XPath Tutorial for Selenium WebDriver

In this tutorial, we are going to study the XPath locators in Selenium WebDriver. In our post on Finding WebElements in Selenium, we studied different types of locators used in Selenium WebDriver. Here, we will be studying how to create Xpath locators, the different types of Xpaths and the ways of finding dynamic elements using XPath.

What is an XPath?

An XPath can be defined as a query language used for navigating through the XML documents in order to locate different elements. The basic syntax of an XPath expression is-

```
//tag[@attributeName='attributeValues']
```

Now, let's understand the different elements in the Xpath expression syntax - tag, attribute and attributeValues using an example. Consider the below image of Google webpage with Firebug inspecting the Search-bar div.

- **'/' or '//'** - The single slash and double slash are used to create absolute and relative XPaths(explained later in this tutorial). Single slash is used to start the selection from root node. Whereas, the double slash is used to fetch the current node matching the selection. For now, we will be using '//' here.

- **Tag** - Tags in HTML begin with '<' and end with '>'. These are used to enclose different elements and provide information about processing of the elements. In the above image, 'div' and 'input' are tags.
- **Attribute** - Attributes define the properties that the HTML elements hold. In the above image, id, classes and dir are the attributes of the outer div.
- **AttrbuteValue** - AttributeValues as the name suggest, are the values of the attributes e.g. 'sb_ifc0' is the attribute value of 'id'.

Using the XPath syntax displayed above, we can create multiple XPath epressions for the google searchbar-div given in the image like- **//div[@id='sb_ifc0']**, **//div[@class='sbib_b']** or **//div[@dir='ltr']**. Any of these expressions can be used to fetch the desired element as long as the attributes chosen are unique.

What are the different types of XPath?

There are two kinds of XPath expressions-

1. **Absolute XPath** - The XPath expressions created using absolute XPaths begins the selection from the root node. These expression either begin with the '/' or the root node and traverse the whole DOM to reach the element.
2. **Relative XPath** - The relative XPath expressions are a lot more compact and use forward double slashes '//'. These XPaths can select the elements at any location that matches the selection criteria and doesn't necessarily begin with root node.

So, which one of the two is better?- The relative XPaths are considered better because these expressions are easier to read and create; and also more robust. The problem with absolute XPaths is, even a slight change in the DOM from the path of root node to the desired element can make the XPath invalid.

Finding Dynamic Elements using XPaths

Many a times in automation, we either don't have unique attributes of the elements that uniquely identify them or the elements are dynamically generated with the attribute's value not known beforehand. For cases like these, XPath provide different methods of locating elements like - using the text written over the elements; using element's index; using partially matching attribute value; by moving to sibling, child or parent of an element which can be uniquely identified etc.

Using text()

Using text(), we can locate an element based on the text written over it e.g. XPath for the 'GoogleSearch' button -

//*[text()='Google Search'] (we used '*' here to match any tag with the desired text)

Using contains()

The contains(), we can match even the partially matching attributes values. This is particularly helpful for locating dynamic values whose some part remains constant e.g. XPath for the outer div in the above image having id as 'sb_ifc0' can be located even with partial id-'sb' using contains() - **//div[contains(@id,'sb')]**

Using element's index

By providing the index position in the square brackets, we can move to the nth element satisfying the condition e.g. **//div[@id='elementid']/input[4]** will fetch the fourth input element inside the div element.

Using XPath axes

XPath axes helps in locating complex web elements by traversing them through sibling, child or parent of other elements which can be identified easily. Some of the widely used axes are-

- **child** - To select the child nodes of the reference node. Syntax - XpathForReferenceNode/child::tag
- **parent** - To select the parent node of the reference node. Syntax - XpathForReferenceNode/parent::tag
- **following** - To select all the nodes that come after the reference node. Syntax - XpathForReferenceNode/following::tag
- **preceding** - To select all the nodes that come before the reference node. Syntax - XpathForReferenceNode/preceding::tag
- **ancestor** - To select all the ancestor elements before the reference node. Syntax - XpathForReferenceNode/ancestor::tag