Rex
Jones II

Selenium
Wait Methods

Table of Contents

Rex
Jones II

Introduction

Hello everybody, Welcome To Selenium 4 Beginners. My name is Rex Allen Jones II. In this video, I will cover the Wait Methods for Selenium. The Browser Methods, WebElement Methods, and Navigation Methods were covered in previous videos.

Number (4). The Wait Methods are a group of methods that pause between execution statements. We wait for a statement to finish before moving to the next statement. A question someone may ask is "Why, do we want our Test Scripts to pause between statements?" We want our Test Scripts to pause between statements because of dynamic content on a web page.
An example of dynamic content is a web page that has elements that you do not see when first opening the web page. Those elements wait a few seconds then show up. Therefore, we need our Test Script to also wait a few seconds then perform an action after the element shows up. Wait complications are one of the reasons we have errors with our Selenium Automation Test Scripts.

In this tutorial, we are going to discuss why we have Wait Complications, the different Type of Wait Methods to handle those wait complications, then demo all of the Selenium Wait Methods. Let's start with the Wait Complications.

Wait Complications

Wait Complications are outside components that cause problems with our Test Scripts. Some of those outside components are Server Location, Server Performance, Machine Performance, and JavaScript Engine Performance. Network is not in this list but it's another outside component that cause wait problems.

Server Locations cause wait problems because the information has to travel. Information travels to the server location then it travels all the way back from the server location. When it comes to Server Performance, our browser sends a ping to the server asking for all of the information so the website can load. If our server's performance is slow then it will take a long time to respond.

Machine performance is different with fast machines and slow machines. A fast machine will render elements on a web page quick. On the other hand, a slow machine with small memory will run our Test Scripts slower than a fast machine.

JavaScript Engines are embedded in browsers and web servers. Therefore, the performance is important when testing web applications that have a lot of JavaScript. Something rendered on one browser may load faster than another browser.

Here's 2 screenshots that compare browsers for a JavaScript benchmark test using JetStream. The URL is https://browserbench.org/JetStream/ I ran a test on Firefox and Chrome. Firefox has a score of 159 while Chrome has a score of 137. To be honest, I ran a test on Internet Explorer 2 times and no score was returned. The bigger score is better. In this case, we see Firefox outperformed Chrome. In addition to the browser, the version of our browser dictates how fast or slow Test Scripts execute.
Next is the different type of Wait Methods.

Wait Method Types

By default, Selenium executes statements very fast. Soon as one statement is finished, Selenium immediately executes the next statement. That can be a problem because Selenium will not wait for a dynamic element.

Therefore, we need to instruct Selenium to wait for dynamic elements before executing the next statement. If Selenium does not wait, then an exception shows up. The exception shows up because our Test Script is attempting to interact with an element that could have changed or may not be available. We have 5 Selenium Wait Method Types to handle dynamic elements.

Thread.sleep() is not a Selenium Wait Method although it causes execution to pause or sleep for a certain number of milliseconds. I added it to this list because many automation engineers implement Thread.sleep() as a wait method. However, it is not a wait method but it is a sleep method. Thread.sleep() is not efficient or reliable. It is not efficient because it will stay sleep for the time we set even if our statement is finished and ready to move to the next statement.

It is not reliable if we consider a Wait Complication such as Machine Performance where one machine could possibly run slower than another machine. For example, if we add a hard-coded value of 5000 milliseconds which is 5 seconds to Thread.sleep(). A faster machine may render the elements in 4 seconds but a slower machine may render the same elements in 6 seconds. As a result, the same Test Script will pass on 1 machine and fail on another machine.

Page Load Timeout sets the wait time for a page to load. Set Script Timeout sets the wait time for JavaScript to execute. Implicit Wait determines the wait time to search for an element. Explicit Wait pauses execution until time has expired or an expected condition is met. Fluent Wait is a specialization of Explicit Wait which also pauses until time has expired or an expected condition is met.
Page Load Timeout, Set Script Timeout, and Implicit Wait are 3 built-in Selenium Wait Methods. All 3 methods are configured using the Timeouts() interface.

