Q #10) What are the different types of locators in Selenium?

The locator can be termed as an address that identifies a web element uniquely within the webpage. Thus, to identify web elements accurately and precisely we have different types of locators in Selenium:

ID

ClassName

Name

TagName

LinkText

PartialLinkText

Xpath

CSS Selector

DOM

Q #11) What is the difference between assert and verify commands?

Assert: Assert command checks whether the given condition is true or false. Let’s say we assert whether the given element is present on the web page or not. If the condition is true then the program control will execute the next test step but if the condition is false, the execution would stop and no further test would be executed.

Verify: Verify command also checks whether the given condition is true or false. Irrespective of the condition being true or false, the program execution doesn’t halt i.e. any failure during verification would not stop the execution and all the test steps would be executed.

Q #12) What is an XPath?

XPath is used to locate a web element based on its XML path. XML stands for Extensible Markup Language and is used to store, organize and transport arbitrary data. It stores data in a key-value pair which is very much similar to HTML tags. Both being markup languages and since they fall under the same umbrella, XPath can be used to locate HTML elements.

The fundamental behind locating elements using XPath is the traversing between various elements across the entire page and thus enabling a user to find an element with the reference of another element.

Q #13) What is the difference between “/” and “//” in Xpath?

Single Slash “/” – Single slash is used to create Xpath with absolute path i.e. the xpath would be created to start selection from the document node/start node.

Double Slash “//” – Double slash is used to create Xpath with relative path i.e. the xpath would be created to start selection from anywhere within the document.

Q #17) Which is the latest Selenium tool?

WebDriver

Q #18) How do I launch the browser using WebDriver?

The following syntax can be used to launch Browser:

WebDriver driver = new FirefoxDriver();

WebDriver driver = new ChromeDriver();

WebDriver driver = new InternetExplorerDriver();

Q #19) What are the different types of Drivers available in WebDriver?

The different drivers available in WebDriver are:

FirefoxDriver

InternetExplorerDriver

ChromeDriver

SafariDriver

OperaDriver

AndroidDriver

IPhoneDriver

HtmlUnitDriver

Q #20) What are the different types of waits available in WebDriver?

There are two types of waits available in WebDriver:

Implicit Wait

Explicit Wait

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, the subsequent test step would only execute when the 30 seconds have elapsed after executing the previous test step/command.

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.

Q #21) How to type in a textbox using Selenium?

The user can use sendKeys(“String to be entered”) to enter the string in the textbox.

Syntax:

WebElement username = drv.findElement(By.id(“Email”));

// entering username

username.sendKeys(“sth”);

Q #22) How can you find if an element in displayed on the screen?

WebDriver facilitates the user with the following methods to check the visibility of the web elements. These web elements can be buttons, drop boxes, checkboxes, radio buttons, labels etc.

isDisplayed()

isSelected()

isEnabled()

Syntax:

isDisplayed():

boolean buttonPresence = driver.findElement(By.id(“gbqfba”)).isDisplayed();

isSelected():

boolean buttonSelected = driver.findElement(By.id(“gbqfba”)).isSelected();

isEnabled():

boolean searchIconEnabled = driver.findElement(By.id(“gbqfb”)).isEnabled();

Q #23) How can we get a text of a web element?

Get command is used to retrieve the inner text of the specified web element. The command doesn’t require any parameter but returns a string value. It is also one of the extensively used commands for verification of messages, labels, errors etc displayed on the web pages.

Syntax:

String Text = driver.findElement(By.id(“Text”)).getText();

Q #24) How to select value in a dropdown?

The value in the dropdown can be selected using WebDriver’s Select class.

Syntax:

selectByValue:

Select selectByValue = new Select(driver.findElement(By.id(“SelectID\_One”)));

selectByValue.selectByValue(“greenvalue”);

selectByVisibleText:

Select selectByVisibleText = new Select (driver.findElement(By.id(“SelectID\_Two”)));

selectByVisibleText.selectByVisibleText(“Lime”);

selectByIndex:

Select selectByIndex = new Select(driver.findElement(By.id(“SelectID\_Three”)));

selectByIndex.selectByIndex(2);

Q #25) What are the different types of navigation commands?

