softech solutions

Selenium – Wait Commands

Lecture Notes

A Complete Guide for Beginners

Softech Solutions Inc.

www.softechsolutionsgroup.com

saney.alam@softechsolutionsgroup.com

Selenium – Wait Commands

Listing out the different WebDriver **Wait** statements that can be useful for an effective scripting and can avoid using the **Thread.sleep()** commands.

ImplicitlyWait Command

Purpose: Selenium WebDriver has borrowed the idea of ***implicit waits*** from **Watir**. This means that we can tell Selenium that we would like it to wait for a certain amount of time before throwing an **exception** that it cannot find the element on the page. We should note that implicit waits will be in place for the entire time the browser is open. This means that any search for elements on the page could take the time the implicit wait is set for.

```
WebDriver driver => new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://url_that_delays_loading");
WebElement myDynamicElement =
driver.findElement(By.id("myDynamicElement"));
```

FluentWait Command

Purpose: Each ***FluentWait*** instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. Furthermore, the user may configure the wait to ignore specific types of exceptions whilst waiting, such as ***NoSuchElementExceptions*** when searching for an element on the page.

```
// Waiting 30 seconds for an element to be present on the page,
checking for its presence once every 5 seconds.

Wait wait = new FluentWait(driver)
            .withTimeout(30, SECONDS)
            .pollingEvery(5, SECONDS)
            .ignoring(NoSuchElementException.class);

WebElement foo = wait.until(new Function() {
      public WebElement apply(WebDriver driver) {
            return driver.findElement(By.id("foo"));
      }
   });
```

ExpectedConditions Command

Purpose: Models a condition that might reasonably be expected to eventually evaluate to something that is neither null nor false.

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

WebElement element =
wait.until(ExpectedConditions.elementToBeClickable(By.id(>someid>)));
```

PageLoadTimeout Command

Purpose: Sets the amount of time to wait for a page load to complete before throwing an error. If the timeout is negative, page loads can be indefinite.

```
driver.manage().timeouts().pageLoadTimeout(100, SECONDS);
```

SetScriptTimeout Command

Purpose: Sets the amount of time to wait for an asynchronous script to finish execution before throwing an error. If the timeout is negative, then the script will be allowed to run indefinitely.

```
driver.manage().timeouts().setScriptTimeout(100,SECONDS);
```

Sleep Command

Purpose: This is rarely used, as it always force the browser to wait for a specific time. ***Thread.Sleep*** is never a good idea and that's why Selenium provides wait primitives. If you use them you can specify much higher timeout value which makes tests more reliable without slowing them down as the condition can be evaluated as often as it's required.

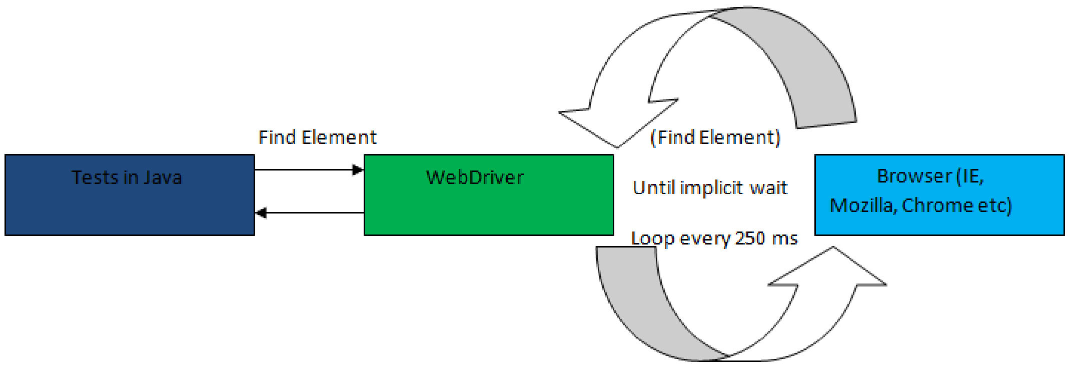
```
thread.sleep(1000);
```

Advanced WebDriver Wait Commands

In this section we will explore more on the **Fluent Waits** and see how we can create our own **Custom Waits or Advance WebDriver Waits**. A fluent wait looks like this:

```
//Declare and initialise a fluent wait
      FluentWait wait = new FluentWait(driver);

//Specify the timout of the wait
      wait.withTimeout(5000, TimeUnit.MILLISECONDS);

//Sepcify polling time
      wait.pollingEvery(250, TimeUnit.MILLISECONDS);

//Specify what exceptions to ignore
      wait.ignoring(NoSuchElementException.class)