Now, let’s demo the Selenium Wait Methods.

Demo

Page Load Time Out

Our 1st Selenium Wait Method is Page Load Time Out. Page Load Time Out sets the wait time for a page to load. driver.manage().timeouts().im. In this intellisense, we see all 3 built-in Selenium Wait Methods. Implicit Wait, Page Load Timeout, and Set Script Timeout. Select Page Load Timeout, enter 3 for time, and TimeUnit dot. Time Unit represents different lengths of time such as DAYS, HOURS, and MICROSECONDS. Let’s select MILLISECONDS.

Next, load the OrangeHRM page driver.get(“https://opensource-demo.orangehrmlive.com”), print the title, sysout “What Is The Page Title? ” + driver.getTitle() and Run. As expected, our Test Script returned a TimeOutException because the page did not load in 3 MilliSeconds. This exception provides a good example of what happens if our page does not load in the max amount of time.

Let’s Imagine if we have a requirement that states we need the page to load in 3 Seconds. If the page does not load in 3 seconds then we must report a defect. Change our time from MilliSeconds to Seconds.

Selenium will wait a max of 3 Seconds for the Orange HRM page to load. If the page loads before 3 seconds then execute the next statement which is print title. Run again. Now the page loads without a problem. What Is The Page Title? OrangeHRM. The entire Test Script executed less than 1 second – 0.959.

Recall from our Introduction, that Thread.sleep() is not a wait method but it is a sleep method. I’m going to show you how it’s a sleep method. Thread.sleep and 3000 milliseconds. Add a throws declaration then import InterruptedException. Run again.

We see the same result. OrangeHRM is the page title. However, look and see how long it took for this Test Script to execute. 4 Seconds. Thread.sleep() waited for the whole 3 seconds although the page had already loaded and was ready to execute the print statement. This is one of the reasons why we should not use Thread.sleep as a Wait method. It’s not dynamic like pageLoadTimeout. That’s it for Selenium Wait Method pageLoadTimeout.

Implicit Wait

Next is Implicit Wait. Let’s walkthrough this Test Script for an AJAX web page. AJAX stands for Asynchronous JavaScript and XML. First, we load the web page which is Welcome To The Internet. This web page is Powered by Elemental Selenium where you get an email once a week to use Selenium like a Pro by Dave Haeffner.

Next, we are going to click the Dynamic Loading hyperlink. After clicking Dynamic Loading, the next step is to click Example 2. The last step is to click the Start button. Do you see how we are waiting for the dynamic element to load?

Finally, in our Test Script, we are going to find Hello World, get the text of Hello World, then print Hello World. Let’s run our test.

Uh oh – NoSuchElementException. Unable to locate element. Which element? The string Hello World. Our Test Script executed so fast that it did not wait for the next step which is find Hello World. That’s why Selenium provides Wait Methods so our execution can pause between these types of statements. If not, our Test Script will fail every time.

Let’s use Implicit Wait. driver.manage().timeouts().implicitlyWait(). The description states Specifies the amount of time the driver should wait when searching for an element if it is not immediately present. This means Selenium will for all instances of driver. Enter 5 and TimeUnit is Seconds. Selenium will wait up to 5 seconds. Let’s Run. Now, we see Hello World! and Pass for our Test Script.

Implicit Wait would have waited for Lines 44, 45, 46, and 48 because they have a driver instance. If the element was not found for those lines then an exception would have been thrown. That’s it for Implicit Wait.

Rex
Jones II

Explicit Wait

Next is Explicit Wait. This is the same Test Script from Implicit Wait. Where we load the page, click the Dynamic Loading hyperlink, click Example 2, then click the Start button. Followed by finding Hello World, getting the text of Hello World, and printing Hello World. I'm going to run this Test Script again. It's going to fail because of this dynamic Hello World element. Fail

Explicit Wait is used on 1 WebElement at a time. For that 1 element which is Hello World, execution will pause until time has expired or an expected condition is met using the WebDriverWait class.
Let's start by writing WebDriverWait and wait as the object reference = new WebDriverWait() pass in driver and TimeOutInSeconds as 5 for the parameters. 5 represents the maximum number of seconds Selenium will wait for an element before throwing an exception.