Following are the navigation commands:

navigate().back() – The above command requires no parameters and takes back the user to the previous webpage in the web browser’s history.

Sample code:

driver.navigate().back();

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();

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();

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(“https://google.com”);

Q #26) How to click on a hyper link using linkText?

driver.findElement(By.linkText(“Google”)).click();

The command finds the element using link text and then click on that element and thus the user would be re-directed to the corresponding page.

The above-mentioned link can also be accessed by using the following command.

driver.findElement(By.partialLinkText(“Goo”)).click();

The above command finds the element based on the substring of the link provided in the parenthesis and thus partialLinkText() finds the web element with the specified substring and then clicks on it.

Q #27) How to handle frame in WebDriver?

An inline frame acronym as iframe is used to insert another document within the current HTML document or simply a web page into a web page by enabling nesting.

Select iframe by id

driver.switchTo().frame(“ID of the frame“);

Locating iframe using tagName

driver.switchTo().frame(driver.findElements(By.tagName(“iframe”).get(0));

Locating iframe using index

frame(index)

driver.switchTo().frame(0);

frame(Name of Frame)

driver.switchTo().frame(“name of the frame”);

frame(WebElement element)

Select Parent Window

driver.switchTo().defaultContent();

Q #28) When do we use findElement() and findElements()?

findElement(): findElement() is used to find the first element in the current web page matching to the specified locator value. Take a note that only first matching element would be fetched.

Syntax:

WebElement element = driver.findElements(By.xpath(“//div[@id='example']//ul//li”));

findElements(): findElements() is used to find all the elements in the current web page matching to the specified locator value. Take a note that all the matching elements would be fetched and stored in the list of WebElements.

Syntax:

List <WebElement> elementList = driver.findElements(By.xpath(“//div[@id='example']//ul//li”));

Q #29) How to find more than one web element in the list?

At times, we may come across elements of the same type like multiple hyperlinks, images etc arranged in an ordered or unordered list. Thus, it makes absolute sense to deal with such elements by a single piece of code and this can be done using WebElement List.

// Storing the list

2

List <WebElement> elementList = driver.findElements(By.xpath("//div[@id='example']//ul//li"));

3

// Fetching the size of the list

4

int listSize = elementList.size();

5

for (int i=0; i<listSize; i++)

6

{

7

// Clicking on each service provider link

8

serviceProviderLinks.get(i).click();

9

// Navigating back to the previous page that stores link to service providers

10

driver.navigate().back();

11

}

11

}

Q #30) What is the difference between driver.close() and driver.quit command?

close(): WebDriver’s close() method closes the web browser window that the user is currently working on or we can also say the window that is being currently accessed by the WebDriver. The command neither requires any parameter nor does it return any value.

quit(): Unlike close() method, quit() method closes down all the windows that the program has opened. Same as close() method, the command neither requires any parameter nor does is return any value.

Q #32) How can we handle web-based pop-up?

WebDriver offers the users a very efficient way to handle these pop-ups using Alert interface. There are the four methods that we would be using along with the Alert interface.

void dismiss() – The dismiss() method clicks on the “Cancel” button as soon as the pop-up window appears.

void accept() – The accept() method clicks on the “Ok” button as soon as the pop-up window appears.

String getText() – The getText() method returns the text displayed on the alert box.

void sendKeys(String stringToSend) – The sendKeys() method enters the specified string pattern into the alert box.

Syntax:

// accepting javascript alert

Alert alert = driver.switchTo().alert();

alert.accept();

Q #33) How can we handle windows based pop up?

Selenium is an automation testing tool which supports only web application testing, that means, it doesn’t support testing of windows based applications. However Selenium alone can’t help the situation but along with some third-party intervention, this problem can be overcome. There are several third-party tools available for handling window based pop-ups along with the selenium like AutoIT, Robot class etc.

Q #34) How to assert the title of the web page?

//verify the title of the web page

assertTrue(“The title of the window is incorrect.”,driver.getTitle().equals(“Title of the page”));

Q #35) How to mouse hover on a web element using WebDriver?

WebDriver offers a wide range of interaction utilities that the user can exploit to automate mouse and keyboard events. Action Interface is one such utility which simulates the single user interactions.

