WebDriver

Why we need automation:

It saves time to execute regression test cases.

It simplifies cross browser testing.

We can do data driven testing.

Defects not found through manual testing and complex testcases can be automated.

Reusability and avoid redundancy.

When to automate and when not to automate?

when not to automate:

If UI of project is going to be changed – no need to automate.

Tight dead lines

Look and feel related components

when to automate:

when we know that project needs more and more regression.

It saves time to execute regression test cases.

when we need cross browser testing and data driven testing.

Defects not found through manual testing and complex testcases can be automated.

Reusability and avoid redundancy.

Automation:

UI/Web application/Browser automation – WebDriver, Nightwatch.js, Protractor etc

WebServices Automation- SOAP and REST – SoapUI, HttpCLient, RestAssured, Chakram

Mobile applications automation- Appium, UIAutomator ,UIAutomation, Perfecto, Calabash etc.

test frameworks – TestNG, JUnit, Chai.js etc..

UI/Web application/Browser automation :

Other automation tools – QTP, SilkTest etc were used earlier

Selenium – IDE, RC, WebDriver, Grid

Selenium WebDriver

IDE :

Record and playback tool- it is firefox plugin.

**download**:

https://addons.mozilla.org/en-US/firefox/addon/selenium-ide/

TestCase- description, teststeps, testdata, expected, actual , result

Command – Action perfomed in browser

Target – locating the element on which html element we are performing the action

Value- input to the element like text boxes..

Assert - if the actual and expected do not match- test wil not continue execution

verify – even if the actual and expected do not match- test wil still continue execution , we can see log statement that the verification failed.

Disadvantage of IDE:

We can execute tests only in firefox- it is only firefox plugin.

We cannot do continuous build and continuous integration.

data driven testing cannot be done.

**Selenium Remote Control** (RC) is a test tool that allows you to write automated web application UI tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser.

Selenium RC comes in two parts.

1. A server which automatically launches and kills browsers, and acts as a HTTP proxy for web requests from them.
2. Client libraries for your favorite computer language

Set of jars using which we can develop automation code.

RC supports different languages for automation like Java,Ruby, Python etc.

**package** **com.example.tests**;

**import** **com.thoughtworks.selenium.\***;

**import** **java.util.regex.Pattern**;

**public** **class** **NewTest** **extends** SeleneseTestCase {

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

setUp("http://www.google.com/", "\*firefox");

}

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

selenium.open("/");

selenium.type("q", "selenium rc");

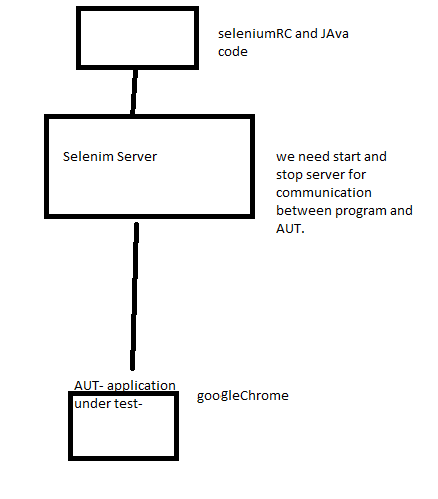
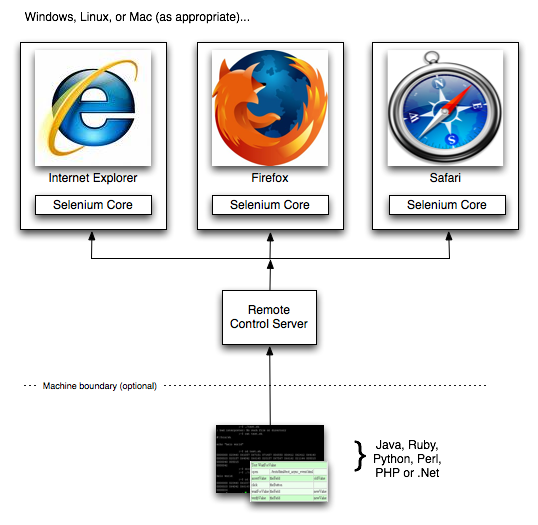
selenium.click("btnG");

selenium.waitForPageToLoad("30000");

assertTrue(selenium.isTextPresent("Results \* for selenium rc"));

}

}



Disadvantage:

It is slower than WebDriver as we need to start/stop server

It injects javascript functions into the browser when the browser was loaded and then uses its javascript to drive the AUT within the browser **Grid.**

**Selenium-Grid** allows you run your tests on different machines against different browsers in parallel. That is, running multiple tests at the same time against different machines running different browsers and operating systems. Essentially, Selenium-Grid support distributed test execution. It allows for running your tests in a distributed test execution environment.

Two reasons why you might want to use Selenium-Grid.

* To run your tests against multiple browsers, multiple versions of browser, and browsers running on different operating systems.
* To reduce the time it takes for the test suite to complete a test pass.

**WebDriver:**

WebDriver is automation tool used for browser/ web applications automation.

WebDriver API provides many classes and interfaces to develop automation scripts.

WebDriver is an main interface in WebDriver API which has multiple implementation classes like FirefoxDriver,ChromeDriver,SafariDriver etc.

