1. What are the limitations in selenium web driver?

* It does not support and non web-based applications, it only supports web based applications.
* Its an open source tool so in case of any technical issues you need to rely on the selenium community forums to get your issue resolved.
* You need to know at least one of the supported language very well in order to automate your application successfully.
* No inbuilt reporting capability so you need plugins like JUnit and TestNG for test reports.
* Lot of challenges with IE browser.

1. Installing/configure selenium?

Step 1: Go to Oracle official site – “JAVA download”, download Java Platform, Standard Edition. All the recent releases are available on the page.

webdriver installtion with eclipse 1

Step 2: As soon as you click on the Download button, following screen would appear. Accept the License agreement for Java installation and choose amongst the various catalogued Java Development Kit’s. Select the one that best suits your system configuration.

webdriver installtion with eclipse 2

Remember to download JDK (Java development kit). The kit comes with a JRE (Java Runtime environment). Thus the user isn’t required to download and install the JRE separately.

Eclipse IDE Installation

Step 1: Go to Eclipse official website and navigate to its download page – Eclipse download. Download Eclipse IDE for Java EE developers. All the recent releases are available on the page.

Make sure you opt and download the appropriate eclipse IDE as per your system configuration. There are two download links available for 64 bit windows operating system and 32-bit windows operating system.

webdriver installtion with eclipse 3

Step 2: As soon as we click on the download link, the user is re-directed to the fresh page securing information about the current download. Click on the download icon and you are done.

webdriver installtion with eclipse 4

It may take a few minutes before you can download the complete zip folder.

Step 3: Once downloaded, copy the folder and place it in the desired location on your file system.

webdriver installtion with eclipse 5

Step 4: Extract the zipped folder, a folder named as eclipse can be seen. The folder embodies all the required application and source files.

webdriver installtion with eclipse 6

Step 5: Launch the eclipse IDE using “eclipse.exe” residing inside the eclipse folder. Refer the above illustration for the same.

Step 6: The application will prompt you to specify the workspace location. Workspace is that location where all your eclipse projects will be residing. Enter/Browse the desired location or the user can simply opt for the default location and click on the OK button.

webdriver installtion with eclipse 7

Configuring WebDriver

As we would be using Java as the programming language for this series and in order to create test scripts in java, we would have to introduce language- specific client drivers. Thus, let us begin with the downloading of Selenium Java Client Libraries.

Download the Selenium Java Client Libraries

Step 1: Go to Selenium’s official website and navigate to its download page – “http://docs.seleniumhq.org/download/”. Refer the section in the below illustration where you can find Client Libraries listed for distinct programming languages. Click on the download link for Java Client Library.

webdriver installtion with eclipse 8

Step 2: Once downloaded, copy the folder and place it in the desired location on your file system.

Step 3: Extract the zipped folder, a folder named as “Selenium-2.41.0.zip“can be seen. The folder embodies all the required jar files which enable users to create test scripts in Java.

Thus these libraries can be configured in Eclipse IDE.

Configuring Libraries with Eclipse IDE

Step 1: Navigate towards Eclipse IDE. Create a new java based project following File -> New -> Java Project. Refer the following figure for the same.

webdriver installtion with eclipse 9

Step 2: Provide a user defined name for your Java Project. Let us provide the name as Learning\_Selenium and Click on the Finish Button. The newly created project can be viewed at the left side of the screen in the package explorer panel.

Step 3: Create a new Java class named as “First\_WebdriverClass” under the source folder by right clicking on it and navigating to New -> class.

Step 4: Now let us configure the libraries into our Java project. For this, select the project and Right click on it. Select “Properties” within the listed options. The following screen appears, Select “Java Build Path” from the options.

webdriver installtion with eclipse 10

Step 5: By default, “Libraries” tab is opened. If not, click on the “Libraries” tab. Then, click on the “Add External Jars…” button. Browse to the location where we have saved the extracted folder for Java Client Libraries.

Step 6: Select all the JAR files present in the “selenium-java-2.41.0” folder and click on open button within the dialog box. The properties dialog box should look like the below illustration.

webdriver installtion with eclipse 11

Step 7: Click on the “OK” button within the dialog box so as to complete the configuration part of Selenium Libraries in our java project.

The project will look like the following:

webdriver installtion with eclipse 12

Available Drivers

There are a number of driver classes available in WebDriver, each catering a specific web browser. Each browser has a different driver implementation in WebDriver.

In WebDriver, a few of the browsers can be automated directly where as some of the web browsers require an external entity to be able to automate and execute the test script. This external entity is known as Driver Server. Thus, user is required to download the Driver Server for different web browsers.

Notice that there is a separate Driver Server for each of the web browser and user cannot use one Driver Server for web browsers other than the one it is designated for.

Below is the list of available web browsers and their corresponding Server Drivers.

webdriver installtion with eclipse 13

1. What are the different ways of locating elements in selenium?

The different types of locator are:

* ID
* Name
* Link Text
* CSS Selector

Tag and ID

Tag and class

Tag and attribute

Tag, class, and attribute

Inner text

* DOM (Document Object Model)

getElementById

getElementsByName

dom:name

dom: index

* XPath

Finding elements by ID

ids are the preferred way to locate elements on a page for 2 main reasons:

According to w3c ids are supposed to be unique on an html page. This makes ids a very explicit and reliable way to locate elements on the page.

Also, all browsers also have highly efficient methods to get an object on the page using their ids. This makes id locators the fastest type of locator in Selenium.

Let us now look at how to use id locators using this html code as an example:

<form name="loginForm">

Login Username: <input id="username" name="login" type="text" />

Password: <input id="password" name="password" type="password" />

<input name="login" type="submit" value="Login" />

</form>

In the above code, the username and password text fields are can be set using their ids. The locators for them would be

driver.findElement(By.id(username));

Even though this is a great locator, it is not realistic for all elements on a page to have ids. The developers add ids to key elements on the page to better control the look and feel or provide the dynamic user interaction. However, ids should also be added to elements that are frequently interacted within tests … to make the pages more testable. Automated test script authors should consider adding, or requesting addition of, ids to these key elements on the page.

In some cases, the ids on an element cannot be reliably used in a test. For instance, if you are displaying objects stored in the database, the objects ids could contain the database id in it, for instance book1347 could be the id for book who’s database id is 1347. In such situations it may not be possible to predict and use the ids and so you may need to use other ways described below to find elements.

Finding elements by name

Generally ids are added to elements when they want to be referenced from css or javascript and names are added to form fields. When referencing element from javascript either can be used. From a test automation standpoint, whenever id is not available/ usable, you should try to use the name instead.

Using the same example above, the way you would find the submit button would be:

driver.findElement(By.name("login"));

There is one big difference between the id and name attributes though … name attributes don’t have to be unique in a page. If there are multiple elements with the same name, then the first element in the page is selected. So, in this example, if another button or form named “login” was present of added later, it could cause the test to fail.

Finding links by text

This locator identifies links by the text in them. Let us look at an example:

<html>

<body>

...

<a href="signin.html">Sign In</a> to modify your account details.

...

</body>

</html>

To click this hyperlink using the anchor tag’s text, you can use the By.linkText() locator:

driver.findElement(By.linkText("Sign In"));

If there are multiple elements with text “Sign In”, the first one is selected.

Btw, this is called linkText because it is used for hyperlinks. In Selenium if you used the link=textPattern locator, you could use it to locate other elements like div, span, td etc. In WebDriver, this locator works only for links.

