**Selenium Keys :**

**Actions Class Method for Keyboard Interaction**

1. **keyDown(Keys modifierKey)-**  
   The keyDown(Keys modifierKey) method takes the modifier Keys as parameter (Shift, Alt and Control Keys – that modifies the purpose of other keys, hence the name). It is used to simulate the action of pressing a modifier key, without releasing. The expected values for the keyDown() method are – Keys.SHIFT, Keys.ALT and Keys.CONTROL only, passing key other than these results in IllegalArgumentException.
2. **keyDown(WebElement element, Keys modifierKey)-**  
   This another implementation of keyDown() method in which the modifier key press action is performed on a WebElement.
3. **keyUp(Keys modifierKey)-**  
   The keyUp() method is used to simulate the modifier key-up or key-release action. This method follows a preceeding key press action.
4. **keyUp(WebElement element, Keys modifierKey)-**  
   This implementation of keyUp() method performs the key-release action on a web element.
5. **sendKeys(CharSequence KeysToSend)-**  
   The sendKeys(CharSequence KeysToSend) method is used to send a sequence of keys to a currently focussed web element. Here, we need to note that it is different from the webElement.sendKeys() method. The Actions sendKeys(CharSequence KeysToSend) is particularly helpful when dealing with modifier keys as it doesn’t release those keys when passed(resulting in correct behaviour) unlike the webElement.sendKeys() method.
6. **sendKeys(WebElement element, CharSequence KeysToSend)-**  
   This implementation of sendKeys() method is used to send a sequence of keys to a web element.

**Actions Class Method for Mouse Interactions**

1. **click()-**  
   This method is used to click at the current mouse pointer position. It is particularly useful when used with other mouse and keyboard events, generating composite actions.
2. **click(WebElement webElement)-**This method is used to click at the middle of a web element passed as parameter to the click() method.
3. **clickAndHold()-**  
   The clickAndHold() method is used to perform the click method without releasing the mouse button.
4. **clickAndHold(WebElement onElement)-**  
   This method performs the click method without releasing the mouse button over a web element.
5. **contextClick()-**  
   This method is used to perform the right click operation(context-click) at the current mouse position.
6. **contextClick(WebElement onElement)-**  
   This method performs the right click operation at a particular web element.
7. **doubleClick()-**  
   As the name suggest, this method performs double click operation at a current mouse position.
8. **doubleClick(WebElement onElement)-**  
   Performs the double click operation at a particular web element.
9. **dragAndDrop(WebElement fromElement, WebElement toElement)-**  
   This is a utility method to perform the dragAndDrop operation directly wherein, we can pass the source element and the target element as parameter.
10. **dragAndDropBy(WebElement fromElement, int xOffset, int yOffset)-**  
    This method is a variation of dragAndDrop(fromElement, toElement) in which instead of passing the target element as parameter, we pass the x and y offsets. The method clicks the source web element and then releases at the x and y offsets.
11. **moveByOffset(int xOffset, int yOffset)-**  
    This method is used to move the mouse pointer to a particular position based on the x and y offsets passed as parameter.
12. **moveToElement(WebElement toElement)-**  
    This method is used to move the mouse pointer to a web element passed as parameter.
13. **moveToElement(WebElement toElement, int xOffset, int yOffset)-**  
    This method moves the mouse pointer by the given x and y offsets from the top-left corner of the specified web element.
14. **release()-**  
    This method releases the pressed left mouse button at the current mouse pointer position.
15. **release(WebElement onElement)-**  
    This method release the pressed left mouse button at a particular web element.