How Does WebDriver ‘Drive’ the Browser Compared to Selenium-RC?[¶](http://www.seleniumhq.org/docs/03_webdriver.jsp#how-does-webdriver-drive-the-browser-compared-to-selenium-rc)

Selenium-WebDriver makes direct calls to the browser using each browser’s native support for automation. How these direct calls are made, and the features they support depends on the browser you are using. Information on each ‘browser driver’ is provided later in this chapter.

For those familiar with Selenium-RC, this is quite different from what you are used to. Selenium-RC worked the same way for each supported browser. It ‘injected’ javascript functions into the browser when the browser was loaded and then used its javascript to drive the AUT within the browser. WebDriver does not use this technique. Again, it drives the browser directly using the browser’s built in support for automation.

**What we need for automation:**

Automation Tool – WebDriver

ProgrammingLanguage- Java

TestFramework- Creating TestSuites, assertions, test reports – TestNG

Maven – Build Tool

Git- code repository management

Jenkins- Continuous integration

**Automation script has 4 main parts:**

Load the browser with required URL.

Identify the element to automate – we use locators

perform the action on the element

assert for actual and expected.

text box - <input id="gh-ac" class="gh-tb ui-autocomplete-input" size="50" maxlength="300" placeholder="Search..." name="\_nkw" autocapitalize="off" autocorrect="off" spellcheck="false" autocomplete="off" aria-live="polite" role="status" aria-haspopup="false" type="text"/>

links - <a href=”google.com” id=”linkid”>google</a>

dropdowns-

<select id =” “ name =””>

<options>

</select>

checkboxes

radio buttons

button- <button id=”” name=”” class=”” >submit</button>

<ul>

<li>watch</li>

<li>Bag</li>

<li>laptop</li>

<li>watch</li>

</ul>

div

<span class="ui-helper-hidden-accessible" role="status" aria-live="polite"/>

**Locating Elements:**

**By class**: 8 static methods – using class name u can call the methods.

By id(String locator)-

name

linkText

partialLinkText

css selectors-

xpath-

tagName

className

eg: By by = By.id(“mail”);

id is first preferred locator as it is faster ,

if we don’t have unique id- if you could find unique name or linkText you can prefer them

else CssSelector is preferred as it is simple and good performance.

In few scenarios we might need xpath over css selectors:

1. backward navigation in html page is possible through xpath but not css selector

2. if we have to locate element using inner text(text found b/n opening tag and closing tag of html element) of element – we can use text() and contains() methods in xpath.

css selectors:

#idValue

.classValue

tagName

<div id =”d”>

<div id=”d1”>

<a class=”aclass”>clickme</a>

</div>

</div>

#d a – any child

#d>div – direct child

#d1>a- direct child

elementName[attributeName=’valueOfAttribute’]

input[id$='location'] – end with

div[class^='col-hidden']- start with

div[class\*='col-vertical-s']- contains

div[class='keel-grid verticalsGrid'] div:nth-of-type(4)

div[class='keel-grid verticalsGrid'] div:nth-child(4)

multiple attributes for same element- works like **and** condition-element which has both class contains common **and** href contains hotels

a[class\*='Common'][href\*='hotels']

input[id='Z49E-destination'] + input – sibling

or

#Z49E-destination + input

input[id='Z49E-destination'] or #Z49E-destination

input[class='Z49E-destination'] or .Z49E-destination

**WebDriver Interface:**

get

WebElement findElement(By by) – takes By class object as input- using By class we pass the locator to findElemeent method and once it finds the element this method will return the element using WebElement interface as return type.

List<WebElement> findElements(By by) – returns list of webelements.

**diff b/n findElement and findElements:**

|  |  |
| --- | --- |
| findElement | findElements |
| findElement is used to locate single element | findElements is used to locate list of elements |
| findElement returns NoSuchElementFoundException if element is not found in page. | findElements returns empty list(list with size-0) if element is not found in page.  So this can be used to verify presence of element i.e., if size is >0 –element is found else element is not found in the page. |

close

getTitle

getPageSource

getCurrentUrl

navigate():

Navigate nav = driver.navigate();

Navigation class methods:

to – similar to get method- it load page with given url

forward- goes to prev page in browser

back- goes to next page in browser

refresh- refresh the page

**diff b/n navigate and get :**

navigate.to is similar to get method which loads the page with given url but navigate has some other methods like back, forward and refresh which we cannot do using get.

windowhandles – will return all window names

windowhandle- will return current window name

switchTo- is used to switch to a specific window by using name of the window.

close and quit:

close will close the current window on which driver focus is on

but quit will close all the windows that are opened by the driver.(quit is useful to close multiple windows opened by script)

maximize

**WebElement Interface:**

click

sendKeys

getText – inner text of any element

getAttributeValue

getCSsValue

findElement

findElements

isDisplayed

isSelected

isEnabled

Select- for drop downs

RadioButtons

**All exceptions during selenium automation:**

org.openqa.selenium.**NoSuchElementException**: no such element: Unable to locate element:

org.openqa.selenium.**StaleElementReferenceException**: stale element reference: element is not attached to the page document

org.openqa.selenium.**NoSuchSessionException**: Session ID is null. Using WebDriver after calling quit()