Another common case is when we need to find links by a portion of the text it contains. In such cases you can find it by specifying the partial text. For example:

driver.findElement(By.partialLinkText("Sign"));

Finding elements by XPath

XPath is a very powerful language to express which element to identify. If you use it correctly, it can produce very reliable and low maintenance locators, but if you use it incorrectly, it can create very brittle test cases.

Let us see some examples:

<table name="cart">

<tr id="item1">

<td class="name item">Mp3 Download: foobar.mp3</td>

<td class="name item"><input name="qty" class="name item formfield disabled" /></td>

</tr>

<tr id="item2">

<td class="name item">Mp3 Player</td>

<td class="name item"><input id="item2\_quantity" name="qty" class="name item formfield required" type="text" /></td>

</tr>

...

</table>

You can target the highlighted input field using the following XPath expressions:

//table/tr[2]/td/input

//input[@id='item2\_quantity']

(//table[@name='cart']//input)[2]

//input[contains(@class,'required')]

//input[contains(@class,'required') and type='text']

And for the last option, your java call would look like:

driver.findElement(By.xpath("//input[contains(@class='required') and type='text']"));

As you can guess, some of these expressions will not be as reliable as others. Of these //table/tr[2]/td/input is the worst because it would break even with slightest modification to the page structure. It can take some time to learn XPath if you aren’t familiar with it, but it is worth the time if you plan to spend a lot of time writing UI automated tests. In any case do not rely on tools, including selenium IDE, to generate the right xpath expression for you. They can help you get started but they are usually bad at identifying the more reliable XPaths.

XPath namespaces

There are some special cases you should be aware of when you work with XPath, like when you are trying to interact with SVG. Like in the example here, the html looks something like this:

<div id="svgchart">

...

<svg xmlns="http://www.w3.org/2000/svg">

<g>

<path .../>

</g>

...

</svg>

...

</div>

Here, if you use XPath like:

//svg

you will get ERROR org.openqa.selenium.NoSuchElementException: Unable to locate element. This is because the svg element is in a different namespace. You will have to specify your xpath with the namespace uri like this instead:

//\*[local-name()='svg' and namespace-uri()='http://www.w3.org/2000/svg']

XPath performance

So, if XPath’s are so versatile, why doesn’t everyone prefer these? It’s because they are often the slowest, especially in older versions of IE! There are some ways you can make XPath’s faster, but they still are a few times slower than ids or names.

Finding elements by CSS

css locators can be used to identify a large number of elements on a page. Let us look at this html snippet for instance:

<table name="cart">

<tr id="item1">

<td class="label">Mp3 Download: foobar.mp3</td>

<td class="item"><input name="qty" class="formfield disabled" /></td>

</tr>

<tr id="item2">

<td class="label">Mp3 Player</td>

<td class="item"><input id="item2\_quantity" name="qty" class="formfield required" type="text" /></td>

</tr>

...

</table>

In this case the css locator variations possible for the highlighted input field are:

input.required

input[class~='required']

input.required[type='text']

#item2\_quantity

input#item2\_quantity

The css locator may not be as expressive as XPath, but it generally executes faster.

Finding elements by class

This is more of a convinience mechanism to identify elements. In the css example above, instead of using

driver.findElements(By.css("input[class~='required']"));

you could use

driver.findElements(By.class("required"));

Getting elements by DOM

DOM stands for Document Object Model. DOM is convention for representing objects in HTML documents.

<form id="loginForm">

Login Username: <input id="username" name="username" type="text" />

Password: <input name="password" type="password" />

<input name="login" type="submit" value="Log in" />

</form>

In this page, the dom expression for the highlighted input field would be:

document.forms[0].elements[0]

document.forms['loginForm'].elements['username']

document.forms['loginForm'].username

document.getElementById('username')

For those who have used Selenium 1 API, you would expect to find a By.dom() equivalent api, but it doesn’t exist. However, you still can get a handle to these elements using dom expressions by using the following code snippet instead:

driver.findElement(byDom("document.forms[0].elements[0]"));

public By byDom(String domExpression) {

final Object o = ((JavascriptExecutor) driver).executeScript("return " + domExpression + ";");

if (o instanceof WebElement) {

return new By() {

@Override

public List<WebElement> findElements(SearchContext searchContext) {

return new ArrayList<WebElement>() {

{

add((WebElement) o);

}

};

}

};

}

}

The three key aspects of this code that you should note,

The driver can be casted to JavascriptExecutor and other interfaces which will provide additional capabilities.

executeScript() in JavascriptExecutor returns an object which can be casted to a WebElement if the javascript expression returns a dom element.

1. Which is fastest way to identify elements in web page?

Finding web elements with Selenium WebDriver by ID is usually the fastest option, but here is the list of the best and fastest selectors Selenium WebDriver Tests run faster:

ID selectors (By.ID – Matches by @id attribute)

* IDs are the safest, fastest locator option and should always be your first choice
* IDs should be unique in every page according to W3C website
* even if the DOM changes, if the ID is still there, then WebDriver can still locate it
* always try and get extra IDs added into the code, this makes testers life easier
* fastest locator as it uses the document.getElementById() javascript command which is optimised by many browsers

1. what is absolute and relative path in xpath?

Absolute XPath :

It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.

The key characteristic of XPath is that it begins with the single forward slash(/) ,which means you can select the element from the root node.

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

students.xsl

<?xml version = "1.0" encoding = "UTF-8"?>

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:template match = "/" >

<html>

<body>

<h3>Details of each Students. </h3>

<table border = "1">

<tr bgcolor = "#9acd32">

<th>Roll No</th>

<th>First Name</th>

<th>Last Name</th>

<th>Nick Name</th>

<th>Marks</th>

</tr>

<tr>

<td><xsl:value-of select = "/class/student[1]/@rollno"/></td>

<td><xsl:value-of select = "/class/student[1]/firstname"/></td>

<td><xsl:value-of select = "/class/student[1]/lastname"/></td>

<td><xsl:value-of select = "/class/student[1]/nickname"/></td>

<td><xsl:value-of select = "/class/student[1]/marks"/></td>

</tr>

<tr>

<td>

<xsl:value-of select = "/class/student/@rollno"/>

</td>

<td><xsl:value-of select = "/class/student[2]/firstname"/></td>

<td><xsl:value-of select = "/class/student[2]/lastname"/></td>

<td><xsl:value-of select = "/class/student[2]/nickname"/></td>

<td><xsl:value-of select = "/class/student[2]/marks"/></td>

</tr>

<tr>

<td>

<xsl:value-of select = "/class/student[3]/@rollno"/>

</td>

<td><xsl:value-of select = "/class/student[3]/firstname"/></td>

<td><xsl:value-of select = "/class/student[3]/lastname"/></td>

<td><xsl:value-of select = "/class/student[3]/nickname"/></td>

<td><xsl:value-of select = "/class/student[3]/marks"/></td>

</tr>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

Relative xpath :

For Relative Xpath the path starts from the middle of the HTML DOM structure. Its start with the double forward slash (//), which means it can search the element anywhere at the webpage.

You can starts from the middle of the HTML DOM structure and no need to write long xpath.