Thus, In the following scenario, we have used Action Interface to mouse hover on a drop down which then opens a list of options.

/ Instantiating Action Interface

2

Actions actions=new Actions(driver);

3

// howering on the dropdown

4

actions.moveToElement(driver.findElement(By.id("id of the dropdown"))).perform();

5

// Clicking on one of the items in the list options

6

WebElement subLinkOption=driver.findElement(By.id("id of the sub link"));

7

subLinkOption.click();

Q #36) How to retrieve CSS properties of an element?

The values of the css properties can be retrieved using a get() method:

Syntax:

driver.findElement(By.id(“id“)).getCssValue(“name of css attribute”);

driver.findElement(By.id(“id“)).getCssValue(“font-size”);

Q #37) How to capture screenshot in WebDriver?

1

import org.junit.After;

2

import org.junit.Before;

3

import org.junit.Test;

4

import java.io.File;

5

import java.io.IOException;

6

import org.apache.commons.io.FileUtils;

7

import org.openqa.selenium.OutputType;

8

import org.openqa.selenium.TakesScreenshot;

9

import org.openqa.selenium.WebDriver;

10

import org.openqa.selenium.firefox.FirefoxDriver;

11

12

public class CaptureScreenshot {

13

WebDriver driver;

14

@Before

15

public void setUp() throws Exception {

16

driver = new FirefoxDriver();

17

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

18

}

19

@After

20

public void tearDown() throws Exception {

21

driver.quit();

22

}

23

24

@Test

25

public void test() throws IOException {

26

// Code to capture the screenshot

27

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

28

// Code to copy the screenshot in the desired location

29

FileUtils.copyFile(scrFile, new File("C:\\CaptureScreenshot\\google.jpg"))

30

}

31

}

Q #38) What is Junit?

Junit is a unit testing framework introduced by Apache. Junit is based on Java.

Q #39) What are Junit annotations?

Following are the JUnit Annotations:

@Test: Annotation lets the system know that the method annotated as @Test is a test method. There can be multiple test methods in a single test script.

@Before: Method annotated as @Before lets the system know that this method shall be executed every time before each of the test methods.

@After: Method annotated as @After lets the system know that this method shall be executed every time after each of the test method.

@BeforeClass: Method annotated as @BeforeClass lets the system know that this method shall be executed once before any of the test methods.

@AfterClass: Method annotated as @AfterClass lets the system know that this method shall be executed once after any of the test methods.

@Ignore: Method annotated as @Ignore lets the system know that this method shall not be executed.

Q #40) What is TestNG and how is it better than Junit?

TestNG is an advanced framework designed in a way to leverage the benefits by both the developers and testers. With the commencement of the frameworks, JUnit gained enormous popularity across the Java applications, Java developers and Java testers with remarkably increasing the code quality. Despite being easy to use and straightforward, JUnit has its own limitations which give rise to the need of bringing TestNG into the picture. TestNG is an open source framework which is distributed under the Apache Software License and is readily available for download.

TestNG with WebDriver provides an efficient and effective test result format that can, in turn, be shared with the stakeholders to have a glimpse on the product’s/application’s health thereby eliminating the drawback of WebDriver’s incapability to generate test reports. TestNG has an inbuilt exception handling mechanism which lets the program to run without terminating unexpectedly.

There are various advantages that make TestNG superior to JUnit. Some of them are:

Added advance and easy annotations

Execution patterns can set

Concurrent execution of test scripts

Test case dependencies can be set

Q #41) How to set test case priority in TestNG?

Setting Priority in TestNG

Code Snippet

1

package TestNG;

2

import org.testng.annotations.\*;

3

public class SettingPriority {

4

@Test(priority=0)

5

public void method1() {

6

}

7

@Test(priority=1)

8

public void method2() {

9

}

10

@Test(priority=2)

11

public void method3() {

12

}

13

}

Test Execution Sequence:

Method1

Method2

Method3

Q #42) What is a framework?

The framework is a constructive blend of various guidelines, coding standards, concepts, processes, practices, project hierarchies, modularity, reporting mechanism, test data injections etc. to pillar automation testing.

