COURSEWARE

Browser manipulation

Contents

Overview

The `WebDriver.get(String path)` instance method can be used to navigate to a website in a simple and easy manner, in this module we will explore programmatic browser control by using the `WebDriver` API.

Basic navigation

Browser navigation is a simple process in Selenium, a `WebDriver.get()` instance method is provided that will open a website in the specific driver instance:

```
@Test
public void bingTest() {
    driver.get("https://www.bing.com");

    Assert.assertEquals("Bing", driver.getTitle());
}
```

Selenium offers a `Navigation` API that can be used to navigate around a website, an instance of the `Navigation` class can be retrieved from a `WebDriver` instance and used to navigate to a supplied URL:

```
@Test
public void bingTest() {
    Navigator navigator = driver.navigate();
    navigator.to("https://www.bing.com");

    Assert.assertEquals("Bing", driver.getTitle());
}
```

The `Navigator` API offers many useful methods for navigating a website, some of them are:

Method	Description
`Navigation.to(String url)`	Navigates to the supplied URL using a HTTP POST operation
`Navigation.to(URL url)`	Uses a URL object rather than a String
`Navigation.back()`	Moves back one page in the browser's history

Method	Description
`Navigation.forward()`	Moves forward one page in the browser's history, does nothing if on the latest page
`Navigation.refresh()`	Refreshes the current page

Basic browser controls

When using a web browser, the user can distinguish between many different parts of the system. There is the browser application itself, and then it may have multiple tabs which are open on different web pages for example. Selenium is not aware of this, Selenium instead quantifies a browser window or a browser tab as just a window represented by a unique string identifier. We can retrieve the identifier of the current window with the `WebDriver.getWindowHandle()` instance method:

```
@Test
public void bingTest() throws Exception {
    String originalTab = driver.getWindowHandle();
    driver.get("https://www.bing.com");

    ((JavascriptExecutor) driver).executeScript("window.open()");
    String newTab = driver.getWindowHandle();

    driver.switchTo().window(originalTab);

    driver.close(); // closes the original tab, switches back to the new tab

    // Force the app to wait so the result can be seen, remove in production
    Thread.sleep(3000);
}
```

In this example, the `WebDriver` instance was coerced to be of the `JavaScriptExecutor` type; this allows for the execution of JavaScript code. The `driver.getWindowHandle()` instance method is what returned the unique string identifier for the new browser tab.

The previous shown example was how we would create and navigate to a new tab using Selenium 3, a `NewWindow` API has been supplied in Selenium 4 to simplify this process:

```
@Test
public void bingTest() throws InterruptedException {
    String originalTab = driver.getWindowHandle();
    driver.get("https://www.bing.com");

    driver.switchTo().newWindow(WindowType.TAB);
    driver.get("https://www.google.com");
    driver.close();

    driver.switchTo().window(originalTab);

    driver.close(); // closes the original tab, switches back to the new tab

    // Force the app to wait so the result can be seen, remove in production
    Thread.sleep(3000);
}
```

The `newWindow` method takes a `WindowType` as input, specify `WindowType.Window` to open a new browser window rather than tab. The `driver.switchTo()` method returns a `TargetLocator` object which is used to locate frames or windows.

- After closing a tab, ensure to switch back to another tab before specifying any extra instructions otherwise a `NoSuchWindowException` will be thrown

The `Window` API

Selenium provides a `Window` API which can be used to manage browser windows, a `Window` object can be retrieved from the `Options` object on a `WebDriver` implementation:

```
Options options = driver.manage();
Window window = options.window();
```

The Window interface is used for controlling the current browser window.From this object, we can retrieve things like the size of our browser window or the coordinates of our window relative to the top left corner of the screen, for example:

```
Window window = driver.manage().window();
Dimension oldWindowSize = window.getSize();
Dimension newWindowSize = new Dimension(1366, 768);
window.setSize(newWindowSize);
```

The `getSize()` method returns a `Dimension` object with the current size of the browser window, we can override this by providing a `Dimension` object to the `setSize()` method. See the following table for more instance methods on the `Window` type:

Method	Description
`Window.getSize()`	Gets the size of the current browser window as a `Dimension` object, not the size of the viewport
`Window.setSize(Dimension targetSize)`	Sets the size of the current browser window, not the viewport
`Window.fullscreen()`	Sets the browser to full screen mode, equivalent to F11 in most browsers
`Window.maximise()`	Maximises the size of the current window
`Window.minimise()`	Minimises the size of the current window
`Window.getPosition()`	Returns a `Point` object representing the coordinates of the current window where the top left of the screen is the origin (0,0)
`Window.setPosition(Point targetPosition)`	Sets the position of the current window relative to the top left origin (0,0)

Taking screenshots

Taking screenshots of web pages is often required during testing, this allows for real humans to review the automated testing with greater certainty – it is also very useful when exceptions occur, a screenshot can be taken when a button fails to work for example.