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

students.xsl

<?xml version = "1.0" encoding = "UTF-8"?>

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:template match = "/" >

<html>

<body>

<h3>Details of each Students. </h3>

<table border = "1">

<tr bgcolor = "#9acd32">

<th>Roll No</th>

<th>First Name</th>

<th>Last Name</th>

<th>Nick Name</th>

<th>Marks</th>

</tr>

<xsl:for-each select = "/class/student">

<tr>

<td><xsl:value-of select = "@rollno"/></td>

<td><xsl:value-of select = "firstname"/></td>

<td><xsl:value-of select = "lastname"/></td>

<td><xsl:value-of select = "nickname"/></td>

<td><xsl:value-of select = "marks"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

1. write code on how to use xpath functions?

XPath is defined as XML path. It is a syntax or language for finding any element on the web page using XML path expression. XPath is used to find the location of any element on a webpage using HTML DOM structure.

Syntax for XPath:

XPath contains the path of the element situated at the web page. Standard syntax for creating XPath is.

Xpath=//tagname[@attribute='value']

// : Select current node.

Tagname: Tagname of the particular node.

@: Select attribute.

Attribute: Attribute name of the node.

Value : Value of the attribute.

XPath axes search different nodes in XML document from current context node. XPath Axes are the methods used to find dynamic elements, which otherwise not possible by normal XPath method having no ID , Classname, Name, etc.

Contains() : Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information.

In this example, we tried to identify the element by just using partial text value of the attribute. In the below XPath expression partial value 'sub' is used in place of submit button. It can be observed that the element is found successfully.

Complete value of 'Type' is 'submit' but using only partial value 'sub'.

Xpath=//\*[contains(@type,'sub')]

Complete value of 'name' is 'btnLogin' but using only partial value 'btn'.

Xpath=.//\*[contains(@name,'btn')]

Start-with function: Start-with function finds the element whose attribute value changes on refresh or on any operation on the webpage. In this expression, match the starting text of the attribute is used to find the element whose attribute changes dynamically. You can also find the element whose attribute value is static (not changes).

For example -: Suppose the ID of particular element changes dynamically like:

Id=" message12"

Id=" message345"

Id=" message8769"

and so on.. but the initial text is same. In this case, we use Start-with expression.

In the below expression, there are two elements with an id starting "message"(i.e., 'User-ID must not be blank' & 'Password must not be blank'). In below example, XPath finds those element whose 'ID' starting with 'message'.

Xpath=//label[starts-with(@id,'message')]

Text() :In this expression , with text function we find the element with exact text match as shown below. In our case, we find the element with text "UserID".

Xpath=//td[text()='UserID']

Following : Selects all elements in the document of the current node( ) [ UserID input box is the current node] as shown in the below screen.

Xpath=//\*[@type='text']//following::input

Descendant: Selects the descendants of the current node as shown in the below screen.

In the below expression, it identifies all the element descendants to current element ( 'Main body surround' frame element) which means down under the node (child node , grandchild node, etc.).

Xpath=//\*[@id='rt-feature']//descendant::a

1. different types of waits or synchronization in selenium web driver?

Synchronization or Waits can be done in two ways.

1. Explicit Waits

2. Implicit Waits

Explicit Waits:

This can be achieved in 2 ways.

Thread.sleep:

Thread.sleep waits the specified time irrespective of the object state.

Ex: Thread.sleep(30000);

Here the execution is halted for 30 Sec., even if the object you are looking exists in 10 sec. So here tool unnecessarily waits for 20 sec.

Execution wont wait after 30 sec.s even if the object does not available, so the chances of your Test fails.

WebDriverWait:

We can tell the tool to wait only till the Condition met. Once the condition is met, the tool proceed with the next step.

This can be done with WebDriverWait in conjunction with ExpectedConditions Class.

There are few methods supported in ExpectedConditions class to support synchronisation.

Here is the example:

WebDriverWait wait = new WebDriverWait(driver, 30);

WebElement o\_element = wait.until(ExpectedConditions.elementToBeClickable(By.id("Object Id")));

Here the tool waits a maximum time of 30 Sec., if the object you are looking is displayed in 10 sec. then the execution proceeds with the next step afte 10 secs. rather than waiting for 30 secs.

If you dont want to include any methods in ExpectedConditions class, then you can use below code:

WebDriver driver = new FirefoxDriver();

driver.get("Your URL");

WebElement o\_Element = (new WebDriverWait(driver, 30))

.until(new ExpectedCondition<WebElement>(){

@Override

public WebElement apply(WebDriver d) {

return d.findElement(By.id("Object Id"));

}});

Here WebDriverWait by default calls the ExpectedCondition every 500 milliseconds until it returns successfully or wait for maximum of 30 sec.

Implicit Waits:

An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available. The default setting is 0. Once set, the implicit wait is set for the life of the WebDriver object instance.

WebDriver driver = new FirefoxDriver();

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

driver.get("Your URL");

WebElement o\_Element = driver.findElement(By.id("Object Id"));

Fluent Wait:

FluentWait contains 3 parameters.

WithTimeOut: defines the maximum amount of time it will wait for any element to be visible.

pollingEvery’ : takes time as an argument e.g. 5 seconds. It will check the status of element after every 5 seconds.

ignoring: we can define the type of exceptions need to be ignored during element search.

Fluent waitJava

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(30, TimeUnit.SECONDS) // wait maximum 30 seconds

.pollingEvery(5, TimeUnit.SECONDS) // check after every 5 seconds

.ignoring(NoSuchElementException.class); // ignore NoSuchElementException

WebElement foo = wait.until(new Function<WebDriver, WebElement>() {

public WebElement apply(WebDriver driver) {

return driver.findElement(By.id("Email"));

}

});

1. writing code with expectedwaits conditions?

The explicit wait is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or the maximum time exceeded before throwing an "ElementNotVisibleException" exception.

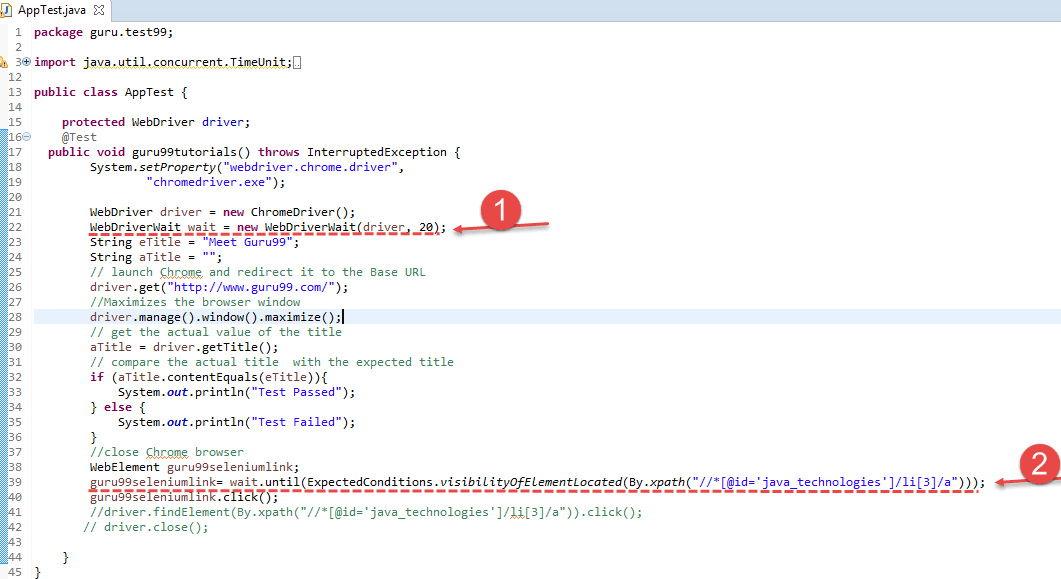
The explicit wait is an intelligent kind of wait, but it can be applied only for specified elements. Explicit wait gives better options than that of an implicit wait as it will wait for dynamically loaded Ajax elements.