Q #43) What are the advantages of the Automation framework?

The advantage of Test Automation framework

Reusability of code

Maximum coverage

Recovery scenario

Low-cost maintenance

Minimal manual intervention

Easy Reporting

Q #44) What are the different types of frameworks?

Below are the different types of frameworks:

Module Based Testing Framework: The framework divides the entire “Application Under Test” into the number of logical and isolated modules. For each module, we create a separate and independent test script. Thus, when these test scripts have taken together builds a larger test script representing more than one module.

Library Architecture Testing Framework: The basic fundamental behind the framework is to determine the common steps and group them into functions under a library and call those functions in the test scripts whenever required.

Data Driven Testing Framework: Data Driven Testing Framework helps the user segregate the test script logic and the test data from each other. It lets the user store the test data into an external database. The data is conventionally stored in “Key-Value” pairs. Thus, the key can be used to access and populate the data within the test scripts.

Keyword Driven Testing Framework: The Keyword Driven testing framework is an extension to Data-driven Testing Framework in a sense that it not only segregates the test data from the scripts, it also keeps the certain set of code belonging to the test script into an external data file.

Hybrid Testing Framework: Hybrid Testing Framework is a combination of more than one above mentioned frameworks. The best thing about such a setup is that it leverages the benefits of all kinds of associated frameworks.

Behavior Driven Development Framework: Behavior Driven Development framework allows automation of functional validations in an easily readable and understandable format to Business Analysts, Developers, Testers, etc.

Q #45) How can I read test data from excels?

Test data can efficiently be read from excel using JXL or POI API. See detailed tutorial here.

Q #46) What is the difference between POI and jxl jar?

#

JXL jar

POI jar

1

JXL supports “.xls” format i.e. binary based format. JXL doesn’t support Excel 2007 and “.xlsx” format i.e. XML based format

POI jar supports all of these formats

2

JXL API was last updated in the year 2009

POI is regularly updated and released

3

The JXL documentation is not as comprehensive as that of POI

POI has a well prepared and highly comprehensive documentation

4

JXL API doesn’t support rich text formatting

POI API supports rich text formatting

5

JXL API is faster than POI API

POI API is slower than JXL API

Overview

HSSF is the POI Project's pure Java implementation of the Excel '97(-2007) file format. XSSF is the POI Project's pure Java implementation of the Excel 2007 OOXML (.xlsx) file format.

HSSF and XSSF provides ways to read spreadsheets create, modify, read and write XLS spreadsheets. They provide:

low level structures for those with special needs

an eventmodel api for efficient read-only access

a full usermodel api for creating, reading and modifying XLS files

FileInputStream fis=new FileInputStream("location of xlsx file");

XSSFWorkbook workbook=new XSSFWorkbook();

int sheets=workbook.getNumberOfSheets();

loop sheets

xssfsheet sheet =workbook.getsheet(I)

keep the row in iterator

Itertor<Row> rows=sheet.Iterator();

Row firstrow=rows.next();

Iterator<Cell> ce=firstrow.celliterator();

while loop ce.hasNext();

Cell value=ce.next

Stale Element Reference Exception

You have probably been directed to this page because you've seen a StaleElementReferenceException in your tests.

Common Causes

A stale element reference exception is thrown in one of two cases, the first being more common than the second:

The element has been deleted entirely.

The element is no longer attached to the DOM.

WebElement element = driver.findElement(By.id("example"));

String text = element.getText();

If "element.getText" returns before the element is removed from the DOM you'll get one result. If, however, the element is removed from the DOM and your code does an automatic lookup for the element again before "element.getText" a different result may be returned.

Should you wish to head down this route, the simplest hook point is to call setElementConverter.

public boolean retryingFindClick(By by) {

boolean result = false;

int attempts = 0;

while(attempts < 2) {

try {

driver.findElement(by).click();

result = true;

break;

} catch(StaleElementException e) {

}

attempts++;

}

return result;

}

What is an Exception?

“An Exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program’s instructions or in simple words, any issue which makes your test case stop in between the execution.”