Here's the power of Explicit Wait. After clicking the Start button, we write wait dot until ExpectedConditions with an s dot. So far, we are going to wait until an expected condition is met. There's many Expected Conditions. Our goal is to check if Hello World is present. We can select visibilityOfElementLocated which is an expectation for checking that an element is present. Locator will be By.xpath(). Go back to the application, Inspect Hello World, Copy xpath, Paste xpath.
Let's run. Hello World is printed to the Console. All of the ExpectedConditions are examples that assist us with writing our own customized Explicit Wait statements.

Let's breakdown Explicit Wait. With the first line, we have the WebDriverWait class which has a Constructor of WebDriverWait and 2 parameters. driver and 5. Let's view JavaDocs for the WebDriverWait class. It has 3 Constructors. 1, 2, and 3. We used the 2nd Constructor which ignore instances of NotFoundException.

NotFoundException is an exception that is thrown when an element is not found. We see both parameters: driver and timeOutInSeconds. driver is the WebDriver instance while timeOutInSeconds is the timeout in seconds when an expectation is called.

The next line is wait until ExpectedConditions. wait is the object reference of the WebDriverWait class. until is a method that repeats until one of the following occurs. Number (1) the function returns neither null nor false. Number (2) the function throws an unignored exception. Number (3) the timeout expires. Number (4) the current thread is interrupted. Since Hello World was returned, Number(1) applies to our Test Script because Hello World is not null nor false.

ExpectedConditions is a class that has so many methods. However, we used visibilityofElementLocated and the By locator was xpath. That's it for Explicit Wait.

(Recap) Implicit Wait vs Explicit Wait

Let's recap and discuss Implicit Wait versus Explicit Wait. These are the 2 most popular Selenium Wait Method Types. Implicit Wait uses a single code line while Explicit Wait uses Multiple Code Lines. Implicit Wait is used on all WebElements. That one code line for Implicit Wait impacts all WebElements in that Test Script. However, Explicit Wait sets a wait time that is used on one WebElement. Expected Conditions are not required for Implicit Wait but Expected Conditions are required for Explicit Wait.

Here’s a list of 16 Expected Conditions. This is not the complete list. All of them would not fit in this table. I think it’s nearly 20 Expected Conditions that are missing. If we choose to, we can create our own customized Explicit Wait using the ExpectedConditions class.

Now, here’s the controversy between Implicit Wait and Explicit Wait. Which one should we use? It’s recommended we use Explicit Wait over Implicit Wait. We must handle the slow loading elements like Hello World on an element-by-element basis. According to Mastering Selenium WebDriver by Mark Collin, Implicit Waits were not originally a part of the WebDriver API. They are a hangover from the old Selenium 1 API. Implicit Wait was not going to be added to Selenium WebDriver API but people was used to it and wanted it back. There’s at least 2 reasons for choosing Explicit Wait rather than Implicit Wait. The 1st reason is Implicit Wait can slow down our test and the 2nd reason is Implicit Wait can break Explicit Wait.

Fluent Wait

Next is Fluent Wait. WebDriverWait extends FluentWait which pauses execution until time has expired or an expected condition is met. WebDriverWait is a specialization of FluentWait that uses WebDriver instances. Therefore, at the core of Explicit Wait is FluentWait because Explicit Wait implements WebDriverWait. Here’s a sample of Fluent Wait that I will use in this demo.

Go to Tools QA and take a look at our (AUT) Application Under Test. We will test a Countdown timer that now shows Buzz Buzz. The timer starts at 40, countdown to zero then show Buzz Buzz.
Load Tools QA driver.get Paste the URL ("http://toolsqa.com/automation-practice-switch-windows");
FluentWait (CTRL+SPACE) Copy and paste the sample.