The simplest way to take a screenshot is to typecast the `WebDriver` instance to a `TakesScreenshot` instance, then call its API:

```
File screenshot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
screenshot.renameTo(new File("./screenshot.png"));
screenshot.createNewFile();
```

Here, the `TakesScreenshot.getScreenshotAs(OutputType outputType)` instance method is called to take a screenshot of the current webpage and return it as a `File` object. The screenshot is then appropriately renamed and then saved

using the `File.createNewFile()` instance method. This outputs the file to the root of the project folder in this case. To make this reusable, move the functionality into a method:

```
public void takeScreenshot(String path) throws IOException {
    File screenshot = ((TakesScreenshot)
driver).getScreenshotAs(OutputType.FILE);
    screenshot.renameTo(new File(path));
    screenshot.createNewFile();
}
```

The method can then be called like so, eunsuring the filename is on the end of the path: `takeScreenshot("./src/test/resources/screenshot.png");`

Key points

- The `Navigation` API is used to navigate the browsers history and to specified URLs
- The `newWindow()` API, new to Selenium 4, can be used to open new windows and tabs
- The `Window` API allows for the control of a windows size and position
- Screenshots of an entire page, or just an element, can be taken using the `TakesScreenshot` API

Tutorial

A simple screenshot manager

Screenshots are very useful and are common in User Acceptance Testing, they provide visual feedback to reviewers that provides more context than just statistics can – statistics, often called **metrics**, are very important and are often utilized to support the achieving of **key performance indicators**. Due to this, we will create a simple class to manage our screenshot functionality. We will also add a way to take screenshots of individual elements on a page.

- A **metric** is anything that is measurable, i.e. it provides some information about something that may or may not have been known previously.
- A **key performance indicator** is similar to a metric, but it is instead used by a business to measure metrics against to indicate whether some business goal is being achieved or not.

```
/**
 * A simple utility class for taking screenshots via a Selenium WebDriver.
 *
 * @author Morgan Walsh
 */
public class ScreenshotManager {

    /**
     * Holds the current screenshot, or null if none have been taken
     */
    private File currentScreenshot;

    /**
     * Takes a screenshot of the current browser tab
     *
     * @param driver - a WebDriver implementation representing the browser
instance
     */
    public void takeScreenshot(WebDriver driver) {
        currentScreenshot = ((TakesScreenshot)
driver).getScreenshotAs(OutputType.FILE);
    }

    /**
     * Takes a screenshot of the element at the specified selector.
     *
     * @param driver - a WebDriver implementation representing the browser
instance
     * @param selector - a By element selector to select the element to
screenshot
     */
    public void takeElementScreenshot(WebDriver driver, By selector) {
        WebElement element = driver.findElement(selector);
        currentScreenshot = element.getScreenshotAs(OutputType.FILE);
    }

    /**
     * Takes a screenshot of the current browser tab.
     *
     * @param driver - a WebDriver implementation representing the browser
instance
     * @param path - the path to save the image, "./images/test-image.png" would
be a valid path as a string
     * @throws IOException - thrown if the file couldn't be saved
     */
    public void takeAndSaveScreenshot(WebDriver driver, String path) throws
IOException {
        takeScreenshot(driver);
        saveScreenshot(path);
    }

    /**
     * Takes a screenshot of the element at the specified selector.
     *
     * @param driver - a WebDriver implementation representing the browser
instance
     * @param selector - a By element selector to select the element to
screenshot
     * @param path - the path to save the image, "./images/test-image.png" would
be a valid path as a string
     * @throws IOException - thrown if the file couldn't be saved
     */
    public void takeAndSaveElementScreenshot(WebDriver driver, By selector,
String path) throws IOException {
        takeElementScreenshot(driver, selector);
        saveScreenshot(path);
    }

    /**
     * Saves the current screenshot in the File object to the specified path,
the last portion of the path is the filename.
```

```
     * <br />
     * <br />
     *
     * @param path - the path to save the image, "./images/test-image.png" would
be a valid path as a string
     * @throws IOException - thrown if the file couldn't be saved
     */
    public void saveScreenshot(String path) throws IOException {
        currentScreenshot.renameTo(new File(path));
        currentScreenshot.createNewFile();
    }
}
```

- Take note of the `WebElement.getScreenshotAs(OutputType outputType)` instance method, this allows for screenshots to be directly taken of a `WebElement` instance

The `ScreenshotManager` class stores a `File` object representing the current screenshot, this could be of the whole screen or just an element. The `takeScreenshot(WebDriver driver)` method will take a screenshot of the entire page, the `takeElementScreenshot(WebDriver driver, By selector)` takes a screenshot of just the specified element. The `saveScreenshot(String path)` method will save the screenshot stored in the `File` object to the designated path. The `takeAndSaveScreenshot` and `takeAndSaveElementScreenshot` methods are combination methods that take advantage of the existing functionality. To use this class, simply instantiate an instance and use as follows:

```
public void takeScreenshotExample() throws IOException {
    WebDriver driver = new ChromeDriver();
    ScreenshotManager screenshotManager = new ScreenshotManager();
    driver.navigate().to(new URL("https", "bing.com", 443, ""));

    screenshotManager.takeAndSaveScreenshot(driver,
"./target/screenshots/screenshot.png");
    screenshotManager.takeAndSaveElementScreenshot(driver, By.name("q"),
"./target/screenshots/element-screenshot.png");
}
```

Exercises

1. Implement the `ScreenshotManager` class from the **Tutorial** section in a new project
    1. Create a test to navigate to a news site of your choice, and then confirm the pages title.
    2. In your test, use an instance of `ScreenshotManager` to take a screenshot of the entire page
2. Create a new test to navigate to a search provider, such as Google or Bing, and then assert that input typed into the `<input>` element for searching actually is inserted.
    1. Use an instance of `ScreenshotManager` to take and save a screenshot of the entire page before entering the input
    2. Use the same instance of `ScreenshotManager` to take and save a screenshot of the `<input>` element that your text is entered in after the text has been sent to it