Frequently Asked Questions

Luke Inman-Semerau edited this page on 21 Apr 2015 · 2 revisions

summary WebDriver FAQs

labels Featured,WebDriver

FAQ

Q: What is WebDriver?

A: WebDriver is a tool for writing automated tests of websites. It aims to mimic the behaviour of a real user, and as such interacts with the HTML of the application.

Q: So, is it like Selenium? Or Sahi?

A: The aim is the same (to allow you to test your webapp), but the implementation is different. Rather than running as a Javascript application within the browser (with the limitations this brings, such as the "same origin" problem), WebDriver controls the browser itself. This means that it can take advantage of any facilities offered by the native platform.

Q: What is Selenium 2.0?

A: WebDriver is part of Selenium. The main contribution that WebDriver makes is its API and the native drivers.

Q: How do I migrate from using the original Selenium APIs to the new WebDriver APIs?

A: The process is described in the Selenium documentation at http://seleniumhq.org/docs/appendix_migrating_from_rc_to_webdriver.html

Q: Which browsers does WebDriver support?

A: The existing drivers are the ChromeDriver, InternetExplorerDriver, FirefoxDriver, OperaDriver and HtmlUnitDriver. For more information about each of these, including their relative strengths and weaknesses, please follow the links to the relevant pages. There is also support for mobile testing via the AndroidDriver, OperaMobileDriver and IPhoneDriver.

Q: What does it mean to be "developer focused"?

A: We believe that within a software application's development team, the people who are best placed to build the tools that everyone else can use are the developers. Although it should be easy to use WebDriver directly, it should also be easy to use it as a building block for more sophisticated tools. Because of this, WebDriver has a small API that's easy to explore by hitting the "autocomplete" button in your favourite IDE, and aims to work consistently no matter which browser implementation you use.

Q: How do I execute Javascript directly?

A: We believe that most of the time there is a requirement to execute Javascript there is a failing in the tool being used: it hasn't emitted the correct events, has not interacted with a page correctly,

or has failed to react when an XmlHttpRequest returns. We would rather fix WebDriver to work consistently and correctly than rely on testers working out which Javascript method to call.

We also realise that there will be times when this is a limitation. As a result, for those browsers that support it, you can execute Javascript by casting the WebDriver instance to a JavascriptExecutor. In Java, this looks like:

```
WebDriver driver; // Assigned elsewhere
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("return document.title");
```

Other language bindings will follow a similar approach. Take a look at the UsingJavascript page for more information.

Q: Why is my Javascript execution always returning null?

A: You need to return from your javascript snippet to return a value, so:

```
js.executeScript("document.title");
```

will return null, but:

```
js.executeScript("return document.title");
```

will return the title of the document.

Q: My XPath finds elements in one browser, but not in others. Why is this?

A: The short answer is that each supported browser handles XPath slightly differently, and you're probably running into one of these differences. The long answer is on the XpathInWebDriver page.

Q: The InternetExplorerDriver does not work well on Vista. How do I get it to work as expected?

A: The InternetExplorerDriver requires that all security domains are set to the same value (either trusted or untrusted) If you're not in a position to modify the security domains, then you can override the check like this:

```
DesiredCapabilities capabilities = DesiredCapabilities.internetExplorer();
capabilities.setCapability(InternetExplorerDriver.INTRODUCE_FLAKINESS_BY_IGNORING_SECURITY_D
WebDriver driver = new InternetExplorerDriver(capabilities);
```

As can be told by the name of the constant, this may introduce flakiness in your tests. If all sites are in the same protection domain, you *should* be okay.

Q: What about support for languages other than Java?

A: Python, Ruby, C# and Java are all supported directly by the development team. There are also webdriver implementations for PHP and Perl. Support for a pure JS API is also planned.

Q: How do I handle pop up windows?

A: WebDriver offers the ability to cope with multiple windows. This is done by using the "WebDriver.switchTo().window()" method to switch to a window with a known name. If the name is not known, you can use "WebDriver.getWindowHandles()" to obtain a list of known windows. You may pass the handle to "switchTo().window()".

Q: Does WebDriver support Javascript alerts and prompts?

