1. **Static wait and**
2. **Dynamic wait**

1 .Static wait: Thread sleep wait

Thread.*sleep*(10000);

2. Dynamic wait (implicit and explicit wait)

2.1 Page load time out

*driver*.manage().timeouts().pageLoadTimeout(10, *TimeUnit*.*SECONDS*);

2.2. Implicit wait

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

2.3 Explicit wait

WebDriverWait webDriverWait11 = new WebDriverWait(*driver*, 30);

webDriverWait11.until(ExpectedConditions.*elementToBeClickable*(*driver*.findElement(By.*id*("id")))).click();

2.4polling wait

WebDriverWait wait22= new WebDriverWait(*driver*, 10);

wait22.pollingEvery(2, *TimeUnit*.*SECONDS*);

wait22.until(ExpectedConditions.*visibilityOf*(*driver*.findElement(By.*id*(""))));

2.5. Fluent wait

Wait<WebDriver> wait2 = new FluentWait<WebDriver>(*driver*).

withTimeout(30, *TimeUnit*.*SECONDS*).

pollingEvery(2, *TimeUnit*.*SECONDS*).

ignoring(NoSuchElementException.class);

WebElement userName = wait2.until(new Function<WebDriver, WebElement>() {

*@Override*

public WebElement apply(WebDriver driver) {

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

}

});

**1 .Static wait: Thread sleep wait**

**Thread.*sleep*(10000);**

Here irrespective of any condition the page will be wait for 10 seconds even after completion of all actions.

**2. Dynamic wait (implicit and explicit wait)**

**2.1 Page load time out**

*driver*.manage().timeouts().pageLoadTimeout(10, *TimeUnit*.*SECONDS*);

This wait to waiting to load the complete page for 10 seconds before 10 second if page loads then next actions will be continued else after 10 sec throws TimeOutException

**2.2. Implicit wait**

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

Implicit wait- Dynamic in nature

-It can be changed anywhere anytime in our code ie we can override and the latest value applied

Implicit wait- is always applied globally which directly applied to driver where it is available for all web element on a page.

**2.3 Explicit wait**

WebDriverWait webDriverWait11 = new WebDriverWait(*driver*, 30);

**webDriverWait11.until(ExpectedConditions.*elementToBeClickable*(*driver*.findElement(By.*id*("id")))).click();**

Explicit wait:

1. Explicit wait- also dynamic nature
2. No explicit keyword or method
3. Available with WebDriverWait class
4. Applied for specific element with some specific expected condition
5. Never use implicit and explicit wait together : Because selenium WD will wait for the element first because of IMLICIT wait and then EXPLICIT wait both will be applied hence total synchronization wait will be increased for each element.

**Do we need to have implicit and explicit wait together ?**

**No, It’s not at all recommended to use it both together, because when we apply** Implicit wait- this will always applied globally which directly applied to a driver where it is available for all web element with respective to driver.

So when we apply both together then first implicitly wait applied to element and again explicit wait applied to same element .

**Can we override Imlicit wait ?**

**Yes, the latest last applied Implicit wait value will be considered.**

**http://www.way2automation.com/webdriver-implicit-vs-explicit-vs-fluent-wait.php**

**WebDriver Wait, Implicit Wait, Explicit Wait and Fluent Wait**

**BY**[**RAMAN**](javascript:)**29/12/2014**

**680**Comments

**Selenium WebDriver: - Synchronizing a Test Using Waits**

One of the Important Factor in test automation for a complex web application is to ensure that the flow of the test cases should be in synchronization with the application under test (AUT).

When tests are run, the application may not always respond with the same speed. For example, it might take a few seconds for a progress bar to reach 100 percent, a status message to appear, a button to become enabled, and a window or pop-up message to open. We can handle these anticipated timing problems by synchronizing our test to ensure that Selenium WebDriver waits until our application is ready before performing a certain action. There are several options that we can use to synchronize our test.