Once we declare explicit wait we have to use "ExpectedCondtions" or we can configure how frequently we want to check the condition using Fluent Wait. These days while implementing we are using Thread.Sleep() generally it is not recommended to use

In the below example, we are creating reference wait for "WebDriverWait" class and instantiating using "WebDriver" reference, and we are giving a maximum time frame of 20 seconds.

Syntax:

WebDriverWait wait = new WebDriverWait(WebDriverRefrence,TimeOut);



Explanation of Code

Code line 39: In the above example, wait for the amount of time defined in the "WebDriverWait" class or the "ExpectedConditions" to occur whichever occurs first.

//locating the element on the web page using expected conditions

WebElement guru99seleniumlink;

guru99seleniumlink= wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//\*[@id='java\_technologies']/li[3]/a")));

guru99seleniumlink.click();

The above java code states that we are waiting for an element for the time frame of 20 seconds as defined in the "WebDriverWait" class on the webpage until the "ExpectedConditions" are met and the condition is "visibilityofElementLocated".

The following are the Expected Conditions that can be used in Explicit Wait

alertIsPresent()

elementSelectionStateToBe()

elementToBeClickable()

elementToBeSelected()

frameToBeAvaliableAndSwitchToIt()

invisibilityOfTheElementLocated()

invisibilityOfElementWithText()

presenceOfAllElementsLocatedBy()

presenceOfElementLocated()

textToBePresentInElement()

textToBePresentInElementLocated()

textToBePresentInElementValue()

titleIs()

titleContains()

visibilityOf()

visibilityOfAllElements()

visibilityOfAllElementsLocatedBy()

visibilityOfElementLocated()

1. How to save screen shots using selenium web driver?

In selenium webdriver, we can take the screen shot using the below command.

File scrFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

Check the framework example of Taking ScreenShot for ONLY Failed Tests using TestNG

The below example explains how to take the screen shot when the test fails.

import java.io.File;

import org.apache.commons.io.FileUtils;

import org.openqa.selenium.By;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.TakesScreenshot;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.Test;

public class takeScreenShotExample{

public WebDriver driver;

@Test

public void openBrowser() throws Exception {

driver = new FirefoxDriver();

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

driver.get("http://www.google.com");

try{

//the below statement will throw an exception as the element is not found, Catch block will get executed and takes the screenshot.

driver.findElement(By.id("testing")).sendKeys("test");

//if we remove the below comment, it will not return exception and screen shot method will not get executed.

//driver.findElement(By.id("gbqfq")).sendKeys("test");

}

catch (Exception e){

System.out.println("I'm in exception");

//calls the method to take the screenshot.

getscreenshot();

}

}

public void getscreenshot() throws Exception

{

File scrFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

//The below method will save the screen shot in d drive with name "screenshot.png"

FileUtils.copyFile(scrFile, new File("D:\\screenshot.png"));

}

}

1. How to handle multiple windows in selenium web driver?

When we have multiple windows in test automation, all we need to do is switching the focus from one window to another. Let us understand the same in the following way:

Window A has a link "Link1" and we need to click on the link (click event).

Window B displays and we perform some actions.

The entire process can be fundamentally segregated into following steps:

Step 1 : Clicking on Link1 on Window A

A new Window B is opened.

Step 2 : Save reference for Window A

Step 3 : Create reference for Window B

Step 3 : Move Focus from Window A to Window B

Window B is active now

Step 3 : Perform Actions on Window B

Complete the entire set of Actions

Step 4 : Move Focus from Window B to Window A

Window A is active now

Let us understand the same with a small coding example.

import java.util.List;

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;

public class MultipleWindowsHandle {

WebDriver driver;

@Before

public void setup() throws Exception {

driver=new FirefoxDriver();

String URL="http://www.seleniummaster.com";

driver.get(URL);

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

}

@Test

public void test() throws Exception {

// Opening site

driver.findElement(By.xpath("//img[@alt='SeleniumMasterLogo']")).click();

// Storing parent window reference into a String Variable

String Parent\_Window = driver.getWindowHandle();

// Switching from parent window to child window

for (String Child\_Window : driver.getWindowHandles())

{

driver.switchTo().window(Child\_Window);

// Performing actions on child window

driver.findElement(By.id("dropdown\_txt")).click();

List dropdownitems=driver.findElements(By.xpath("//div[@id='DropDownitems']//div"));

int dropdownitems\_Size=dropdownitems.size();

System.out.println("Dropdown item size is:"+dropdownitems\_Size);

((WebElement) dropdownitems.get(1)).click();

driver.findElement(By.xpath("//\*[@id='anotherItemDiv']")).click();

}

//Switching back to Parent Window

driver.switchTo().window(Parent\_Window);

//Performing some actions on Parent Window

driver.findElement(By.className("btn\_style")).click();

}

@After

public void close() {

driver.quit();

}

}

1. How to launch webpage using chrome driver?

Like Internet Explorer, Google Chrome also requires user to download an external utility ChromeDriver, a standalone server to execute test scripts on Google Chrome. User can download the ChromeDriver file from here. The rest of the process would remain the same as that of Internet Explorer.

Syntax:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class ChromeDriverTest {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// Create WebDriver reference

WebDriver driver;

// Set path for Chrome Driver executable

System.setProperty("webdriver.chrome.driver", "C:/lib/chromedriver.exe");

// Launch ChromeDriver

driver = new ChromeDriver();

// Open the web page

driver.get("http://google.com");

// Enter the text in the search box

WebElement searchText = driver.findElement(By.name("q"));

searchText.sendKeys("ChromeDriver");

// Close the driver

driver.quit();

}

}

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class ChromeDriverTest {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// Create WebDriver reference

WebDriver driver;

// Set path for Chrome Driver executable

System.setProperty("webdriver.chrome.driver", "C:/lib/chromedriver.exe");

// Launch ChromeDriver

driver = new ChromeDriver();

// Open the web page

driver.get("http://google.com");

// Enter the text in the search box

WebElement searchText = driver.findElement(By.name("q"));

searchText.sendKeys("ChromeDriver");

// Close the driver

driver.quit();

}

}

1. What is desired capabilities in selenium web driver?

DesiredCapabilities help to set the properties for WebDriver

Example of using DesiredCapabilities in selenium Grid:

Desired capabilities used to support Selenium GridJava

import java.net.MalformedURLException;

import java.net.URL;

import org.openqa.selenium.Platform;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.openqa.selenium.remote.RemoteWebDriver;

public class DesiredCapabilityExample {

public static void main(String[] args) throws MalformedURLException {

DesiredCapabilities caps = new DesiredCapabilities().firefox();

caps.setVersion("30");

caps.setPlatform(Platform.WIN8);

WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), caps);

}

}

import java.net.MalformedURLException;

import java.net.URL;

import org.openqa.selenium.Platform;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.openqa.selenium.remote.RemoteWebDriver;

public class DesiredCapabilityExample {

public static void main(String[] args) throws MalformedURLException {

DesiredCapabilities caps = new DesiredCapabilities().firefox();

caps.setVersion("30");

caps.setPlatform(Platform.WIN8);

WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), caps);

}

}

1. How to set language while opening website?

Internationalization is a process of designing a software application so that it can be adapted to various languages and regions without any changes

Localization is a process of adapting internationalized software for a specific region or language by adding local specific components and translating text.

