScript statements through Python Selenium WebDriver.

It may so happen that in some real-time projects, Selenium WebDriver cannot perform an action on a particular web element. For example, since WebDriver simulates end-user interaction, it is natural that it will refuse to click on an element that is not visible to the end user (sometimes it also happens even though the web element is visible on the page). There can be several other similar reasons or scenarios.

In these cases, we can rely on JavaScript to click or perform actions on that web element, and these JavaScript statements can be executed through WebDriver.

You can do everything that the WebElement interface does and more with JavaScript.

What is JavaScript?

JavaScript is a scripting language that runs on client side, i.e, on the browser, and does some magic things when you surf through web pages. For more details, search for the keyword "JavaScript" on DZone.

How Do We Use JavaScript in WebDriver?

Python Selenium WebDriver provides a built-in method:

```
driver.execute_script("some javascript code here");
```

There are two ways we can execute JavaScript within the browser.

Method 1: Executing JavaScript at Document Root Level

In this case, we capture the element that we want to work with, using JavaScript-provided methods, then declare some actions on it and execute this JavaScript using WebDriver.

Example:

```
javaScript = "document.getElementsByName('username')[0].click();"
driver.execute_script(javaScript)
```

What Are We Doing Here?

Step 1: We are inspecting and fetching the element by a property 'name' using JavaScript. (Also, 'id' and 'class' properties can be used.)

Step 2: Declare and perform a click action on an element using JavaScript.

Step 3: Call execute_script() method and pass the JavaScript that we created as a string value

Step 3: Call execute_script() method and pass the JavaScript that we created as a string value.

Notice `[0]` in the `getElementsByName('username')[0]` statement above. JavaScript functions `getElementsByName`, `getElementsByClassName`, and so on return all matching elements as an array. In our case, we need to act on the first matching element that can be accessed via `index [0]`. If you know what you are doing, i.e., if you know the index of the element you want to operate, you can directly use the index, such as `getElementsByName('username')[2]`.

However, if you are using the JavaScript function '`getElementById`', you do not need to use any indexing, as it will return only one element ('id' should be unique).

While executing, WebDriver will inject the JavaScript statement into the browser and the script will perform the job. In our example, it performs a click operation on the target element. This JavaScript has its own namespace and does not interfere with the JavaScript in the actual web page.

Method 2: Executing JavaScript at Element Level

In this case, we capture the element that we want to work with using WebDriver, then we declare some actions on it using JavaScript and execute this JavaScript using WebDriver by passing the web element as an argument to the JavaScript.

Is this confusing? Let's break it down.

For example:

```
userName = driver.find_element_by_xpath("//button[@name='username']")
driver.execute_script("arguments[0].click();", userName)
```

What Are We Doing Here?

Step 1: Inspect and capture the element using WebDriver-provided methods like '`find_element_by_xpath`':

```
userName = driver.find_element_by_xpath("//button[@name='username']")
```

Step 2: Declare and perform a click action on the element using JavaScript:

```
arguments[0].click()
```

Step 3: Call `execute_script()` method with the JavaScript statement that we created as a string value and web element captured using WebDriver as arguments:

```
driver.execute_script("arguments[0].click();", userName)
```

The above two lines of code can be shortened to the format below, where we find an element using WebDriver, declare some JavaScript functions, and execute the JavaScript using WebDriver.

```
driver.execute_script("arguments[0].click();", driver.find_element_by_xpath("//button[@name='username']"))
```

Another issue faced more frequently is the need to scroll to the bottom of the web page. You can perform this operation in a single line of code:

```
driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
```

Also, you can have more than one JavaScript action in your statement.

For example:

```
userName = driver.find_element_by_xpath("//button[@name='username']")
password = driver.find_element_by_xpath("//button[@name='password']")
driver.execute_script("arguments[0].click();arguments[1].click();", userName, password)
```

In this case, usage of the order of web elements matters. Accessing `index` with `[0]` anywhere inside a JavaScript statement will retrieve the first web element passed.

```
driver.execute_script("arguments[1].click();arguments[0].click();", userName, password)
```

How to Return Values

Another important aspect of the JavaScript executor is that it can be used to fetch values from web elements. That means the `execute_script()` method can return values.

For example:

```
print driver.execute_script('return document.getElementById("fsr").innerText')
```

Note that if you want something returned by JavaScript code, you need to use return. Also, elements can be located with Selenium and passed into the script.

What Happens When an Element Is Not Found?

When JavaScript cannot find an element to operate on, it throws a WebDriver exception with the respective error message.

Scenario 1: We're trying to read a property using `' print driver.execute_script('return document.getElementById("fsr").innerText') '` but there is no such element available in the web page. We get the following message in the exception trace:

```
selenium.common.exceptions.WebDriverException: Message: unknown error: Cannot read property 'innerText' of null
```

Scenario 2: We're trying to use an invalid operation or error function name inside JavaScript, like `' print driver.execute_script('document.getElementById("fsr").clic();') '`. (Note the spelling mistake in the `click()` method name.)

```
selenium.common.exceptions.WebDriverException: Message: unknown error: document.getElementById(...).clic is not a function
```

Summary

Here is a summary of a few potential actions for which JavaScript could be used.

- get elements text or attribute
- find an element
- do some operation on an element, like `click()`

- change attributes of an element
- scroll to an element or location on a web page
- wait until the page is loaded

Basic knowledge of JavaScript helps a lot when handling the DOM with Selenium. You can find more details and articles on the allselenium website.

Like This Article? Read More From DZone

DZone Article
Using PyUnit for Testing a Selenium Python Test Suite

DZone Article
Selenium WebDriver for Cross-Browser Testing, Part 2

DZone Article
Why Should You Switch to Cypress for Modern Web Testing?

Free DZone Refcard
Getting Started With Appium

Topics: JAVASCRIPT, PYTHON, SELENIUM 3.0, TEST AUTOMATION, TUTORIAL, WEB DEV