/*This is how we specify the condition to wait on.This is what we
will explore more in this chapter */
      wait.until(ExpectedConditions.alertIsPresent());
```

Fluent waits are also called smart waits also. They are called smart primarily because they don't wait for the max time out, specified in the ***.withTimeout(5000, TimeUnit.MILLISECONDS)***. Instead it waits for the time till the condition specified in .until(YourCondition) method becomes true.
A flow diagram explaining the working of Fluent wait is explained in the below diagram.



When the until method is called, following things happen in strictly this sequence

- ***Step 1****: In this step fluent wait captures the wait start time.*
- ***Step 2****: Fluent wait checks the condition that is mentioned in the* **`.until()`** *method*
- ***Step 3****: If the condition is not met, a thread sleep is applied with time out of the value mentioned in* the **`.pollingEvery(250, TimeUnit.MILLISECONDS)`** *method call. In the example above it is of 250 milliseconds.*
- ***Step 4****: Once the thread sleep of step 3 expires, a check of start time is made with the current time. If the difference between wait start time, as captured in step 1, and the current time is less than time specified in* **`.withTimeout(5000, TimeUnit.MILLISECONDS)`** *then step 2 is repeated.*

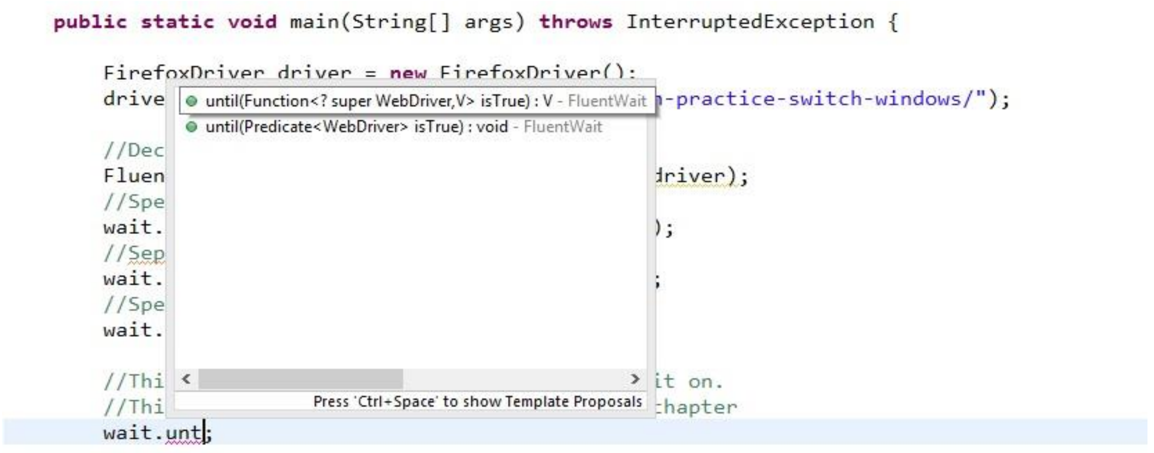
This process keeps on happening till the time either the time out expires or the condition comes out to be true.

In this example we have used ***`ExpectedConditions`*** class. This is quite a handy class and will suite most of your needs. However, at times we might need more than what is present in ***`ExpectedConditions`***. If you pay close attention to the ***`.until()`*** method you will realize that is an overloaded function which can take two types of parameters.

- *A Function()*
- *A Predicate()*

Let's take a look at one such example. We will take a simple scenario which is present on this page http://softech.wpengine.com/automation-practice-switch-windows/

Problem: On this page there is a button "Change Color" this button turns red an after a few seconds. Let's write a simple wait which can wait for the color of the button to change. Once the color has changed, click on the button.
Before we start let's just take a look at what is signature of ***`.until()`*** method overload.



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

    FirefoxDriver driver = new FirefoxDriver();
    drive                                        -practice-switch-windows/");

    //Dec
    Fluen                                        driver);
    //Spe
    wait.                                        );
    //Sep
    wait.                                        ;
    //Spe
    wait.

    //Thi                                        it on.
    //Thi                                        chapter
    wait.unt;
```

until(Function<? super WebDriver,V> isTrue) : V - FluentWait
until(Predicate<WebDriver> isTrue) : void - FluentWait

Pay attention that ***.until()*** method can accept a ***Function< ? super WebDriver, V>*** or a ***Predicate***;

What is a Function?

Package: ***com.google.common.base.Function***
A Function here is a generic interface which asks you to implement following methods

- ***apply(F from)***
- ***equals(Object obj)***

Implementation of equals can be ignored and it will picked from the base implementation.

