CS1632: Lecture 13

Web testing with Selenium Part II

Bill Laboon/Dustin Iser

Using Selenium with Java

- Libraries needed
  - Unit testing framework (Junit)
  - Selenium libraries
- Executables needed
  - WebDriver
- Write some code!

Setting up the driver

```
WebDriver driver = HtmlUnitDriver();
WebDriver driver = new FirefoxDriver();


// HtmlUnitDriver is lightweight and faster
// FirefoxDriver is most supported
// Other drivers exist!
// I recommend you use one of these two
```

Navigating to a web page

```
driver.get("http://www.google.com");
```

WebElements

After executing ‘get’, the WebDriver now contains references to all the WebElements that make up Google’s home page.

A WebElement is anything that makes up a webpage.

- Divs
- Labels
- Buttons
- Images
- Pages

Find Elements

You can use driver.findElement() to find a specific WebElement on a page.

FindElement accepts one argument, a By object.

You can look up WebElements several different ways.

- By.id(“some-id”);
- By.cssSelector(“some css selector”);
- By.linkText(“some link text”);
- By.xpath(“//some/x/path”);

Element selection methods

Id is preferred.

Testers should always encourage the use of ids in web applications.

Sometimes ids are difficult. For example:

- Grids
- Lists
- Dynamic elements

XPath selectors

```
<html>
    <body>
        <h1>Hello World!</h1>
        <h2>Hello Mars!</h2>
    </body>
</html>
```

If we wanted to select the h2 element via xpath:

/html/body/h2
Or
//h2

XPath selectors

Can use index locators

```
<html>
    <body>
        <h1>Hello World!</h1>
        <h1 id=’sub-header’>Hello Mars!</h1>
    </body
</html>

//h1[1]
```

XPath selectors

Can use attributes.

```
<html>
	<body>
		<h1>Hello World!</h1>
		<h1 id=’sub-header’>Hello Mars!</h1>
	</body
</html>
```

```
//h1[@id=’sub-header’]
Or
//*[@id=’sub-header’]
```

XPath selectors

Can use axes.

```
<html>
    <body>
        <h2>Hello World!<h2>
        <br>
        <li id='list-of-elements'>
            <li>
                <h1>First!</h1>
                <h2>Second!</h2>
            </li>
        </li>
    </body>
```

```
//*[@id='list-of-elements]/descendant::h2
```

XPath selectors

When they can’t be avoided, follow some best practices.

- Find an anchoring point
- Don’t use index locators
- Use the text() method
- Watch the depth

Find an anchoring point

Use a nearby element that has an id to anchor your xpath selector.

```
<html>
    <body>
        <div>
            <div>
                <div id='everything-divs'>
                    <h1>Hello World!</h1>
                </div>
            </div>
        </div>
    </body>
</html>
```

```
//div[@id='everything-divs']/h1
```

Don’t use index locators

Unless you specifically want to get the nth element of a list, don’t use index locators.

Use the text() method

```
<html>
    <body>
        <div>
            <div>
                <div>
                    <h1>Hello World!</h1>
                </div>
            </div>
        </div>
    </body>
</html>
```

```
//*[text()='Hello World!']
```

Watch the depth

Try to avoid xpath selectors that exceed a depth of 2.

//really/long/xpaths/are/brittle/and/make/tests/fail/easily

Manipulating web browsers

Now that you have a reference to a WebElement, you can do things.

textBox.sendKeys(“Hello World!”);

checkbox.click();

The WebDriverWait

With modern dynamic websites, it is necessary for tests to wait for a website to be ready before executing additional steps.

Two different ways to wait:

- Implicit
- Explicit

Implicit wait

Instructs the test to wait n seconds. The test waits “exactly” n seconds.

Thread.sleep(5);

This isn’t good practice for a couple reasons:

- Wastes resources
- Tests still fail

Explicit wait

Instructs the test to wait n seconds or until a condition evaluates to true.

For example, wait 30 seconds or until the submit button is enabled.

```
WebDriverWait wait = new WebDriverWait(driver, 30);
wait.until((Predicate<WebDriver>)w -> {
        return w.findElement(By.id(“submit_button”).isEnabled);
});
```

This is good practice because:

- Builds rigid tests
- Doesn’t waste resources.

Stopping the driver

```
// Good clean-up practice
driver.quit();
```

Assertions

```
// Get the foo element, check that its text is “foo!”
WebElement fooElement = driver.findElement(By.id("foo"));

assertEquals(fooElement.getText(), "foo!");
```