When exception occurs, the normal flow of program halts & an exception object is created. The program then tries to find someone that can handle the raised exception. The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurred, type of exception etc. The process of creating the exception object and handing it over to run-time environment is called “throwing the exception”.

There is a complete list of Exceptions in Selenium WebDriver mentioned in the Selenium Doc which you may or may not encounter in course of your testing. Hence in this article we will focus on some most common exceptions in Selenium WebDriver,

ElementNotVisibleException: Although an element is present in the DOM, it is not visible (cannot be interacted with). E.g. Hidden Elements – defined in HTML using type=”hidden”.

ElementNotSelectableException: Although an element is present in the DOM, it may be disabled (cannot be clicked/selected).

InvalidSelectorException: Selector used to find an element does not return a WebElement. Say XPath expression is used which is either syntactically invalid or does not select WebElement.

NoSuchElementException: WebDriver is unable to identify the elements during run time, i.e. FindBy method can’t find the element.

NoSuchFrameException: WebDriver is switching to an invalid frame, which is not available.

NoAlertPresentException: WebDriver is switching to an invalid alert, which is not available.

NoSuchWindowException: WebDriver is switching to an invalid window, which is not available.

StaleElementReferenceException: The referenced element is no longer present on the DOM page (reference to an element is now Stale). E.g. The Element belongs to a different frame than the current one OR the user has navigated away to another page.

SessionNotFoundException: The WebDriver is performing the action immediately after ‘quitting’ the browser.

TimeoutException: The command did not complete in enough time. E.g. the element didn’t display in the specified time. Encountered when working with waits.

WebDriverException: The WebDriver is performing the action immediately after ‘closing’ the browser.

How to Handle Exceptions in Selenium WebDriver

Try/Catch: A method catches an exception using a combination of the try and catch keywords. Try is the start of the block and Catch is at the end of try block to handle the exceptions. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

try{

// Some code

}

catch(Exception e){

// Code for Handling the exception

}

Below methods can be used to display Exception information:

printStackTrace(): prints the stack trace , exception name and description.

toString(): returns a text message describing the exception name and description.

getMessage(): displays the description of exception

TestNG Listeners

Pankaj Leave a Comment

TestNG Listeners are used to inspect and modify the testing behavior. TestNG listeners always extend org.testng.ITestNGListener marker interface. TestNG listeners can be defined for a test class using org.testng.annotations.Listeners annotation.

package com.journaldev.listeners;

import org.testng.ITestContext;

import org.testng.ITestListener;

import org.testng.ITestResult;

public class Test3TestListener implements ITestListener{

public Test3TestListener() {

System.out.println("Test3Listener constructor");

}

@Override

public void onTestStart(ITestResult result) {

System.out.println("Test Started. "+result.getStartMillis());

}

@Override

public void onTestSuccess(ITestResult result) {

System.out.println("Test Success. "+result.getEndMillis());

}

@Override

public void onTestFailure(ITestResult result) {

System.out.println("Test Failed. "+result.getTestName());

}

@Override

public void onTestSkipped(ITestResult result) {

System.out.println("Test Skipped. "+result.getTestName());

}

@Override

public void onTestFailedButWithinSuccessPercentage(ITestResult result) {

}

@Override

public void onStart(ITestContext context) {

System.out.println("Context Name = "+context.getName());

}

@Override

public void onFinish(ITestContext context) {

System.out.println(context.getPassedTests());

}

}

Git Commands

A list of my commonly used Git commands

If you are interested in my Git aliases, have a look at my .bash\_profile, found here: https://github.com/joshnh/bash\_profile/blob/master/.bash\_profile

Getting & Creating Projects

Command

Description

git init

Initialize a local Git repository

git clone ssh://git@github.com/[username]/[repository-name].git

Create a local copy of a remote repository

Basic Snapshotting

Command

Description

git status

Check status

git add [file-name.txt]

Add a file to the staging area

git add -A

Add all new and changed files to the staging area

git commit -m "[commit message]"

Commit changes

git rm -r [file-name.txt]

Remove a file (or folder)

Branching & Merging

Command

Description

git branch

List branches (the asterisk denotes the current branch)

git branch -a

List all branches (local and remote)

git branch [branch name]

Create a new branch

git branch -d [branch name]