Let’s take care of the red X’s and lines. Import Wait, Fluent Wait, and NoSuchElementException. There’s 2 NoSuchElementException packages but we will select the package from org.openqa.selenium. Import Function by selecting the package from com.google.common.base. Modify the SECONDS to include TimeUnit.

We see the first comment states waiting 30 seconds for an element to be present on the page. That comment is referring to the withTimeout method which sets how long to wait for the evaluated condition to be true. The next comment states checking for it presence once every 5 seconds. This comment refers to the pollingEvery method which sets how often the condition should be evaluated. The ignoring method will ignore the NoSuchElementException. We see parameters show exception to ignore.

I will change the pollingEvery method to 1 second. This Test Script will fail because the Countdown timer on Tools QA starts at 40 and the withTimeout method remains at 30. In our demos, I do not want to always show you the Happy Path scenarios. How our Test Script looks when it pass but also show the failures. This will return an exception.

Rex
Jones II

Let’s modify the sample code. First, I will change foo to element. Next, we must find the element which is the Countdown timer. Therefore, I will write driver.findElement(By.xpath(“”)); then go back to our AUT Tools QA and get the xpath’s value. Inspect the element. Copy xpath and paste xpath. Assign the value to WebElement timer.

After finding Buzz Buzz, we want to get the text of Buzz Buzz so let’s write timer.getText then assign the value to String messageTimer. The last modification is to write an if then else statement. You can watch video number (15) Java’s Conditional if Statements to get a better understanding of all 3 if statements. Within this if statement, we are going to print Buzz Buzz. If (messageTimer.equals(Buzz Buzz)) then print (“The Sound Of An Opportunity Clock Is ” + messageTimer) else print (messageTimer).

Here’s a breakdown of this if statement. If the Message Timer equals Buzz Buzz then print The Sound Of An Opportunity Clock Is Buzz Buzz. If it does not equals Buzz Buzz then print whatever the message shows on Tools QA after the clock reaches 40 seconds.

Do you see the red X and how our return type is WebElement? That means, we must return a WebElement. The error message shows this method must return a result of type WebElement.
In this case, our WebElement is timer so we write return timer after printing Buzz Buzz and write return null if the Message Timer is not Buzz Buzz. Now, the error goes away. You can watch video number (9) Relationship Between Java’s Classes, Objects, and Methods to get more information about return types. We are finished modifying the sample code.

This Test Script will wait until the timeout expires or the condition becomes true. What is the condition? The condition is when Buzz Buzz shows up on the page. Our code will poll every second which means it will check the condition every second. The NoSuchElementException will get ignored. We can add more exceptions to ignore if we choose to.

The purpose of this function is to find Buzz Buzz and return Buzz Buzz. We have 2 arguments: WebDriver and WebElement. WebDriver is the parameter for the apply method. WebElement is the return type for the apply method. WebDriver is the input while WebElement is the output.

driver is the input parameter which helps find an element. The element is assigned to WebElement timer. We get the text then assign the value to messageTimer. When messageTimer equals Buzz Buzz, the value is printed and returned as an output.

Now, let’s Run demoFluentWait. We see the polling for every second starting at 40 then counting down. This Test Script should fail when the Countdown timer reaches 10 because we are waiting 30 seconds. Yes, the TimeoutException showed up. Expected Condition failed. Tried for 30 seconds with 1 second interval.

This time, the Test Script will pass when I change 30 to 41. I will also change the comments. 30 to 41 and 5 to 1. Run again. The Sound Of An Opportunity Clock Is Buzz Buzz. An advantage of using Fluent Wait with a Function element is the flexibility of passing any input object type and returning any object type. Object Type – Input. Object Type – Returned. In this scenario, we passed WebDriver and returned WebElement. That’s it for Fluent Wait. Thank You for watching Selenium Wait Methods.

Rex
Jones II

Download Documents

You can download your Transcript, Presentation and Automation Code at https://tinyurl.com/Selenium-Wait-Methods