Selenium

Class 8

Agenda

Actions Class in Selenium

Synchronization & Waits in Selenium

Handling Keyboard & Mouse Events

Sometimes in our test automation we will have to work with submenu and submenu items render in DOM only when we mouse hover on main menu.

In that case, we face difficulty to click on sub menu item. In order to perform mouse hover actions, we need to chain all of the actions that we want to achieve in one go.

To do this we need to make the driver move to the parent element that has child elements and click on the child element.

To achieve this we use Actions class in Selenium WebDriver.

Actions class

The Actions class allows us to build a chain of actions that we would like to perform.

Operations Supported by Actions Class:

1. Mouse Actions: helps user to perform all the operations related with mouse operations like : clicking, dragging, moving, clicking and dragging, hovering, double clicking, right clicking.
2. KeyBoard Actions: helps user to emulate keyboard operations on keys **CTRL, ALT, Shift with KeyUp, KeyDown** methods

Actions class

To create an object ‘action‘ of Selenium Actions class
Actions action=new Actions(driver);

To focus on element using WebDriver
action.moveToElement(element).perform();

To click on the element:
action.moveToElement(element).click().perform();

Mouse Over the element:

```
WebElement element = driver.findElement(By.xpath("xpath"));
//Create object 'action' of an Actions class
Actions action = new Actions(driver);
//Mouseover on an element
action.moveToElement(element).perform();
```

Available Methods in Selenium Actions Class

List of most commonly used keyboard and mouse events provided by the Selenium Actions class.

Method	Description
moveToElement(toElement)	It shifts the mouse to the center of the element.
contextClick()	Makes a context/right click at the existing mouse location.
doubleClick()	It performs a double-click at the existing mouse location.
dragAndDrop(source, target)	Invokes click-and-hold at the source location and moves to the location of the target element before releasing the mouse. **Parameters:** **source** – element to grab. **target** – element to release.

Right Click

We can right click an element or a webpage using Actions class in selenium and contextClick method.

```
WebElement element = driver.findElement(By.id("Selenium"));
// Create object for Actions class
Actions act = new Actions(driver);
// Perform Right click operation using action (object) on element.
act.contextClick(element).perform();
```

Double Click

Double click involves clicking on your mouse button twice to make a double click, and not just two clicks, in a short time the mouse button should be pressed twice, usually around half a second.

Used to do double-clicking to do different things, such as opening a program, opening a folder or choosing a word of text.

It double click on the current mouse position.

```
// Create object of Action class
Actions action = new Actions(driver);
// Find element using locator and store into WebElement
WebElement element = driver.findElement(By.id("elementId"));
// Perform Double click operation using action (object) on element.
action.doubleClick(element).perform();
```

Drag and Drop

1. Click and Hold the source
2. Move the element
3. Release the element

Different methods of Action class we will be used for Drag and Drop

- **clickAndHold(WebElement element)** – Clicks a web element at the middle(without releasing).

- **moveToElement(WebElement element)** – Moves the mouse pointer to the middle of the web element without clicking.

- **release(WebElement element)** – Releases the left click (which is in pressed state).

- **build()** – Generates a composite action
  Whenever we are executing more than one actions we need to use build() method to compile all these into a single step.

Drag and Drop

```
public class DragAndDropusingClickandHold {

 public static void main(String[] args) {
  System.setProperty("webdriver.gecko.driver", "D:\\geckodriver.exe");
  WebDriver driver = new FirefoxDriver();
  driver.get(URL);

  // Store Drag and Drop location into WebElement
  WebElement drag = driver.findElement(By.id("Drag_id"));
  WebElement drop = driver.findElement(By.id("Drop_id"));

  Actions act = new Actions(driver);   // Create object of actions class

   act.clickAndHold(drag).perform();  // Click & Hold drag Webelement

   act.moveToElement(drop).perform();  // Move to drop Webelement

   act.release(drop).perform();   // Release drag webelement into drop webelement
 }
}
```

Drag and Drop

```
public class DragAndDrop {

 public static void main(String[] args) {
  System.setProperty("webdriver.gecko.driver", "D:\\geckodriver.exe");
  WebDriver driver = new FirefoxDriver();
  driver.get(URL);

   Actions act = new Actions(driver);   // Create object of actions class

  // Store Drag and Drop location into WebElement
  WebElement drag = driver.findElement(By.xpath("put x path"));
   WebElement drop = driver.findElement(By.xpath("put x path"));

   act.dragAndDrop(drag, drop).perform();   // Drag element to destination

 }
}
```

Test Case

Task One

Ahead to http://uitestpractice.com/Students/Index

Click on the Actions

Click on the click me !

Handle the alert and click okay

Double Click Double Click Me !

Handle the alert and click okay

Close the browser

Test Case

Task Two

Ahead to http://uitestpractice.com/Students/Index

Click on the Actions

Handle the drag and drop

Close the browser

Test Case

Task Three

Ahead to http://uitestpractice.com/Students/Index

Click on the Actions

Click and hold on 1,2,3,4 boxes

Close the browser

Synchronization Waits in Selenium

What is wait ? Why we need it ?

It is the process of matching the speed of action and reaction, selenium does the action and web application shows the reaction.

Example: Click Login button gmail login (selenium does this action) web application navigates to the inbox(reaction).

Selenium Wait commands play an important role while executing Selenium tests. When a page is loaded to browser, the elements within that page may load at different time intervals.

This makes locating elements difficult, if the element is not present in the DOM, it will raise **ElementNotVisibleExceptionexception.**

Waits help the user to troubleshoot issues while redirecting to different web pages by refreshing the entire web page and reloading the new web elements.

Synchronization Waits in Selenium

In most of the cases the selenium does the action in micro or milliseconds but the application takes time in seconds.

At the same time selenium will not wait till it loads or becomes visible at the time selenium will not able to find the element within the java processing time so selenium throws “**NoSuchElementException**’.

We cannot alter the speed of the application, so we must slow down selenium.

Synchronization Waits in Selenium

There are several ways to synchronize the selenium with application

- pageLoadTimeout
- ImplicitWait
- ExplicitWait or WebdriverWait
- FluentWait

Page Load Timeout in Selenium Webdriver

Page load timeout in selenium requests/set the time limit for a page to load, If the page is not loaded within the given time frame selenium throws TimeOutException exception

Page Load timeout is applicable only to driver.get() and driver.navigate().to() methods in selenium webdriver

Setting Negative time limit makes the selenium to wait for the page load

Page load timeout is not applicable when user clicks a link to open a page.

driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);

Waits in Selenium Webdriver

Selenium Webdriver provides three types of waits:

ImplicitWait
ExplicitWait
FluentWait