If we want to check whether our application is properly internationalized , then we will manually change the language preferences in the browser itself.But if we want to check the same using WebDriver then we have to change the user language preferences.

Using Firefox Browser :

FirefoxProfile profile = new FirefoxProfile();

//setting the locale french : ‘fr’

profile.setPreference(“intl.accept\_languages”,”fr”);

driver = new FirefoxDriver(profile);

driver.get(“http://google.co.in&#8221;);

Using Chrome Browser :

System.setProperty(“webdriver.chrome.driver”,”D:/DollarArchive/chromedriver.exe”);

ChromeOptions options = new ChromeOptions();

options.addArguments(“–lang= sl”);

ChromeDriver driver = new ChromeDriver(options);

driver.get(“http://google.co.in&#8221;);

Unfortunately it wont work for IE browser, We need to change it manually.

1. How to handle windows based popups (upload and dropdown)?

There are many cases, where a application displays multiple windows when you open a website. Those are may be advertisements or may be a kind of information showing on popup windows. We can handle multiple windows using Windows Handlers in selenium webdriver.

Step 1: After opening the website, we need to get the main window handle by using driver.getWindowHandle();

The window handle will be in a form of lengthy alpha numeric

Step 2: We now need to get all the window handles by using driver.getWindowHandles();

Step 3: We will compare all the window handles with the main Window handles and perform the operation the window which we need.

The below example shows how to handle multiple windows and close all the child windows which are not need. We need to compare the main window handle to all the other window handles and close them.

package com.pack;

import java.util.Set;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.Test;

public class WindowExamples {

static WebDriver driver;

@Test

public void test\_CloseAllWindowsExceptMainWindow() {

driver = new FirefoxDriver();

// It will open Naukri website with multiple windows

driver.get("http://www.naukri.com/");

// To get the main window handle

String windowTitle= getCurrentWindowTitle();

String mainWindow = getMainWindowHandle(driver);

Assert.assertTrue(closeAllOtherWindows(mainWindow));

Assert.assertTrue(windowTitle.contains("Jobs - Recruitment"), "Main window title is not matching");

}

public String getMainWindowHandle(WebDriver driver) {

return driver.getWindowHandle();

}

public String getCurrentWindowTitle() {

String windowTitle = driver.getTitle();

return windowTitle;

}

//To close all the other windows except the main window.

public static boolean closeAllOtherWindows(String openWindowHandle) {

Set<String> allWindowHandles = driver.getWindowHandles();

for (String currentWindowHandle : allWindowHandles) {

if (!currentWindowHandle.equals(openWindowHandle)) {

driver.switchTo().window(currentWindowHandle);

driver.close();

}

}

driver.switchTo().window(openWindowHandle);

if (driver.getWindowHandles().size() == 1)

return true;

else

return false;

}

}

1. Write code to verify any application login page is working or not?

In my database I have the username = user@javachap.com and password = javachap

If I run the code below, it passes the test although the username and password does not exist in my database.

@Test

public void testLogin()

{

String username="abc";

String password="123";

boolean valueFound=false;

// Check the db

try

{

pstmt=conn.prepareCall("select \* from user where USR\_EMAIL=? and USD\_PASSWORD=?");

pstmt.setString(1,username);

pstmt.setString(2,password);

rs=pstmt.executeQuery();

valueFound = rs.next();

}

catch(Exception e)

{

// report some error

}

1. How to select items from drop down/select box?
2. How to know if check box is checked or not in webpage?

//Checking

public void CheckingChkbox(WebElement chkbx1){

boolean checkstatus;

checkstatus=chkbx1.isSelected();

if (checkstatus==true){

System.out.println("Checkbox is already checked");

}

else

{

chkbx1.click();

System.out.println("Checked the checkbox");

}

}

//Unchecking

public void UnCheckingChkbox(WebElement chkbx1){

boolean checkstatus;

checkstatus=chkbx1.isSelected();

if (checkstatus==true) {

chkbx1.click();

System.out.println("Checkbox is unchecked");

}

else

{

System.out.println("Checkbox is already unchecked");

}

}

1. tell me code to pass values from parent window to child window?
2. Write code to findout if all links are working or not?

First we will try to find all anchor tags on the page by using Webdriver. By using the below syntax:

List<WebElement> anchorTagsList = driver.findElements(By.tagName("a"));

Let us look into the example :

package com.linked;

import java.util.List;

import org.apache.http.HttpResponse;

import org.apache.http.client.HttpClient;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.impl.client.HttpClientBuilder;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class FindBrokenLinksExample {

private WebDriver driver;

private int invalidLinksCount;

@BeforeClass

public void setUp() {

driver = new FirefoxDriver();

driver.get("http://google.com");

}

@Test

public void validateInvalidLinks() {

try {

invalidLinksCount = 0;

List<WebElement> anchorTagsList = driver.findElements(By

.tagName("a"));

System.out.println("Total no. of links are "

+ anchorTagsList.size());

for (WebElement anchorTagElement : anchorTagsList) {

if (anchorTagElement != null) {

String url = anchorTagElement.getAttribute("href");

if (url != null && !url.contains("javascript")) {

verifyURLStatus(url);

} else {

invalidLinksCount++;

}

}

}

System.out.println("Total no. of invalid links are "

+ invalidLinksCount);

} catch (Exception e) {

e.printStackTrace();

System.out.println(e.getMessage());

}

}

@AfterClass

public void tearDown() {

if (driver != null)

driver.quit();

}

public void verifyURLStatus(String URL) {

HttpClient client = HttpClientBuilder.create().build();

HttpGet request = new HttpGet(URL);

try {

HttpResponse response = client.execute(request);

// verifying response code and The HttpStatus should be 200 if not,

// increment invalid link count

////We can also check for 404 status code like response.getStatusLine().getStatusCode() == 404

if (response.getStatusLine().getStatusCode() != 200)

invalidLinksCount++;

} catch (Exception e) {

e.printStackTrace();

}

}

}

1. Write code on how to use javascriptexecutor?

JavaScriptExecutor is an interface which provides mechanism to execute Javascript through selenium driver. It provides “executescript” & "executeAsyncScript" methods, to run JavaScript in the context of the currently selected frame or window.

Syntax:-

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript(Script,Arguments);

example:

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class JavaScriptExecuter {

public static void main(String[] args) {

WebDriver driver = new FirefoxDriver();

//Launching the browser application

driver.get("http://www.uftHelp.com");

//Adding wait

driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);

//Maximize window

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

//Creating the Javascriptexecutor interface object by Type casting

JavascriptExecutor js = (JavascriptExecutor)driver;

//Fetching the Domain Name

String sDomain = js.executeScript("return document.domain;").toString();

System.out.println("Domain = "+sDomain);

//Fetching the URL

String sURL = js.executeScript("return document.URL;").toString();

System.out.println("URL = "+sURL);

//Fetching the Title

String sTitle = js.executeScript("return document.title;").toString();

System.out.println("Title = "+sTitle);

//Vertical scroll - down by 200 pixels

js.executeScript("window.scrollBy(0,200)");

System.out.println("Successfully did the vertical scroll by 200px");

}

}

1. Difference between assert and verify?

When an “assert” command fails, the test execution will be aborted. So when the Assertion fails, all the test steps after that line of code are skipped. The solution to overcoming this issue is to use a try-catch block. We use the Assertion in the try catch block. Mostly, the assert command is used when the end result of the check value should pass to continue to the next step.

