Automated Web Testing with Selenium

Erik Doernenburg
ThoughtWorks

Agenda

• What is Selenium?
• Writing Maintainable Tests
What is Selenium?

- Test tool for web applications
- Runs in any mainstream browser
- Supports tests in many languages
  - Selenese (pure HTML, no backend required)
  - Java, C#, Perl, Python, Ruby
- Record/playback (Selenium IDE)
- Open Source with corporate backing
- Lives at selenium.openqa.org

Demo

- Record a test in Selenium IDE
- Show same test written in Java
Java Test example

```java
public void testGoogleTestSearch() throws Exception {
    selenium.open("http://www.google.com/webhp");
    assertEquals("Google", selenium.getTitle());
    selenium.type("q", "Selenium OpenQA");
    selenium.click("btnG");
    selenium.waitForPageToLoad("5000");
    assertEquals("Selenium OpenQA - Google Search",
                selenium.getTitle());
}
```

Java SetUp/TearDown example

```java
public void setUp() throws Exception {
    selenium = new DefaultSelenium(
            "localhost", 4444, "*chrome",
            "http://www.google.com");
    selenium.start();
}

public void tearDown() throws Exception {
    selenium.stop();
}
```
A few Selenese commands

<table>
<thead>
<tr>
<th>click</th>
<th>getHtmlSource</th>
<th>isVisible</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>getTitle</td>
<td>keyPress</td>
</tr>
<tr>
<td>createCookie</td>
<td>getValue</td>
<td>mouseOver</td>
</tr>
<tr>
<td>dragdrop</td>
<td>goBack</td>
<td>open</td>
</tr>
<tr>
<td>fireEvent</td>
<td>isElementPresent</td>
<td>refresh</td>
</tr>
<tr>
<td>getEval</td>
<td>isTextPresent</td>
<td>type</td>
</tr>
</tbody>
</table>

Element locators

- **ID**: id=foo
  - `selenium.click("btnG");`
- **Name**: name=foo
- **First ID, then name**: identifier=foo
- **DOM**: `document.forms['myform'].myDropdown`
- **XPath**: `xpath=//table[@id='table1']//tr[4]/td[2]`
- **Link Text**: link=sometext
- **CSS Selector**: `css=a[href="#id3"]`
- Sensible defaults, e.g. xpath if starts with `//`
How Selenium works

Agenda

• What is Selenium?
• Writing Maintainable Tests
Standard end-user black-box test

1. Login as administrator
2. Create a user
3. Log out
4. Login as that user
5. Create a folder
6. Create a thingy in that folder
7. Search for that thingy in the search box
8. Make sure your thingy shows up on the search results page

Fragile Automated Tests

- Exercising irrelevant features
  - Logging in/Logging out
  - Creating a folder
  - Creating a thingy
- If the UI for any one of those features changes, your search test fails
Know when to record tests

- Recorded tests reuse no code
- “Record & Tweak” vs. “Fire and Forget”
- Slight change in folder creator page means all of those tests have to be re-recorded from scratch
- Use the recorder to create reusable code

Unit testing vs. Integration testing

- Selenium tests are integration tests
  - Functional/Acceptance/User/Compatibility
- Unit tests verify a unit in isolation
  - If FooTest.java fails, the bug must be in Foo.java
  - Cannot fail due to browser incompatibilities
  - Must be completely isolated from each other
- Integration tests verify that units work together
  - Requires testing multiple configurations (browsers)
  - Tend to build on the side-effects of earlier tests
**Presentation Model**

- Create a layer of classes that mimic the UI
  - a field for every text box, etc.
  - a method for every button
- Test the application flow using this model
  - Can use normal unit test framework
  - Insulated from design changes
- Use Selenium to check wiring and browser compatibility

**Create abstractions**

- Tests can use all language features
  - extract method, inheritance, ...

```java
public void testSearchForThingy()
{
    createTestUserAndLogin();
    createThingyInFolder("Foo", "Test Folder");
    searchForThingy("Foo");
    assertTrue(isPresentInResultList("Foo"));
}
```

- Re-use makes tests less fragile
Use your code directly

• Prepare your search tests using model API

```java
FolderBean fb = new FolderBean();
fb.setParent(FolderBean.ROOT);
fb.setName("foo");
fb.createNewFolder(); // adds a folder to the DB

selenium.open("/search");
selenium.type("query", "foo");
selenium.click("search");
assertTrue(selenium.isTextPresent("foo found");
```

• Your tests and web app are written in same language...

A class per test vs. A class per page

• Do create a class for each test
  • this inherits from TestCase
  • contains the ‘flow’ of the test

• If the same pages are used by multiple tests
  • create a separate hierarchy of classes, one per page
  • inject the test into the page to access Selenium

```java
public void testWorkflow() {
    WelcomePage welcomePage = new WelcomePage(this);
    welcomePage.selectDailyView();
    DailyView dailyView = new DailyView(this);
    dailyView.selectLocation("LDN");
    dailyView.clickOk();
```
JUnit vs. TestNG

- JUnit is “opinionated software”
- Dependencies between tests are explicitly prevented
- Separate tests are testing separate units
- Everything gets torn down after every test method
- Constantly starting/stopping browser

TestNG has `dependsOnX`

```java
public void setUp() {
  log("setup");
}

public void testFoo() {
  log("foo");
}

public void testBar() {
  log("bar");
}

public void tearDown() {
  log("teardown");
}
```

Summary

- Use Selenium when it makes sense
  - when you want to reproduce a user’s interaction with your application in a real web browser
  - when you depend on the browser (AJAX)
- Do consider presentation model and HTTPUnit
- Use Selenium for Integration Testing
- Use Selenium in your development environment
- Use the features offered by your language