Synchronization: Waits in Selenium

Value of Automation testing lies in execution. You automate a test case because you want to execute it multiple times saving your time and manual efforts. But sometimes your test cases failed due to network speed, due to slow rendering of Ajax and JavaScript etc. We need to overcome these challenges in our automated scripts.

Suppose, you entered username and password, you clicked on submit button and now you will be navigated to home page. But, here comes the challenge. If you do not synchronize your script, your script will not wait for the next page and you will get NoSuchElementException. WebDriver provides a good to use wait mechanism to solve these problems.

There are 3 types of waits that Selenium provide:

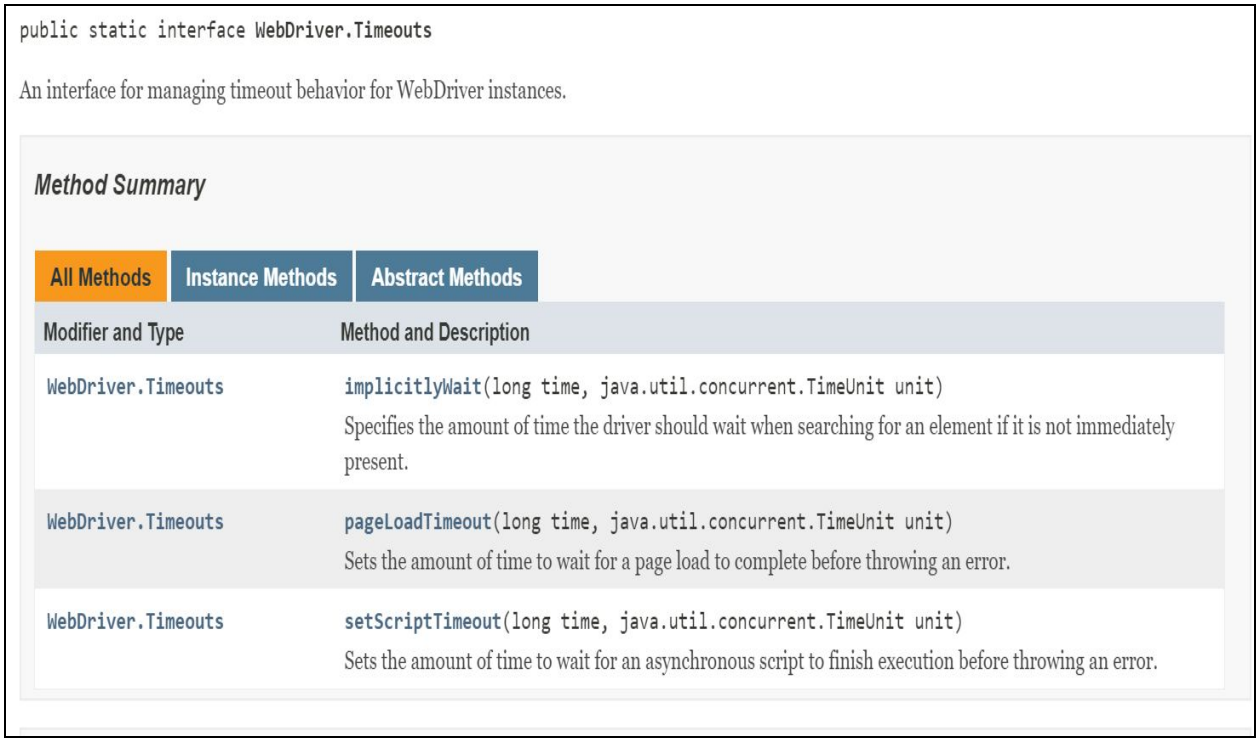
1. Implicit Wait
2. Explicit Wait
3. Fluent Wait

So, let's start to know more about them.

Implicit Wait:

Implicit wait comes from Timeouts interface which is an inner interface of WebDriver interface. Timeouts interface has three abstract methods implicitlyWait, pageLoadTimeout, setScriptTimeout. All of these methods are similar in terms of arguments but their functionality is different. pageLoadTimeout will wait for any page to be loaded as per specified time. Similarly setScriptTimeout will for an script to finish execution before throwing error.

Implicit wait is dynamic in nature. For example, if you have defined a wait for 10 seconds, but your element is located in 5 seconds then your script will move on to next line of code ignoring remaining 5 seconds.



public static interface WebDriver.Timeouts

An interface for managing timeout behavior for WebDriver instances.

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
WebDriver.Timeouts	implicitlyWait(long time, java.util.concurrent.TimeUnit unit) Specifies the amount of time the driver should wait when searching for an element if it is not immediately present.
WebDriver.Timeouts	pageLoadTimeout(long time, java.util.concurrent.TimeUnit unit) Sets the amount of time to wait for a page load to complete before throwing an error.
WebDriver.Timeouts	setScriptTimeout(long time, java.util.concurrent.TimeUnit unit) Sets the amount of time to wait for an asynchronous script to finish execution before throwing an error.

ImplicitlyWait method takes two arguments one is the wait time that tells WebDriver to wait for a certain amount of time before throwing NoSuchElementException and second is the time unit whether its microseconds, milliseconds, seconds, minutes, hours or days. In below example, we are telling WebDriver to wait for 10 seconds.

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

The default time setting is 0.

ImplicitWait is a global wait. That means it is defined just after your WebDriver initialization and tied to WebDriver throughout the driver instance till it life ends. As I said, it is a global wait so it will wait for specified time for every element in the script to be displayed. So, if your script is failed at some test step and you do not have a proper exit mechanism there then implicit wait will wait for each element in your script and it will certainly increase the script execution time.