In simple words, if the assert condition is true then the program control will execute the next test step but if the condition is false, the execution will stop and further test step will not be executed.

Verify command in selenium:

When a “verify” command fails, the test will continue executing and logging the failure. Mostly, the Verify command is used to check non-critical things. In such cases where we move forward even though the end result of the check value is failed.

In simple words, there wont be any halt in the test execution even though the verify condition is true or false.

Note: In TestNG, we use only Assert Statements.

package testWordpress;

import org.testng.Assert;

import org.testng.annotations.Test;

public class AssertionsTest {

@Test

public void testMultiply() {

System.out.println(&quot;Before Error &quot;);

Assert.assertEquals(21, multiply(10, 5));

System.out.println(&quot;After Error &quot;);

}

public int multiply(int x, int y) {

return x / y;

}

}

System.out.println(“After Error “); will never executed. 😦

Please check eclipse console to conform.

We can resolve this problem by using Try-Catch block.

How?

Lets check out following code

package testWordpress;

import org.testng.Assert;

import org.testng.annotations.Test;

public class AssertionsTest {

@Test

public void testMultiply() {

System.out.println(&quot;Before Error &quot;);

try{

Assert.assertEquals(21, multiply(10, 5));

}catch(Throwable t){

// recovered

// java code to fail the test

System.out.println(&quot;After Error &quot;);

}

}

public int multiply(int x, int y) {

return x / y;

}

}

1. Difference between driver.close and driver.quit methods?

driver.close – It closes the the browser window on which the focus is set.

driver.quit – It basically calls driver.dispose method which in turn closes all the browser windows and ends the WebDriver session gracefully.

You should use driver.quit whenever you want to end the program. It will close all opened browser window and terminates the WebDriver session. If you do not use driver.quit at the end of program, WebDriver session will not close properly and files would not be cleared off memory. This may result in memory leak errors.

1. Common exceptions in selenium?

Most common Exceptions:

1) NoSuchElementException : FindBy method can’t find the element.

2) StaleElementReferenceException : This tells that element is no longer appearing on the DOM page.

3) TimeoutException: This tells that the execution is failed because the command did not complete in enough time.

4) ElementNotVisibleException: Thrown to indicate that although an element is present on the DOM, it is not visible, and so is not able to be interacted with

5) ElementNotSelectableException: Thrown to indicate that may be the element is disabled, and so is not able to select.

24) how to handle ajax calls in selenium?

Ajax is a technique used for creating fast and dynamic web pages. This technique is asynchronous and uses a combination of Javascript and XML .

It will updates the part/s of a web page without reloading the whole page.

Some of the famous applications that uses AJAX technique are Gmail, Google Maps, Facebook, Youtube, etc

So, let’s discuss the options that we can deploy to handle AJAX calls in Selenium webdriver.

1- Using <Thread.Sleep(Time In Ms)> For Handling AJAX Controls.

<Thread.sleep()> is an obvious choice for handling AJAX calls. But it may not give you the best result. Instead, a test could break intermittently if the server response time exceeds the time specified in sleep. Additionally, the test has to wait for the given time even in a situation of the timely response. Though keeping all the odds aside, this method does work, and we’ve tested it as working.

2- Using JavaScript To Handle AJAX Calls In Selenium Webdriver.

This method is only useful if the web application has jQuery in use to handle AJAX calls. Since jQuery uses a mechanism which keeps the no. of active AJAX calls in check, we can utilize this information to find out their final status.

Here is a sample code to showcase the handling of AJAX controls using Selenium Webdriver. You can integrate it in your test execution class.

Use JavaScript to Handle AJAX Calls in Selenium Webdriver.Java

public void waitForAjaxControls(int timeoutInSeconds) {

System.out

.println("Querying active AJAX controls by calling jquery.active");

try {

if (browser instanceof JavascriptExecutor) {

JavascriptExecutor jsDriver = (JavascriptExecutor) browser;

for (int i = 0; i < timeoutInSeconds; i++) {

Object numberOfAjaxConnections = jsDriver

.executeScript("return jQuery.active");

// return should be a number

if (numberOfAjaxConnections instanceof Long) {

Long n = (Long) numberOfAjaxConnections;

System.out

.println("Number of active jquery AJAX controls: "

+ n);

if (n.longValue() == 0L)

break;

}

Thread.sleep(1000);

}

} else {

System.out.println("Web driver: " + browser

+ " can't run javascript.");

}

} catch (InterruptedException e) {

System.out.println(e);

}

}

public void waitForAjaxControls(int timeoutInSeconds) {

System.out

.println("Querying active AJAX controls by calling jquery.active");

try {

if (browser instanceof JavascriptExecutor) {

JavascriptExecutor jsDriver = (JavascriptExecutor) browser;

for (int i = 0; i < timeoutInSeconds; i++) {

Object numberOfAjaxConnections = jsDriver

.executeScript("return jQuery.active");

// return should be a number

if (numberOfAjaxConnections instanceof Long) {

Long n = (Long) numberOfAjaxConnections;

System.out

.println("Number of active jquery AJAX controls: "

+ n);

if (n.longValue() == 0L)

break;

}

Thread.sleep(1000);

}

} else {

System.out.println("Web driver: " + browser

+ " can't run javascript.");

}

} catch (InterruptedException e) {

System.out.println(e);

}

}

3- How To Use Implicit Wait To Handle AJAX Calls In Selenium Webdriver.

An implicit wait is a Webdriver mechanism to query the DOM for a specified time duration to locate an element or a set of elements till they become available. The default timeout value is 0.

Once you define it, the implicit wait is available for the lifetime of the Webdriver instance.

Define Implicit wait for handling of AJAX calls.Java

WebDriver browser = new FirefoxDriver();

browser.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);

browser.get("http://www.techbeamers.com/");

WebElement ajaxControl = browser.findElement(By.id("DummyElement"));

WebDriver browser = new FirefoxDriver();

browser.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);

browser.get("http://www.techbeamers.com/");

WebElement ajaxControl = browser.findElement(By.id("DummyElement"));

4- How To Use Webdriver Explicit Wait For Handling AJAX Controls.

It is yet another Webdriver’s built-in feature to handle AJAX calls. Just like the <Thread.sleep()>, you can get it working when no other tricks other work.

Use of An Explicit Wait to Handle AJAX Controls.Java

WebDriver browser = new FirefoxDriver();

browser.get("http://www.techbeamers.com/");

WebElement ajaxControl = (new WebDriverWait(browser, 15))

.until(ExpectedConditions.presenceOfElementLocated(By

.id("DummyElement")));

WebDriver browser = new FirefoxDriver();

browser.get("http://www.techbeamers.com/");

WebElement ajaxControl = (new WebDriverWait(browser, 15))

.until(ExpectedConditions.presenceOfElementLocated(By

.id("DummyElement")));

5- Using Webdriver Fluent Wait To Handle AJAX Calls.

It’s an implementation of the Webdriver’s Wait interface which brings both the timeout and polling interval to use. The Fluent wait makes use of the timeout to wait for a condition, also the frequency for the no. of attempts.

Its example you can see in the last section where we’ve demonstrated the combined use of <FluentWait> and the <WebdriverWait>.

6- Using <WebdriverWait> For Handling The AJAX Calls.