A: Yes, using the Alerts API:

```
// Get a handle to the open alert, prompt or confirmation
Alert alert = driver.switchTo().alert();
```

```
// Get the text of the alert or prompt
alert.getText();
// And acknowledge the alert (equivalent to clicking "OK")
alert.accept();
```

Q: Does WebDriver support file uploads?

A: Yes.

You can't interact with the native OS file browser dialog directly, but we do some magic so that if you call WebElement#sendKeys("/path/to/file") on a file upload element, it does the right thing. Make sure you don't WebElement#click() the file upload element, or the browser will probably hang.

Handy hint: You can't interact with hidden elements without making them un-hidden. If your element is hidden, it can probably be un-hidden with some code like:

```
((JavascriptExecutor)driver).executeScript("arguments[0].style.visibility = 'visible'; argum
```

Q: The "onchange" event doesn't fire after a call "sendKeys"

A: WebDriver leaves the focus in the element you called "sendKeys" on. The "onchange" event will only fire when focus leaves that element. As such, you need to move the focus, perhaps using a "click" on another element.

Q: Can I run multiple instances of the WebDriver sub-classes?

A: Each instance of an HtmlUnitDriver, ChromeDriver and FirefoxDriver is completely independent of every other instance (in the case of firefox and chrome, each instance has its own anonymous profile it uses). Because of the way that Windows works, there should only ever be a single InternetExplorerDriver instance at one time. If you need to run more than one instance of the InternetExplorerDriver at a time, consider using the Remote!WebDriver and virtual machines.

Q: I need to use a proxy. How do I configure that?

A: Proxy configuration is done via the `org.openqa.selenium.Proxy` class like so:

```
Proxy proxy = new Proxy();
proxy.setProxyAutoconfigUrl("http://youdomain/config");

// We use firefox as an example here.
DesiredCapabilities capabilities = DesiredCapabilities.firefox();
capabilities.setCapability(CapabilityType.PROXY, proxy);

// You could use any webdriver implementation here
WebDriver driver = new FirefoxDriver(capabilities);
```

Q: How do I handle authentication with the HtmlUnitDriver?

A: When creating your instance of the HtmlUnitDriver, override the "modifyWebClient" method, for example:

```
WebDriver driver = new HtmlUnitDriver() {
  protected WebClient modifyWebClient(WebClient client) {
    // This class ships with HtmlUnit itself
    DefaultCredentialsProvider creds = new DefaultCredentialsProvider();

    // Set some example credentials
    creds.addCredentials("username", "password");

    // And now add the provider to the webClient instance
    client.setCredentialsProvider(creds);

    return client;
  }
};
```

Q: Is WebDriver thread-safe?

A: WebDriver is not thread-safe. Having said that, if you can serialise access to the underlying driver instance, you can share a reference in more than one thread. This is not advisable. You /can/ on the other hand instantiate one WebDriver instance for each thread.

Q: How do I type into a contentEditable iframe?

A: Assuming that the iframe is named "foo":

```
driver.switchTo().frame("foo");
WebElement editable = driver.switchTo().activeElement();
editable.sendKeys("Your text here");
```

Sometimes this doesn't work, and this is because the iframe doesn't have any content. On Firefox you can execute the following before "sendKeys":

```
((JavascriptExecutor) driver).executeScript("document.body.innerHTML = '<br>'");
```

This is needed because the iframe has no content by default: there's nothing to send keyboard input to. This method call inserts an empty tag, which sets everything up nicely.

Remember to switch out of the frame once you're done (as all further interactions will be with this specific frame):

```
driver.switchTo().defaultContent();
```

Q: WebDriver fails to start Firefox on Linux due to java.net.SocketException

A: If, when running WebDriver on Linux, Firefox fails to start and the error looks like:

```
Caused by: java.net.SocketException: Invalid argument
        at java.net.PlainSocketImpl.socketBind(Native Method)
        at java.net.PlainSocketImpl.bind(PlainSocketImpl.java:365)
        at java.net.Socket.bind(Socket.java:571)
        at org.openqa.selenium.firefox.internal.SocketLock.isLockFree(SocketLock.java:99)
        at org.openqa.selenium.firefox.internal.SocketLock.lock(SocketLock.java:63)
```

It may be caused due to IPv6 settings on the machine. Execute:

```
sudo sysctl net.ipv6.bindv6only=0
```