As a result of this, there is another wait called Explicit Wait which is specific to the element you have decided to wait for.

Explicit Wait:

Explicit wait is not global in nature. They are applicable only to those web elements that are specified by the user. Unlike implicit wait, here we can specify time in only seconds by which our WebDriver will wait for the element.
Explicit Wait comes from *WebDriverWait* class which implements *Wait* interface and is a subclass of *FluentWait*.

Explicit Wait is a better option to handle Ajax and JavaScript components that loads dynamically. We can declare explicit wait duration like below where 10 is the wait seconds to wait for.

```
WebDriverWait wait = new WebDriverWait(driver,10);
```

Once we declare explicit wait, we also have to specify an expected condition for an element to wait for like below where locator is id, name, xpath etc of your element.

```
wait.until(ExpectedConditions.elementToBeClickable(locator));
```

Some of the commonly used expected conditions are:

1. **alertIsPresent()**
2. **elementToBeClickable()**
3. **frameToBeAvaliableAndSwitchToIt()**
4. **presenceOfElementLocated()**
5. **visibilityOfElementLocated()**

I would advise you to design some generic methods in your framework with these expected conditions and then invoke them for elements where ever you must wait for.

```
public void waitForElementToBeDisplayed(WebElement element, long timeout) {
            WebDriverWait wait = new WebDriverWait(driver,timeout);
            wait.until(ExpectedConditions.visibilityOf(element));
            }

public void waitForElementToBeClickable(WebElement element, long timeout) {
            WebDriverWait wait = new WebDriverWait(driver,timeout);
            wait.until(ExpectedConditions.elementToBeClickable(element));
}
```

Explicit Wait by default keeps checking the Expected Condition every 500 milliseconds until it locates the element. It throws timeout exception if element is not located by the specified wait time duration. It is also dynamic in nature. For example, if you have specified wait time for 10 seconds, but your element is located within 5 seconds then WebDriver will move to next line of code by ignoring remaining seconds.

Fluent Wait:

Fluent wait is a class that implements wait interface. Fluent Wait is like Explicit Wait that it also waits for a specified Expected Condition for a specified time duration. But it is much more advanced. You can use pollingEvery() method to specify the frequency with which Fluent wait must set up a repeat cycle to check expected condition, you can use ignoring method to ignore an exception like NoSuchElementException.

Fluent Wait is generally used for Elements that are located at different interval of time. Sometimes this element is located in 5 seconds, sometimes in 10 seconds, Sometimes in 18 seconds or even more than that. Now if you specify an explicit wait for 15 seconds, it will throw a timeout exception. Fluent Wait can use pollingEvery() method to verify element with a defined frequency within the defined time frame.

Syntax:

```
Wait wait = new FluentWait(driver).withTimeout(Duration.ofSeconds(10))
            .pollingEvery(Duration.ofSeconds(2)
             .ignoring(NoSuchElementException.class);
```

Where 10 second is the total wait time and 2 second is the frequency by which Fluent Wait keep checking whether element is loaded or not.

Now using this wait reference you can define an expected condition like below where locator is id, name, xpath, css etc of your element and then you can perform relevant operation like click, sendKeys etc on the element:

```
WebElement element = wait.until(new Function<WebDriver, WebElement>()
             {
               public WebElement apply(WebDriver driver) {
               return driver.findElement(locator);
             }
             });

element.click();
```

So, in a nutshell here is how fluent wait works:

1. *It defines a maximum time to wait along with a polling time to keep checking the element.*
2. *Fluent Wait then checks for the expected condition defined in until()*
3. *If the condition fails then it ignores the exception and waits and again check as per the time defined in pollingEvery method.*
4. *If it reaches maximum time to wait and still expected condition is not met then it throws the exception.*

I would recommend you to through official document of Fluent Wait class here to get understanding of other fluent wait methods like withMessage(), IgnoreAll() etc.

In my framework, I have implemented a generic method of fluent wait to fetch element instead of using WebDriver's findElement method. The benefit is instead of defining conditions as per the element, it waits for element throughout the driver's life cycle. It's pollingEvery method check every 2 seconds and the execution is continued if element is found.

```
public WebElement fluentWait(final By locator) {
    Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
            .withTimeout(Duration.ofSeconds(30))
            .pollingEvery(Duration.ofSeconds(5))
            .ignoring(NoSuchElementException.class);


    WebElement element = wait.until(new Function<WebDriver, WebElement>() {
        public WebElement apply(WebDriver driver) {
            return driver.findElement(locator);
        }
    });


    return  element;
};
```

So Instead of driver.findElement(By.id(“my id”)) or driver.findElement(By.xpath(“my xpath”)) etc, I simply use fluentWait(By.id(“my id”)) to fetch an element. Something like this:

WebElement element = fluentWait(By.xpath(“my xpath”);

And then I perform operation on that element.

element.sendKeys();

So it is that simple to leverage functionality of fluent wait in your framework.

Summary:

You should never use Thread.sleep() in your code as it is the static wait and considered a bad coding practice. You should always opt for any of the above-mentioned waits considering elements and their loading time. Advantages of explicit wait over implicit wait is that you can also set an expected condition while implicit wait only checks visibility of an element, explicit wait can set a condition like whether element is also clickable or not, whether it is selectable or not.

So, this was all about waits in Selenium WebDriver. I will be back with more knowledge sharing articles. Happy reading! 😊