It is one of the best Webdriver strategies to handle the AJAX controls. It allows you to implant a condition to check at regular intervals and break to next step as soon as the condition get fulfilled.

Apart from the <WebdriverWait>, we also use the <ExpectedCondition> to get the entire mechanism in place.

1. We have web table, need to click on second row from table?

Browser("browser").page("page").webtable("webtable").object.Rows(2).Click

1. Tell me steps to verify child form has proper data or not?
2. How to assign the value to text box other than sendkeys method?

Any other way is only to use native Javascript action to enter the value in the text box:

WebDriver driver = new FirefoxDriver();

JavascriptExecutor executor = (JavascriptExecutor)driver;

executor.executeScript("document.getElementById("textbox\_id").value='new value';);

1. Selenium grid, how to execute scripts on multiple browser?

When we say parallel test execution in Selenium is achieved by Selenium Grid than statement is partly incorrect.

Testing Framework like testng is used for parallel test execution

Selenium Grid is used for automated testing execution on Distributed systems parallely

Selenium Grid Concept

In Selenium Grid architecture we have 1 Hub which acts as central controlling authority and connecting nodes. Nodes must be registered to Hub

Consider node as port opened on machine (loacal or remote). Each node is capable to opening multiple browsers.

On single machine we can have multiple nodes opened

Grid Hub decides what tests needs to routed on which node, we can’t control them

Best Practices of Selenium Grid

Single machine should open one node only

Each node should run only single type of browser

We need various driver objects for various threads to be run parallely, so create driver as ThreadLocal variable

Let consider an example what we would be achieving in our Grid:

SeleniumGrid

Start Hub and Nodes

Download “Selenium Standalone Server” from “http://www.seleniumhq.org/download/” on all 3 machines

Goto machine 1 and open command prompt.

Navigate to location where jar is located.

Start Hub by command: java -jar selenium-server-standalone-2.48.2.jar -role hub

Open browser and navigate to http://localhost:4444/grid/console and verify hub is started by checking below image.HubStarted

Goto machine 2 and open command prompt

Navigate to location where jar is located.

Start Node by command: java -Dwebdriver.chrome.driver={path to chromedriver.exe}-jar selenium-server-standalone-2.48.2.jar -role webdriver -hub -port 5560 -browser browserName=chrome,maxInstances=2,maxSession=2

Goto machine 1 and in browser navigate to http://localhost:4444/grid/console and verify node is started by checking below image.SeleniumGridNode1

Goto machine 3 and open command prompt.

Navigate to location where jar is located.

Start Node by command: java -jar selenium-server-standalone-2.48.2.jar -role webdriver -hub -port 5557 -browser browserName=firefox,maxInstances=5,maxSession=2

Goto machine 1 and in browser navigate to http://localhost:4444/grid/console and verify node is started by checking below image. SeleniumGridNode2

Nodes can be opened for various settings like browser, platform, version.

Now Selenium Hub and Nodes are created, lets make some @Test and execute them.

Make Project on any machine, my project structure as below:PackageStructure

Source code for files are given below.

Execute testng.xml and execution will start parallely on both machines.

testng.xml

<?xml version="1.0" encoding="UTF-8"?>

<suite name="Parallel test suite" parallel="classes" thread-count="2">

<test name="Regression 1">

<parameter name="myBrowser" value="firefox"/>

<classes>

<class name="myPackage.TestParallel" />

<class name="myPackage.TestParallel" />

</classes>

</test>

<test name="Regression 2">

<parameter name="myBrowser" value="chrome"/>

<classes>

<class name="myPackage.TestParallel" />

<class name="myPackage.TestParallel" />

</classes>

</test>

</suite>

BaseClass.java

package myPackage;

package myPackage;

import java.net.MalformedURLException;

import java.net.URL;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.Platform;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.openqa.selenium.remote.RemoteWebDriver;

import org.testng.annotations.AfterClass;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Parameters;

public class BaseClass {

//ThreadLocal will keep local copy of driver

public static ThreadLocal<RemoteWebDriver> dr = new ThreadLocal<RemoteWebDriver>();

@BeforeTest

//Parameter will get browser from testng.xml on which browser test to run

@Parameters("myBrowser")

public void beforeClass(String myBrowser) throws MalformedURLException{

RemoteWebDriver driver = null;

if(myBrowser.equals("chrome")){

DesiredCapabilities capability = new DesiredCapabilities().chrome();

capability.setBrowserName("chrome");

capability.setPlatform(Platform.WINDOWS);

driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), capability);

}

else if(myBrowser.equals("firefox")){

DesiredCapabilities capability = new DesiredCapabilities().firefox();

capability.setBrowserName("firefox");

capability.setPlatform(Platform.WINDOWS);

driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), capability);

}

//setting webdriver

setWebDriver(driver);

getDriver().manage().window().maximize();

getDriver().manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

}

public WebDriver getDriver() {

return dr.get();

}

public void setWebDriver(RemoteWebDriver driver) {

dr.set(driver);

}

@AfterClass

public void afterClass(){

getDriver().quit();

dr.set(null);

}

}

TestParallel.java

package myPackage;

import java.net.MalformedURLException;

import java.net.URL;

import org.openqa.selenium.By;

import org.openqa.selenium.Platform;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.openqa.selenium.remote.RemoteWebDriver;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class TestParallel extends BaseClass {

@Test

public void test\_01() throws InterruptedException, MalformedURLException{

try{

getDriver().get("http://www.w3schools.com/");

getDriver().findElement(By.xpath("html/body/div[2]/div/a[4]")).click();

//Wait intentially added to show parallelism execution

Thread.sleep(10000);

getDriver().findElement(By.xpath("//\*[@id='gsc-i-id1']")).sendKeys("test");

Thread.sleep(5000);

}

catch(Exception e){

System.out.println(e);

}

}

}

1. Write code for drag/drop in selenium?

package practiceTestCases;

import java.util.concurrent.TimeUnit;

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.interactions.Action;

import org.openqa.selenium.interactions.Actions;

public class DragAndDrop {

public static void main(String[] args) throws InterruptedException {

WebDriver driver = new FirefoxDriver();

String URL = "http://www.dhtmlx.com/docs/products/dhtmlxTree/index.shtml";

driver.get(URL);

// It is always advisable to Maximize the window before performing DragNDrop action

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

driver.manage().timeouts().implicitlyWait(10000, TimeUnit.MILLISECONDS);

WebElement From = driver.findElement(By.xpath(".//\*[@id='treebox1']/div/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[1]/td[4]/span"));

WebElement To = driver.findElement(By.xpath(".//\*[@id='treebox2']/div/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[1]/td[4]/span"));

Actions builder = new Actions(driver);

Action dragAndDrop = builder.clickAndHold(From)

.moveToElement(To)

.release(To)

.build();

dragAndDrop.perform();

}

}

1. Write code for right click in selenium?

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.Keys;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.interactions.Actions;

public class Right\_click {

public static void main(String args[]) throws Exception{

// Initialize WebDriver

WebDriver driver = new FirefoxDriver();

// Wait For Page To Load

driver.manage().timeouts().implicitlyWait(120,TimeUnit.SECONDS);

// Go to Myntra Page

driver.get("http://www.myntra.com/");

// Maximize Window

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

WebElement R1 = driver.findElement(By.xpath("//a[@href='/shop/men?src=tn&nav\_id=5']"));

// Initialize Actions class object

Actions builder = new Actions(driver);