Delete a branch

git push origin --delete [branch name]

Delete a remote branch

git checkout -b [branch name]

Create a new branch and switch to it

git checkout -b [branch name] origin/[branch name]

Clone a remote branch and switch to it

git checkout [branch name]

Switch to a branch

git checkout -

Switch to the branch last checked out

git checkout -- [file-name.txt]

Discard changes to a file

git merge [branch name]

Merge a branch into the active branch

git merge [source branch] [target branch]

Merge a branch into a target branch

git stash

Stash changes in a dirty working directory

git stash clear

Remove all stashed entries

Sharing & Updating Projects

Command

Description

git push origin [branch name]

Push a branch to your remote repository

git push -u origin [branch name]

Push changes to remote repository (and remember the branch)

git push

Push changes to remote repository (remembered branch)

git push origin --delete [branch name]

Delete a remote branch

git pull

Update local repository to the newest commit

git pull origin [branch name]

Pull changes from remote repository

git remote add origin ssh://git@github.com/[username]/[repository-name].git

Add a remote repository

git remote set-url origin ssh://git@github.com/[username]/[repository-name].git

Set a repository's origin branch to SSH

Inspection & Comparison

Command

Description

git log

View changes

git log --summary

View changes (detailed)

git diff [source branch] [target branch]

Preview changes before merging

Select fruits = new Select(driver.findElement(By.id("fruits")));

fruits.selectByVisibleText("Banana");

fruits.selectByIndex(1);

Select Methods

The following are the most common methods used on drop down list.

Method

Description

selectByVisibleText() and deselectByVisibleText() Example:

Selects/deselects the option that displays the text matching the parameter.

Parameter: The exactly displayed text of a particular option

selectByValue() and deselectByValue() Example:

Selects/deselects the option whose "value" attribute matches the specified parameter.

Parameter: value of the "value" attribute

Remember that not all drop-down options have the same text and "value", like in the example below.

selectByIndex() and deselectByIndex() Example:

Selects/deselects the option at the given index.

Parameter: the index of the option to be selected.

isMultiple() Example:

Returns TRUE if the drop-down element allows multiple selections at a time; FALSE if otherwise.

No parameters needed

deselectAll() Example:

Clears all selected entries. This is only valid when the drop-down element supports multiple selections.

No parameters needed

Why Do We Need Waits In Selenium?

Most of the web applications are developed using Ajax and Javascript. When a page is loaded by the browser the elements which we want to interact with may load at different time intervals.

Not only it makes this difficult to identify the element but also if the element is not located it will throw an "ElementNotVisibleException" exception. Using Waits, we can resolve this problem.

Let's consider a scenario where we have to use both implicit and explicit waits in our test. Assume that implicit wait time is set to 20 seconds and explicit wait time is set to 10 seconds.

Suppose we are trying to find an element which has some "ExpectedConditions "(Explicit Wait), If the element is not located within the time frame defined by the Explicit wait(10 Seconds), It will use the time frame defined by implicit wait(20 seconds) before throwing an "ElementNotVisibleException".

Selenium Web Driver Waits

Implicit Wait

Explicit Wait

Implicit Wait

Selenium Web Driver has borrowed the idea of implicit waits from Watir.

The implicit wait will tell to the web driver to wait for certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, web driver will wait for that time before throwing an exception.

In the below example we have declared an implicit wait with the time frame of 10 seconds. It means that if the element is not located on the web page within that time frame, it will throw an exception.

To declare implicit wait:

Syntax:

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

System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );

driver = new ChromeDriver();

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

String eTitle = "Demo Guru99 Page";

Explanation of Code

In the above example,

Consider Following Code:

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

Implicit wait will accept 2 parameters, the first parameter will accept the time as an integer value and the second parameter will accept the time measurement in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS, etc.

Explicit Wait

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);

driver = new ChromeDriver();

WebDriverWait wait=new WebDriverWait(driver, 20);

String eTitle = "Demo Guru99 Page";

Explanation of Code

Consider Following Code:

WebElement guru99seleniumlink;

guru99seleniumlink = wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i")));

guru99seleniumlink.click();

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

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()

Fluent Wait

The fluent wait is used to tell the web driver to wait for a condition, as well as the frequency with which we want to check the condition before throwing an "ElementNotVisibleException" exception.