To get the socket to bind both to IPv6 and IPv4 addresses of the host with the same calls. More permanent solution is disabling this behaviour by editing /etc/sysctl.d/bindv6only.conf

Q: WebDriver fails to find elements / Does not block on page loads

A: This problem can manifest itself in various ways:

- Using WebDriver.findElement(...) throws ElementNotFoundException, but the element is clearly there - inspecting the DOM (using Firebug, etc) clearly shows it.
- Calling Driver.get returns once the HTML has been loaded - but Javascript code triggered by the onload event was not done, so the page is incomplete and some elements cannot be found.
- Clicking on an element / link triggers an operation that creates new element. However, calling findElement(s) after click returns does not find it. Isn't click supposed to be blocking?
- How do I know when a page has finished loading?

Explanation: WebDriver has a blocking API, generally. However, under some conditions it is possible for a get call to return before the page has finished loading. The classic example is Javascript starting to run after the page has loaded (triggered by onload). Browsers (e.g. Firefox) will notify WebDriver when the basic HTML content has been loaded, which is when WebDriver returns. It's difficult (if not impossible) to know when Javascript has finished executing, since JS

code may schedule functions to be called in the future, depend on server response, etc. This is also true for clicking - when the platform supports native events (Windows, Linux) clicking is done by sending a mouse click event with the element's coordinates at the OS level - WebDriver cannot track the exact sequence of operations this click creates. For this reason, the blocking API is imperfect - WebDriver cannot wait for all conditions to be met before the test proceeds because it does not know them. Usually, the important matter is whether the element involved in the next interaction is present and ready.

Solution: Use the Wait class to wait for a specific element to appear. This class simply calls findElement over and over, discarding the NoSuchElementException each time, until the element is found (or a timeout has expired). Since this is the behaviour desired by default for many users, a mechanism for implicitly-waiting for elements to appear has been implemented. This is accessible through the WebDriver.manage().timeouts() call. (This was previously tracked on issue 26).

Q: How can I trigger arbitrary events on the page?

A: WebDriver aims to emulate user interaction - so the API reflects the ways a user can interact with various elements.

Triggering a specific event cannot be achieved directly using the API, but one can use the Javascript execution abilities to call methods on an element.

Q: Why is it not possible to interact with hidden elements?

A: Since a user cannot read text in a hidden element, WebDriver will not allow access to it as well.

However, it is possible to use Javascript execution abilities to call getText directly from the element:

```
WebElement element = ...;
((JavascriptExecutor) driver).executeScript("return arguments[0].getText();", element);
```

Q: How do I start Firefox with an extension installed?

A:

```
FirefoxProfile profile = new FirefoxProfile()
profile.addExtension(....);

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

Q: I'd like it if WebDriver did....

A: If there's something that you'd like WebDriver to do, or you've found a bug, then please add an add an issue to the WebDriver project page.

Q: Selenium server sometimes takes a long time to start a new session ?

A: If you're running on linux, you will need to increase the amount of entropy available for secure random number generation. Most linux distros can install a package called "randomsound" to do this.

On Windows (XP), you may be running into http://bugs.sun.com/view_bug.do?bug_id=6705872, which usually means clearing out a lot of files from your temp directory. temp directory.

Q: What's the Selenium WebDriver API equivalent to TextPresent ?

A:

```
driver.findElement(By.tagName("body")).getText()
```

will get you the text of the page. To verifyTextPresent/assertTextPresent, from that you can use your favourite test framework to assert on the text. To waitForTextPresent, you may want to investigate the WebDriverWait class.

Q: The socket lock seems like a bad design. I can make it better

A: the socket lock that guards the starting of firefox is constructed with the following design constraints:

- It is shared among all the language bindings; ruby, java and any of the other bindings can coexist at the same time on the same machine.
- Certain critical parts of starting firefox must be exclusive-locked on the machine in question.
- The socket lock itself is not the primary bottleneck, starting firefox is.

The SocketLock is an implementation of the Lock interface. This allows for a pluggable strategy for your own implementation of it. To switch to a different implementation, subclass the FirefoxDriver and override the "obtainLock" method.

Q: Why do I get a UnicodeEncodeError when I send_keys in python

A: You likely don't have a Locale set on your system. Please set a `locale` LANG=en_US.UTF-8 and LC_CTYPE="en_US.UTF-8" for example.