**Synchronizing a test with an implicit wait**

When an implicit wait is implemented it tells 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's instance. However, an implicit wait may slow down our tests when an application responds normally, as it will wait for each element to appear in the DOM and increase the overall execution time.

*driver.manage().timeouts().implicitlyWait(Time period, TimeUnit.SECONDS);*

***Time period****:* Here time value is given as input. How many seconds the driver has to wait.  
***TimeUnit.SECONDS****:* Time period is measured as second here. You can use other time unit like day, microseconds, nanoseconds etc.

Example: -

**Synchronizing a test with FluentWait**

When a FluentWait instance is implemented it 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.

@Test

**public** **void** test(){

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

       .withTimeout(30, SECONDS)

       .pollingEvery(5, SECONDS)

       .ignoring(NoSuchElementException.class);

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

     public WebElement apply(WebDriver driver) {

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

     }

   });}

Above code snippet will Wait 30 seconds for an element to be present on the page and check for its presence once every 5 seconds.

**Synchronizing a test with an Explicit wait**

When an Explicit Wait is implemented it provides a better control when compared with an implicit wait. Unlike an implicit wait, we can write custom code or conditions for wait before proceeding further in the code. An explicit wait can only be implemented in cases where synchronization is needed and the rest of the script is working fine. The Selenium WebDriver provides ***WebDriverWait***and ***ExpectedCondition*** classes for implementing an explicit wait. The ExpectedCondition class provides a set of predefined conditions to wait before proceeding further in the code.

The following are some common conditions that we frequently come across when automating web browsers supported by the ExpectedCondition class:-

|  |  |
| --- | --- |
| **Predefined condition** | **Selenium method** |
| An element is visible and enabled | elementToBeClickable(By locator) |
| An element is selected | elementToBeSelected(WebElement element) |
| Presence of an element | presenceOfElementLocated(By locator) |
| Specific text present in an element | textToBePresentInElement(By locator,  java.lang.String text) |

**How it Works**:-

We can create a wait for a set of common conditions using the ExpectedCondition class. First, we need to create an instance of the WebDriverWait class by passing the **driver instance** and **timeout**for a wait as follows:

WebDriverWait wait = new WebDriverWait(driver, 10);

Next, ExpectedCondition is passed to the wait.until() method as follows:

wait.until(ExpectedConditions.titleContains("selenium"));

**Note**: - The WebDriverWait object will call the ExpectedCondition class object every 500 milliseconds until it returns successfully.

Example: -

@Test

**public** **void** test(){

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

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("Element")));    driver.findElement(By.id("Element")).clear();

driver.findElement(By.id("Element")).sendKeys("India");

driver.findElement(By.id("Element")).click();

}

**Custom-expected condition**: WebDriver provide us to create custom wait condition.

Example:

**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.support.ui.ExpectedCondition;

**import** org.openqa.selenium.support.ui.WebDriverWait;

**import** org.testng.annotations.AfterMethod;

**import** org.testng.annotations.BeforeMethod;

**import** org.testng.annotations.Test;

**public** **class** testwait {

**private** WebDriver driver;

**private** String baseUrl;

       @BeforeMethod

**public** **void** setUp() **throws** Exception {

             driver = **new** FirefoxDriver();

             baseUrl = "http://www.flipkart.com";

       }

       @Test

**public** **void** testUntitled() **throws** Exception {

             driver.get(baseUrl);

             // custom explicit wait for search field

**new** WebDriverWait(driver, 10)

                           .until(**new** ExpectedCondition<WebElement>() {

                                 @Override

**public** WebElement apply(WebDriver d) {

**return** d.findElement(By.*id*("fk-top-search-box"));

                                 }

                           });

             driver.findElement(By.*id*("fk-top-search-box")).sendKeys("Laptop");

             driver.findElement(

                           By.*xpath*(".//\*[@id='fk-header-search-form']/div/div/div[2]/input[1]"))

                           .click();

       }

       @AfterMethod

**public** **void** tearDown() **throws** Exception {

             driver.quit();

       }

}