**Cookies**

driver.manage().getCookies(); // Return The List of all Cookies

driver.manage().getCookieNamed(arg0); //Return specific cookie according to name

driver.manage().addCookie(arg0); //Create and add the cookie

driver.manage().deleteCookie(arg0); // Delete specific cookie

driver.manage().deleteCookieNamed(arg0); // Delete specific cookie according Name

driver.manage().deleteAllCookies(); // Delete all cookies

## TimeOut

Time **timeOut** attribute within the @Test annotation method is assigned a value specifying the number of milliseconds. In case the test method exceeds the timeout value, the test method is marked as failure with **ThreadTimeoutException**.

|  |  |
| --- | --- |
| 1 | @Test(timeOut = 1000) |

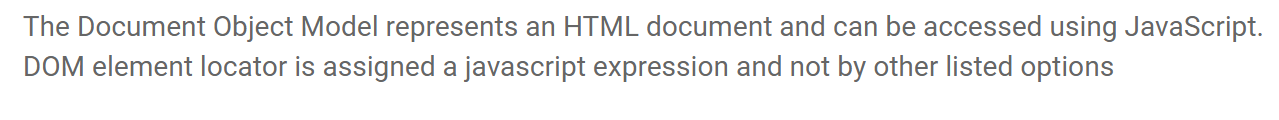
PS: The test stops execution as soon as the timeout duration is reached, marking the test as failure.

* Default port used by Jenkins – **8080**
* Default port for Selenium hub – **4444**
* Default port used by Selenium for communication – **4444**

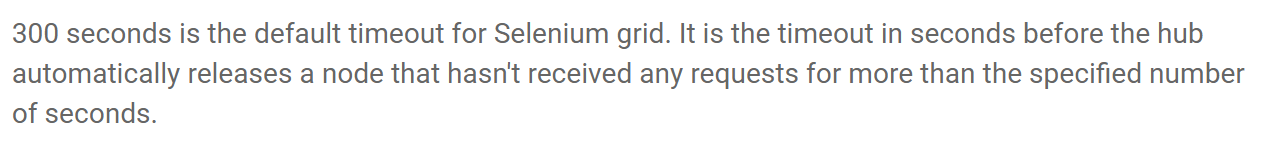
Remote Web Driver compose of **2 pieces:**