// Perform Right Click on MEN and Open "Men" content in a new tab

builder.contextClick(R1).sendKeys(Keys.ARROW\_DOWN).sendKeys(Keys.ENTER).perform();

//ContextClick() is a method to use right click

/\* Perform Right Click on MEN and Open "Men" content in a new different Window

builder.contextClick(hindiLanguage).sendKeys

(Keys.ARROW\_DOWN).sendKeys(Keys.ARROW\_DOWN).sendKeys(Keys.ENTER).perform();

//closing current driver window

driver.close();

\*/

}

}

1. Write code for scroll to specific element?

I will be using this application as Sample- Sample Application for Demo

package Demo;

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class TestScroll {

public static void main(String[] args) {

// Start browser

WebDriver driver=new FirefoxDriver();

// Maximize browser

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

// Pass application URL

driver.get("http://manos.malihu.gr/repository/custom-scrollbar/demo/examples/complete\_examples.html");

// Create instance of Javascript executor

JavascriptExecutor je = (JavascriptExecutor) driver;

//Identify the WebElement which will appear after scrolling down

WebElement element = driver.findElement(By.xpath(".//\*[@id='mCSB\_3\_container']/p[3]"));

// now execute query which actually will scroll until that element is not appeared on page.

je.executeScript("arguments[0].scrollIntoView(true);",element);

// Extract the text and verify

System.out.println(element.getText());

}

}

package Demo;

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class TestScroll {

public static void main(String[] args) {

// Start browser

WebDriver driver=new FirefoxDriver();

// Maximize browser

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

// Pass application URL

driver.get("http://manos.malihu.gr/repository/custom-scrollbar/demo/examples/complete\_examples.html");

// Create instance of Javascript executor

JavascriptExecutor je = (JavascriptExecutor) driver;

//Identify the WebElement which will appear after scrolling down

WebElement element = driver.findElement(By.xpath(".//\*[@id='mCSB\_3\_container']/p[3]"));

// now execute query which actually will scroll until that element is not appeared on page.

je.executeScript("arguments[0].scrollIntoView(true);",element);

// Extract the text and verify

System.out.println(element.getText());

}

}

1. How to launch different browsers (IE,firefox,safari)

Firefox:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class FirefoxDriverTest {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// Create webDriver reference

WebDriver driver;

// Launch FirefoxDriver

driver = new FirefoxDriver();

// Open the web page

driver.get("http://google.com");

// Enter the text in the search box

WebElement searchText = driver.findElement(By.name("q"));

searchText.sendKeys("FirefoxDriver");

// Close the driver

driver.quit();

}

}

IE:

There are 3 simple steps to be able to execute test scripts on Internet Explorer:

User can download the server from here and keep it in a desired location.

Set the system’s property to point to the location where the server file is kept.

Launch the Internet Explorer browser.

Syntax:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.ie.InternetExplorerDriver;

public class InternetExplorerDriverTest {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// Create WebDriver reference

WebDriver driver;

// Set path for Internet Explorer Driver executable

System.setProperty("webdriver.ie.driver", "C:/lib/IEDriverServer.exe");

// Launch InternetExplorerDriver

driver = new InternetExplorerDriver();

// Open the web page

driver.get("http://google.com");

// Enter the text in the search box

WebElement searchText = driver.findElement(By.name("q"));

searchText.sendKeys("InternetExplorerDriver");

// Close the driver

driver.quit();

}

}

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.ie.InternetExplorerDriver;

public class InternetExplorerDriverTest {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// Create WebDriver reference

WebDriver driver;

// Set path for Internet Explorer Driver executable

System.setProperty("webdriver.ie.driver", "C:/lib/IEDriverServer.exe");

// Launch InternetExplorerDriver

driver = new InternetExplorerDriver();

// Open the web page

driver.get("http://google.com");

// Enter the text in the search box

WebElement searchText = driver.findElement(By.name("q"));

searchText.sendKeys("InternetExplorerDriver");

// Close the driver

driver.quit();

}

}

Safari:

SafariDriver comes bundled with the Selenium server’s package, thus user is not required to download or set any external entity. It is readily available to execute the test scripts on Safari. Thus, user can directly launch the Safari Browser similar to that of Mozilla Firefox.

Syntax:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.safari.SafariDriver;

public class SafariDriverTest {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// Create webDriver reference

WebDriver driver;

// Launch SafariDriver

driver = new SafariDriver();

// Open the web page

driver.get("http://google.com");

// Enter the text in the search box

WebElement searchText = driver.findElement(By.name("q"));

searchText.sendKeys("SafariDriver");

// Close the driver

driver.quit();

}

}

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.safari.SafariDriver;

public class SafariDriverTest {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// Create webDriver reference

WebDriver driver;

// Launch SafariDriver

driver = new SafariDriver();

// Open the web page

driver.get("http://google.com");

// Enter the text in the search box

WebElement searchText = driver.findElement(By.name("q"));

searchText.sendKeys("SafariDriver");

// Close the driver

driver.quit();

}

}

1. How to use autoit to implement download/upload files?

To upload a file in Selenium Webdriver we will create AutoIT script, which will handle file-uploaded window, and then we will combine Selenium script with AutoIt scripts.

Click on Upload button you will get file uploader we will handle the same using AutoIt.

Step 1- Open Editor and Finder Tool

Step 2– We need to write script to upload file so we will use some method of AutoIt.

Each method will have some own functionality

ControlFocus-This will give focus on the window

ControlSetText-This will set the file path

ControlClick-This will click on button

Step 1-

Click on Browse button , a new window will open now open finder tool and Click on Finder tool and drag to the file name as I shown in below screenshot.

This will give all the detail about that window and file name section info; we will use only some attribute like window title, class, and instance.

Open AutoIt Editor and Write Script

In ControlClick method we will give control id of open button

Step 2-

Save the script to a particular location with some unique name.

Note- By default script will be saved as .au3 extension

Step 3– Now Compile the script so for compiling right click on file and Select compile script this will generate a .exe file of the file.

Step 4- Now write Selenium program and add this .exe file and run your program

Here is the code

package demo;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class DemoFileUpload {

public static void main(String[] args) throws Exception {

// This will open Firefox browser

WebDriver driver=new FirefoxDriver();

// This will maximize browser to full screen

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

// This will open respective URL

driver.get("your application url");

// This will click on Upload button

driver.findElement(By.xpath("//\*[@type='file']")).click();

// This will invoke AutoIT script here give the path of the script

//and this will throw IO exception so u can use throw or try catch

// In my case I am using throws

Runtime.getRuntime().exec("C:\\Users\\mukesh\_otwani\\Desktop\\AutoItScripts\\blogUpload.exe");

// Once you will run this program AutoIt script will be invoked and respective f//ile will be attached

}

}

package demo;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class DemoFileUpload {

public static void main(String[] args) throws Exception {

// This will open Firefox browser

WebDriver driver=new FirefoxDriver();

// This will maximize browser to full screen

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

// This will open respective URL

driver.get("your application url");

// This will click on Upload button

driver.findElement(By.xpath("//\*[@type='file']")).click();

// This will invoke AutoIT script here give the path of the script

//and this will throw IO exception so u can use throw or try catch

// In my case I am using throws

Runtime.getRuntime().exec("C:\\Users\\mukesh\_otwani\\Desktop\\AutoItScripts\\blogUpload.exe");

// Once you will run this program AutoIt script will be invoked and respective f//ile will be attached

}

}