How to define one?

```
      Function<WebDriver, Boolean> function = new Function<WebDriver, Boolean>()
      {
            public Boolean apply(WebDriver arg0) {
                  return null;
            }
      };
```

Pay attention that we have to define the apply method ourselves. Now the important point to note here is the signature of the apply method. Method will accept a WebDriver as the input argument and will return a Boolean value. This is because the way we have defined the Function (Function). First argument mentioned will be the input argument and the second will be the return value of the apply method.

Now implement the apply method so that it returns false in case of condition failure and return true in case of condition being met. In our case we will return false if the color of the button is not red else we will return true. Here is the code sample

```
Function<WebDriver, Boolean> function = new Function<WebDriver, Boolean>()
{
      public Boolean apply(WebDriver arg0) {
            WebElement element = arg0.findElement(By.id("colorVar"));
            String color = element.getAttribute("color");

            if(color.equals("red"))
            {
                  return true;
            }
            return false;
      }
};
```

Now that we have the wait condition we can simple apply the wait condition in the ***`FluentWait`***. Like this, notice that I have added some print statements to see what is happening in the code

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

       WebDriver driver = new FirefoxDriver();
       driver.get("http://softechsolutionsgroup.com/automation-
practice/");

       FluentWait<WebDriver> wait = new FluentWait<WebDriver>(driver);
       wait.pollingEvery(250,  TimeUnit.MILLISECONDS);
       wait.withTimeout(2, TimeUnit.SECONDS);

       Function<WebDriver, Boolean> function = new Function<WebDriver,
Boolean>()
       {
              public Boolean apply(WebDriver arg0) {
                     WebElement element =
                     arg0.findElement(By.id("colorVar"));
                     String color = element.getAttribute("color");
                     System.out.println("The color if the button is " +
                     color);
                     if(color.equals("red"))
                     {
                            return true;
                     }
                     return false;
              }
       };

       wait.until(function);
}
```

This is the output that you get on the console window:

The color of the button is color: white;
The color of the button is color: white;
The color of the button is color: white;
The color of the button is color: white;
The color of the button is color: white;
The color of the button is color: red;

See that the function tests for the color property of the control for around 5 times and in the last attempt it is able to find the color property to be red. This is when the wait exits.

A function can also be used to return objects instead of a Boolean value. In case of object make sure that your implementation of apply method returns neither null till the time object is nor found. Let's see this with the example. In the above test page we have a new object that we added in the HTML page after every few seconds. This object has an id = target. Let's wait for this object using a fluent wait. I will directly share the code with you, hope you can understand the details by looking at it by now.

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

            WebDriver driver = new FirefoxDriver();
            driver.get("http://softechsolutionsgroup.com/automation-
practice/");

            FluentWait<WebDriver> wait = new
FluentWait<WebDriver>(driver);
            wait.pollingEvery(250,  TimeUnit.MILLISECONDS);
            wait.withTimeout(2, TimeUnit.MINUTES);
            wait.ignoring(NoSuchElementException.class); //make sure
that this exception is ignored
            Function<WebDriver, WebElement> function = new
Function<WebDriver, WebElement>()
                        {
                              public WebElement apply(WebDriver arg0)
{
                                    System.out.println("Checking for
the element!!");
                                    WebElement element =
arg0.findElement(By.id("target"));
                                    if(element != null)
                                    {
                                          System.out.println("Target
element found");
                                    }
                                    return element;
                              }
                        };

            wait.until(function);
      }

}
```

What is a Predicate?

A **predicate** is similar to a **function** but it always returns a ***Boolean*** expression. You can define a predicate like this

```
Predicate<WebDriver> predicate = new Predicate<WebDriver>(){
      public boolean apply(WebDriver arg0) {
            return false;
      }

};
```

It’s just a matter of time that you implement some condition in the apply method such that it returns true in success and false otherwise. The color changing button can be waited on using a predicate like this

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

      WebDriver driver = new FirefoxDriver();
      driver.get("http://softechsolutionsgroup.com/automation-
      practice/");

      FluentWait<WebDriver> wait = new FluentWait<WebDriver>(driver);
      wait.pollingEvery(250,  TimeUnit.MILLISECONDS);
      wait.withTimeout(2, TimeUnit.MINUTES);

      //make sure that this exception is ignored
      wait.ignoring(NoSuchElementException.class);

      Predicate<WebDriver> predicate = new Predicate<WebDriver>(){
            public boolean apply(WebDriver arg0) {
                  WebElement element =
                  arg0.findElement(By.id("colorVar"));
                  String color = element.getAttribute("color");
                  System.out.println("The color if the button is " +
                  color);
                  if(color.equals("red"))
                  {
                        return true;
                  }
                  return false;
            }
      };
      wait.until(predicate);
}
```