**Practical Use of Different types of Selenium WebDriver Waits**

we will discuss about **different types of waits provided by the WebDriver**. We will also discuss about v**arious types of navigation options** available in WebDriver.

Waits help the user to troubleshoot issues while re-directing to different web pages by refreshing the entire web page and re-loading the new web elements. At times there can be Ajax calls as well. Thus, a time lag can be seen while reloading the web pages and reflecting the web elements.

Users are often found navigating through various web pages back and forth. Thus, navigate() commands/methods provided by the WebDriver helps the user to simulate the real time scenarios by navigating between the web pages with reference to the web browser’s history.

**WebDriver equips the user with two genesis of waits in order to handle the recurring page load**s, web element loads, appearance of windows, pop ups and error messages and reflection of web elements on the web page.

* Implicit Wait
* Explicit Wait

Let us discuss each of them in details considering practical approach.

### ****WebDriver Implicit Wait****

Implicit waits are used to provide a default waiting time (say 30 seconds) between each consecutive test step/command across the entire test script. Thus, subsequent test step would only execute when the 30 seconds have elapsed after executing the previous test step/command.

**Key Notes**

* Implicit wait is a single line of a code and can be declared in the setup method of the test script.
* When compared to Explicit wait, Implicit wait is transparent and uncomplicated. The syntax and approach is simpler than explicit wait.

Being easy and simple to apply, implicit wait introduces a few drawbacks as well. It gives rise to the test script execution time as each of the command would be ceased to wait for a stipulated amount of time before resuming the execution.

Thus, in order to trouble shoot this issue, WebDriver introduces Explicit waits where we can explicitly apply waits whenever the situation arises instead of forcefully waiting while executing each of the test step.

**Import Statements**

**import**java.util.concurrent.TimeUnit – To be able to access and apply implicit wait in our test scripts, we are bound to import this package into our test script.

**Syntax**  
drv.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

Include the above line of code into your test script soon after instantiation of WebDriver instance variable. Thus, this is all what is required to set an implicit wait into your test script.

**Code Walkthrough**

The implicit wait mandates to pass two values as parameters. The first argument indicates the time in the numeric digits that the system needs to wait. The second argument indicates the time measurement scale. Thus, in the above code, we have mentioned the “30” seconds as default wait time and the time unit has been set to “seconds”.

### ****WebDriver Explicit Wait****

Explicit waits are used to halt the execution till the time a particular condition is met or the maximum time has elapsed. Unlike Implicit waits, Explicit waits are applied for a particular instance only.

WebDriver introduces classes like WebDriverWait and ExpectedConditions to enforce Explicit waits into the test scripts. In the ambit of this discussion, we will use “gmail.com” as a specimen.

**Scenario to be automated**

1. Launch the web browser and open the “gmail.com”
2. Enter a valid username
3. Enter a valid password
4. Click on the sign in button
5. Wait for Compose button to be visible after page load

**WebDriver Code using Explicit wait**

Please take a note that for script creation, we would be using “Learning\_Selenium” project created in the former modules.

**Step 1**: Create a new java class named as “Wait\_Demonstration” under the “Learning\_Selenium” project.

**Step 2**: Copy and paste the below code in the “Wait\_Demonstration.java” class.

Below is the test script that is equivalent to the above mentioned scenario.

**package** com.webdriver.explicitwait;

**import** java.util.concurrent.TimeUnit;

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.Test;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.firefox.FirefoxDriver;

**import** org.openqa.selenium.support.ui.ExpectedConditions;

**import** org.openqa.selenium.support.ui.WebDriverWait;

/\*Import above packages prior to the script creation.

\* The packages refer to the Select class which

\* is required to handle the dropdown

\* \*/

/\*\*

\* **@author** Smita B Kumar

\*

\*/

**public** **class** WaitDemonstration {

// created reference variable for WebDriver

WebDriver driver;

@Before

**public** **void** setUp() **throws** InterruptedException {

// objects and variables instantiation

System.*setProperty*("webdriver.gecko.driver", "D:\\Selenium\\Software\\geckodriver-v0.13.0-win64\\geckodriver.exe");

// initializing driver variable using FirefoxDriver

driver=**new** FirefoxDriver();

// launching gmail.com on the browser

driver.get("https://gmail.com");

// maximized the browser window

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(10, TimeUnit.***SECONDS***);

}

@Test

**public** **void** test() **throws** InterruptedException {

// saving the GUI element reference into a "username" variable of WebElement type

WebElement username = driver.findElement(By.*id*("Email"));

// entering username

username.sendKeys("smitabrijesh");

// entering password

driver.findElement(By.*id*("Passwd")).sendKeys("password");

// clicking signin button

driver.findElement(By.*id*("signIn")).click();

// explicit wait - to wait for the compose button to be click-able

WebDriverWait wait = **new** WebDriverWait(driver,30);

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*xpath*("//div[contains(text(),'COMPOSE')]")));

driver.findElement(By.*xpath*("//div[contains(text(),'COMPOSE')]")).click();

/\*We create a reference variable “wait” for

\* WebDriverWait class and instantiate it using WebDriver

\* instance and maximum wait time for the execution

\* to layoff. The maximum wait time quoted is

\* measured in “seconds”.

