Selenium

Class 10

Agenda

Taking screenshots in Selenium

Advanced xPath

Javascript Executor

Action Class

File Upload

Screenshot using Selenium

Screenshots are desirable for bug analysis. In automation, it is necessary to take the screenshot for verification so we can prove also that our test case has covered certain functionality or not.

Test cases may fail while executing the test cases. While we are executing the test cases manually we just take a screenshot and place in a result repository. The same can be done by using Selenium WebDriver.

It's very important to take screenshot when we execute a test script. When we execute huge number of test scripts, and if some test fails, we need to check why the test has failed.

It helps us to debug and identify the problem by seeing the screen shot.

Screenshot using Selenium

Failures may occur because of below reasons:

- Actual and Expected values are not matching
- When there is no element
- When page takes more time to load
- When an unexpected alert comes in to focus
- When there is Assertion issues

There is an interface called **TakesScreenshot** which provides **getScreenshotAs** method and which selenium uses to take screenshot.

Screenshot using Selenium

Our browser classes (Like ChromeDriver, FirefoxDriver ...) extends RemoteWebdriver and RemoteWedriver implements TakesScreenshot Interface along with Webdriver Interface.

We can take screenshot of a webpage using getScreenshotAs method from the TakesScreenshot, but we cannot initialize TakesScreenshot as it is an interface.

So to take screenshot of the page we have to cast our driver object to TakesScreenshot interface type:

TakesScreenshot scrShot =(TakesScreenshot)webdriver;

Note: casting is a process of converting 1 dataType/objectType to another

Screenshot using Selenium

Convert webdriver object to TakesScreenshot

TakesScreenshot scrShot =(TakesScreenshot)webdriver;

Call getScreenshotAs method to create image file

File SrcFile=scrShot.getScreenshotAs(OutputType.FILE);

Copy file at destination

FileUtils.copyFile(SrcFile, DestFile);

```java
public class ScreenShotDemo extends CommonMethods {

    public static void main(String[] args) throws InterruptedException {
        setUp("chrome", Constants.HRMS_URL);

        // login into HRMS
        String userName = "Admin";
        String password = "Hum@nhrm123";
        driver.findElement(By.id("txtUsername")).sendKeys(userName);
        driver.findElement(By.id("txtPassword")).sendKeys(password);
        driver.findElement(By.id("btnLogin")).click();
        // verify employee is logged in
        String uid = driver.findElement(By.id("welcome")).getText();
        if (uid.contains(userName)) {
            System.out.println("User "+userName+" is logged in");

            TakesScreenshot ts=(TakesScreenshot)driver;
            File screen=ts.getScreenshotAs(OutputType.FILE);
            try {
                FileUtils.copyFile(screen, new File("screenshots/HRMS/AdminLogin.png"));
            } catch (IOException e) {
                System.out.println(e.getMessage());
            }

        }else {
            System.out.println("User "+userName+" is NOT logged in");
        }

        driver.quit();
    }
}
```

Advanced xPath

Sometimes multiple elements will have the same attribute values or no attributes defined in its html and in this case we need to work on creating unique xpath.

To find an element which is not having an identity, We find the related elements of that element
and access that element using a relation between identified element

Advanced xPath

1. If Parent is having unique identification
   parentXpath/childTagName

2. If immediate child is having unique identification
   childXpath/..

3. If the next element is having unique identification
   nextElementXpath/preceding-sibling::tagName

4. If the previous element is having unique identification
   previousElementXpath/following-sibling::tagName

Test Case

TC 1: HRMS Login

1. Navigate to “http://166.62.36.207/humanresources/symfony/web/index.php/auth”
2. Login to the application by writing xpath based on “parent and child relation”

TC 2: HRMS Login

3. Navigate to “http://166.62.36.207/humanresources/symfony/web/index.php/auth”
4. Login to the application by writing xpath based on “following and preceding siblings”

JavascriptExecutor

JavaScriptExecutor is something which present in every selenium tool and in all the languages we can execute it on the browser.

JavaScriptExecutor allows you to run pure Javascript code irrespective of the Selenium language binding(Java, C#, Python etc.) you are using.

JavaScriptExecutor is an interface which provides the mechanism to execute Javascript through selenium driver.

We should go for JavaScriptExecutor **only** when we are not able to perform a particular task with our selenium like sometimes we may not be able click a element

JavascriptExecutor

Using JavaScriptExecutor we can perform:

- Click on some elements
- Scroll page
- Refresh page
- Highlight an element

JavaScriptExecutor provides “**execute script**” & "**executeAsyncScript**" methods, to run JavaScript in the context of the currently selected frame or window.

What is Javascript?

A Javascript is a small chunks of program that makes a website interactive. For example, a javascript could create a pop-up alert box , or provide a dynamic drop down menu.

JavaScript code can listen to event on the web page like reacting to a click on a button, reacting to when check a checkbox, or when we enter any value in to the text bar.

Javascript can change the content of the html page dynamically, it can also change the attributes of a web element like change the a button from disabled state to enabled state.

How to use JavascriptExecutor

We have to cast driver object into JavascriptExecutor type to use the methods present in the JavascriptExecutor interface.

cast the driver object to JavascriptExecutor
JavascriptExecutor js = (JavascriptExecutor) driver;

access the methods:
js.executeScript("javascript command");

Note: casting is a process of converting 1 dataType/objectType to another

Methods present in JavascriptExecutor

Perform Scroll on application using Selenium

```
JavascriptExecutor js = (JavascriptExecutor)driver;
js.executeScript("window.scrollBy(0,150)");
//Vertical scroll - down by 150 pixels
```

To scroll an app to specified elements

```
JavascriptExecutor je = (JavascriptExecutor) driver;
je.executeScript("arguments[0].scrollIntoView(true);", element);
```

Click Button using JavaScriptExecutor

```
JavascriptExecutor js = (JavascriptExecutor)driver;
js.executeScript("arguments[0].click();", element);
```

Highlight Element using JavaScriptExecutor

```
JavascriptExecutor js = (JavascriptExecutor)driver;
js.executeScript("arguments[0].style.backgroundColor='AnyColor'", element);
```

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

We can drag and drop an element or a webpage using Actions class in selenium and dragAndDrop method.

```
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"));

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

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

 }
}
```

Upload File using Selenium Webdriver

There are different ways to handle file uploads with Selenium Webdriver.
The first and the Easy way is simple case of just finding the element and typing the absolute path of the document into it.

```
WebElement element= driver.findElement(By.name("datafile"));
element.sendKeys("C:\Users\Sandesh\Desktop\testfile.txt");
```

Test Case

TC 1: Upload file and Take Screenshot

1. Navigate to “http://samples.gwtproject.org/samples/Showcase/Showcase.html#!CwFileUpload”
2. Upload file
3. Verify file got successfully uploaded and take screenshot

Selenium

Selenium has 4 components:

1. Selenium Integrated Development Environment (IDE) [outdated]
2. Selenium Remote Control (RC) [outdated]
3. Selenium WebDriver
4. Selenium Grid