* A Client
* A Server

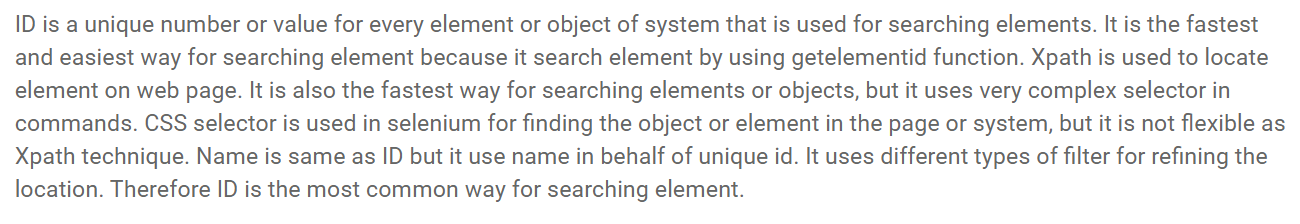
**DOM – Document Object Model**



**Default timeout for Selenium Grid: 300 Secs**



Common way of Locator



* Default port used by **Jenkins – 8080**
* Default port for **Selenium hub – 4444**
* Default port used by Selenium for **communication – 4444**
* SafariDriver implemented using – **Extensions**
* **Global Extension** of SafariDriver is responsible for communicating with the WebDriver Client
* Earliest version of Selenium was developed in **JavaScript**
* **Firebug** in Selenium is used for **Inspecting Elements**
* Default time for WAITFOR command is **30 secs.**
* **AJAX – Asynchronous JavaScript and XML**
* **Performance logtype** in Selenium, logs relating to the performance characteristics of the page under test (i.e resource load timings)
* Peer to Peer in HTML5 -technology -**WebRTC**
* **getCookies()** method extracts a cookie and parse it.
* **Open command in Selenium IDE is used to open a page using the URL**
* Default timeout value for Selenium hub release a node automatically **– 300 seconds**
* In Selenium grid – maximum number of browsers that can run in parallel by fault – **5 browsers**
* **DOM** **Query i**s used to locate element in JavaScript
  + **DOM** query is used for Evaluating / finding element on the page
  + **Xpath** query is use to identify nodes in XML document
  + **CSS Selector** is a selection of code that is use to define style property of web page
* Selenium RC server is an packaged **Java Jar file**
* **In Selenium 2 ways for storing value into Variable**
  + store(expression, variable name)
  + storedVars[‘variable name’]
* All variables created in test cases are stored in a Java script **Associative Array .** Associative Array has string index rather than sequential numeric index.
* userExtension can be given to selenium Server during startup for loading Javascript file
* In Javascript, executeScript() is used to move to a web Element
* **Selenium IDE does not support** – **Unix OS**
* HTTP Cookie is also called as a web cookie, a browser cookie or an Internet cookie.
* **WebDriverEventListener** is provided in WebDriver to track events those take place in WebDriver during **script execution.**
* Under the banner of **HTML5**, modern web standards such as CSS3, SVG, XHR2, WebSockets, IndexedDB, and AppCache are pushing the boundaries for what a browser can achieve using web standards.
* **Selenium IDE will NOT work on Firefox version 55 onwards.**
* **WAR file.** The Web application ARchive (WAR) file version of Jenkins can be installed on any operating system or platform that supports Java
* In TestNG, each instance or suites to be run in a separate thread
* Another common usage of wrapping Selenium methods is to check for presence of an element on page before carrying out some operation. This is sometimes called a **‘safe operation’.**For instance, the following method could be used to implement a safe operation that depends on an expected element being present.
* Jenkins supported OS – Linux, Mac OS and Windows
* Mavan supported OS – Linux, Mac OS and Windows
* AutoIT Supported OS – Windows
* **JAVA\_HOME** environment is essential in integrating Selenium and Mavan
* Minimum version of JDK needed to install latest Mavan installation is –

**JDK Version 1.7**

* Selenium IDE is an **XPI file which is an Firefox add-in** file
* **FireEvent** is used to customizing events in Selenium

**Pattern Match in Regular Expression:**

[] Any Single character

Asterisk \* - 0 or more character

+ - 1 or more characters

Implicit Wait:

Driver.manage().timeouts().implicitlyWait(10,Timeout.SECONDS);

**Maximum Parameter passed in the function:**

* clickAndHold() – 2
* isDisplayed() – 0
* dragAndDrop – 2
* dragAndDropBy - 3

**Number of Argument Accepted by the function:**

* afterChangeValueOf(WebElement arg0, WebDriver arg1) – 2
* afterClickOn(WebElement arg0, WebDriver arg1) – 2
* afterFindBy(By arg0, WebElement arg1, WebDriver arg2) – 3
* afterNavigateBack(WebDriver driver) – 1
* afterNavigateForward(WebDriver arg0) – 1
* afterNavigateTo(String arg0, WebDriver arg1) – 2
* afterScript(String arg0, WebDriver arg1) - 2
* beforeChangeValueOf(WebElement arg0, WebDriver arg1) – 2
* beforeClickOn(WebElement arg0, WebDriver arg1) – 2
* beforeFindBy(By arg0, WebElement arg1, WebDriver arg2) – 3
* beforeNavigateBack(WebDriver driver) – 1
* beforeNavigateForward(WebDriver arg0) – 1
* beforeNavigateTo(String arg0, WebDriver arg1) – 2
* beforeScript(String arg0, WebDriver arg1) – 2
* onException(Throwable arg0, WebDriver arg1) – 2
* afterNavigateRefresh(WebDriver driver) - 1
* beforeNavigateRefresh(WebDriver driver) - 1
* beforeClickOn(WebElement element,   WebDriver driver) -2
* afterClickOn(WebElement element,  WebDriver driver)-2
* beforeScript(java.lang.String script,  WebDriver driver) - 2
* afterScript(java.lang.String script, WebDriver driver) - 2

selectByValue(“pencil”)

captureElementBitmap(canvasElement)