\* \*/

}

@After

**public** **void** teardown() {

// closes all the browser windows opened by web driver

driver.quit();

}

}

**Import Statements**

* **import**org.openqa.selenium.support.ui.ExpectedConditions
* **import**org.openqa.selenium.support.ui.WebDriverWait
* Import above packages prior to the script creation. The packages refer to the Select class which is required to handle the dropdown.

**Object Instantiation for WebDriverWait class**

------------

WebDriverWait wait =**new**WebDriverWait(drv,30);

We create a reference variable “wait” for WebDriverWait class and instantiate it using WebDriver instance and maximum wait time for the execution to layoff. The maximum wait time quoted is measured in “seconds”.

The WebDriver instantiation was discussed in the initial modules of WebDriver.

### ****Expected Condition****

wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//div[contains(text(),'COMPOSE')]")));

drv.findElement(By.xpath("//div[contains(text(),'COMPOSE')]")).click();

The above command waits for a stipulated amount of time or an expected condition to occur whichever occurs or elapses first.

Thus to be able to do this, we use the “wait” reference variable of WebDriverWait class created in the previous step with ExpectedConditions class and an actual condition which is expected to occur. Therefore, as soon as the expected condition occurs, the program control would move to the next execution step instead of forcefully waiting for the entire 30 seconds.

In our specimen, we wait for the “compose” button to be present and loaded as a part of home page load and thus, then we move forward with calling the click command on the “compose” button.

**Types of Expected Conditions**

ExpectedConditions class provides a great help to deal with scenarios where we have to ascertain for a condition to occur before executing the actual test step.

ExpectedConditions class comes with a wide range of expected conditions that can be accessed with the help of the WebDriverWait reference variable and until() method.

**Let us discuss a few of them at length:**

**#1) elementToBeClickable()** – The expected condition waits for an element to be clickable i.e. it should be present/displayed/visible on the screen as well as enabled.

**Sample Code**  
wait.until(ExpectedConditions.elementToBeClickable(By.xpath(“//div[contains(text(),’COMPOSE’)]”)));

**#2) textToBePresentInElement() –**The expected condition waits for an element having a certain string pattern.

**Sample Code**  
wait.until(ExpectedConditions.textToBePresentInElement(By.xpath(“//div[@id= ‘forgotPass'”), “text to be found”));

**#3) alertIsPresent()-** The expected condition waits for an alert box to appear.

**Sample Code**  
wait.until(ExpectedConditions.alertIsPresent()) !=null);

**#4) titleIs()** – The expected condition waits for a page with a specific title.

**Sample Code**  
wait.until(ExpectedConditions.titleIs(“gmail”));

**#5) frameToBeAvailableAndSwitchToIt()** – The expected condition waits for a frame to be available and then as soon as the frame is available, the control switches to it automatically.

**Sample Code**  
wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.id(“newframe”)));

### ****Navigation Using WebDriver****

There is a very common user action where the user clicks on the back and forward buttons of the web browser back n forth to navigate to the different web pages visited in the current session on the browser’s history. Thus to simulate such actions performed by the users, WebDriver introduces Navigate commands.

**Let us examine these commands in detail:**

**#1) navigate().back()**  
This command lets the user to navigate to the previous web page.

**Sample code:**  
driver.navigate().back();  
The above command requires no parameters and takes back the user to the previous webpage in the web browser’s history.

**#2) navigate().forward()**  
This command lets the user to navigate to the next web page with reference to the browser’s history.

**Sample code:**  
driver.navigate().forward();  
The above command requires no parameters and takes forward the user to the next webpage in the web browser’s history.

**#3) navigate().refresh()**  
This command lets the user to refresh the current web page there by reloading all the web elements.

**Sample code:**  
driver.navigate().refresh();  
The above command requires no parameters and reloads the web page.

**#4) navigate().to()**  
This command lets the user to launch a new web browser window and navigate to the specified URL.

**Sample code:**  
driver.navigate().to(“http://google.com”);  
The above command requires a web URL as a parameter and then it opens the specified URL on a freshly launched web browser.

### ****Conclusion****

In this module, we tried to make you acquainted with the WebDriver’s waits. We discussed and exercised both the explicit and the implicit waits. At the same time, we also discussed about the different navigate commands.

**Here are the cruxes of this article:**

* WebDriver enables the user to choose amongst the available waits to handle situations where the execution flow may require a sleep for few seconds in order to load the web elements or to meet a specific condition. There are two types of waits available in WebDriver.
  + Implicit Wait
  + Explicit Wait
* **Implicit waits** are used to provide a default waiting time between each consecutive test step/command across the entire test script. Thus, subsequent test step would only execute when the specified amount of time have elapsed after executing the previous test step/command.
* **Explicit waits** are used to halt the execution till the time a particular condition is met or the maximum time has elapsed. Unlike Implicit waits, Explicit waits are applied for a particular instance only.
* WebDriver introduces classes like WebDriverWait and ExpectedConditions to enforce Explicit waits
* **ExpectedConditions** class provides a great help to deal with scenarios where we have to ascertain for a condition to occur before executing the actual test step.
* ExpectedConditions class comes with a wide range of expected conditions that can be accessed with the help of the WebDriverWait reference variable and until() method.
* **Navigate() methods**/commands are used to simulate the user behavior while navigating between various web pages back and forth.