Frequency: Setting up a repeat cycle with the time frame to verify/check the condition at the regular interval of time

Let's consider a scenario where an element is loaded at different intervals of time. The element might load within 10 seconds, 20 seconds or even more then that if we declare an explicit wait of 20 seconds. It will wait till the specified time before throwing an exception. In such scenarios, the fluent wait is the ideal wait to use as this will try to find the element at different frequency until it finds it or the final timer runs out.

Syntax:

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(timeout, SECONDS)

.pollingEvery(timeout, SECONDS)

.ignoring(Exception.class);

Explanation of Code

Consider Following Code:

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

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

In the above example, we are declaring a fluent wait with the timeout of 30 seconds and the frequency is set to 5 seconds by ignoring "NoSuchElementException"

Consider Following Code:

public WebElement apply(WebDriver driver) {

return driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i"));

Difference between Implicit Wait Vs Explicit Wait

Implicit Wait

Explicit Wait

Implicit Wait time is applied to all the elements in the script

Explicit Wait time is applied only to those elements which are intended by us

In Implicit Wait, we need not specify "ExpectedConditions" on the element to be located

In Explicit Wait, we need to specify "ExpectedConditions" on the element to be located

It is recommended to use when the elements are located with the time frame specified in implicit wait

It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected)

Conclusion:

Implicit, Explicit and Fluent Wait are the different waits used in selenium. Usage of these waits are totally based on the elements which are loaded at different intervals of time. It is always not recommended to use Thread.Sleep() while Testing our application or building our framework.

How to access web table element in selenium web driver

There are 2 types of HTML table possible in the web.

Static web table.

Dynamic web table.

Static web table: Normally static tables are static in nature, means a number of column and row is fixed, it will never increase in runtime and data inside the each cell is also fixed.

Dynamic web table: It is just opposite of static web table, here row, column and cell data could be not fixed or we can say dynamic. It will change at runtime.

public static void main(String argc[]){

System.setProperty("webdriver.chrome.driver","D:/workspace/TestProject/src/chromedriver.exe");

driver = new ChromeDriver() ;

driver.get("https://www.w3schools.com/html/html\_tables.asp");

int rowCount = driver.findElements(By.xpath("//\*[@id='customers']/child::\*/child::tr")).size();

for (int i=1;i<=rowCount;i++){

int columnCount = driver.findElements(By.xpath("//\*[@id='customers']/child::\*/child::tr["+i+"]/child::td")).size();

for(int j=1;j<=columnCount;j++){

WebElement element = driver.findElement(By.xpath("//\*[@id='customers']/child::\*/child::tr["+i+"]/child::td["+j+"]"));

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

}

System.out.println();

}

driver.quit();

}

We need to follow few steps to retrieve data from web table

We need to uniquely identify table object.

Count number of rows present in the table.

Count number of column present in each row.

Get the value from each cell.

Step 1. We need to uniquely identify table object

Now we can start with the below table. We need to identify the web table uniquely.

Here is the list of annotations that TestNG supports −

Sr.No.

Annotation & Description

1

@BeforeSuite

The annotated method will be run only once before all tests in this suite have run.

2

@AfterSuite

The annotated method will be run only once after all tests in this suite have run.

3

@BeforeClass

The annotated method will be run only once before the first test method in the current class is invoked.

4

@AfterClass

The annotated method will be run only once after all the test methods in the current class have run.

5

@BeforeTest

The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.

6

@AfterTest

The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run.

7

@BeforeGroups

The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

8

@AfterGroups

The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

9

@BeforeMethod

The annotated method will be run before each test method.

10

@AfterMethod

The annotated method will be run after each test method.

11

@DataProvider

Marks a method as supplying data for a test method. The annotated method must return an Object[ ][ ], where each Object[ ] can be assigned the parameter list of the test method. The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation.

12

@Factory

Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[ ].

13

@Listeners

Defines listeners on a test class.

14

@Parameters

Describes how to pass parameters to a @Test method.

15

@Test

Marks a class or a method as a part of the test.