HttpUnit - Are You Getting the Right Response?

Advanced Topics in Java

Khalid Azim Mughal
khalid@ii.uib.no
http://www.ii.uib.no/~khalid/atij/

Version date: 2006-09-04

Overview

- Installing HttpUnit
- Functional Testing with HttpUnit
- Creating Test Scenarios using HttpUnit
- Example illustrating a Test Scenario

Installing HttpUnit

- HttpUnit can be downloaded from: `http://httpunit.sourceforge.net/`
- Unpack the distribution zip file there you want to install HttpUnit.
- An environment variable, called `HTTPUNIT_HOME`, with the following entry can facilitate using HttpUnit:
  ```
  HTTPUNIT_HOME = C:\httpunit-1.6
  ```
- The `CLASSPATH` environment variable should include the following entries in its definition:
  ```
  %HTTPUNIT_HOME%\lib\httpunit.jar;%HTTPUNIT_HOME%\jars\js.jar;%HTTPUNIT_HOME%\jars\nekohtml.jar;%HTTPUNIT_HOME%\jars\servlet.jar;%HTTPUNIT_HOME%\jars\Tidy.jar;%HTTPUNIT_HOME%\jars\xercesImpl.jar;%HTTPUNIT_HOME%\jars\xmlParserAPIs.jar
  ```
- Add the above libraries to the Eclipse project containing the test cases.

Testing a Web Site with HttpUnit

- *Functional tests* are performed to test that a web site/application is providing the correct response to client requests.
- HttpUnit is a framework, based on JUnit, that can be used for functional testing (aka *block-box testing*) of a web application.
  - It emulates the web-client interaction with a web application.
  - It is a *HTTP client simulator* that:
    - simulates a web browser's GET and POST requests, and maintains state
    - provides an object model of the response that can be verified by the test cases.
      - The entity body of the response is primarily HTML content.
  - Each test case describes a *scenario* (in a use case) that can be accomplished by a user while interacting with the web application through a web browser.
- Note that Swing and AWT applications are not suitable for testing with HttpUnit.
  - For this purpose, see *jfcUnit* (*http://jfcunit.sourceforge.net/*)

Implementing Test Cases with HttpUnit

- Main functionality is provided by the package `com.meterware.httpunit`
- Setting up a test session:
  1. Create a `WebConversation` object which communicates with the server.
  2. Submit an initial http request using the `getResponse()` method.
  3. Examine the response (`WebResponse`) either textually (using the `getText()` method), as a DOM object (using the `getDOM()` method), or by using response-element specific methods.
  4. In necessary, create a new request based on either submitting a form (`WebForm`) or clicking on a link (`WebLink`) -- and continue with step 3.

Web Application under Test

- The scenario to be tested (see the horocope application):

The Responses

From index.html

```
...
<TABLE>
  <TBODY>
    ...
    <TR><TD> <A href="SimpleHoroscopeII.html">
             Simple Horoscope II (tested with HttpUnit)</A>
    </TD></TR>
    ...
  </TBODY>
</TABLE>
...
```

From SimpleHoroscopeII.html

```
...
<form name="signs"
      action="SimpleHoroscopeII"
      method="post">
  <input type="radio" name="sign" value="aquarius'
         checked="checked"/>
         Aquarius (Jan 21 - Feb 19) <br/>
  ...
  <input type="radio" name="sign" value="libra"/>
         Libra (Sept 24 - Oct 23) <br/>
  ...
  <input type="submit" value="Read the stars!"/>
</form>
...
```

From servlet SimpleHoroscopeII

```
...
<tr>
  <td>Sign</td>
  <td>Horoscope</td>
</tr>
<tr>
  <td>LIBRA</td>
  <td>If you think knowing your horoscope
      is going to help you, you are gravely mistaken.</td>
</tr>
</table>
...
```

The Test Case Scenario Class

- Class `HoroTestingWithHttpUnit` implements a test case scenario.
- Each scenario is implemented as a *unit test* in the JUnit framework.

```
// HttpUnit uses JUnit
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
// Important classes in HttpUnit
import com.meterware.httpunit.WebConversation;
import com.meterware.httpunit.WebForm;
import com.meterware.httpunit.WebLink;
import com.meterware.httpunit.WebRequest;
import com.meterware.httpunit.WebResponse;
import com.meterware.httpunit.WebTable;
```

```
public class HoroTestingWithHttpUnit extends TestCase {

  final static String horo_url = "http://localhost:8080/horoscope/"; // The web-app

  // Create a test suite.
  public static Test suite() { return new TestSuite(HoroTestingWithHttpUnit.class); }

  // Running the unit tests
  public static void main(String[] args) {
    junit.textui.TestRunner.run(suite()); // Text-based user interface.
  }

  public HoroTestingWithHttpUnit(String methodName) { super(methodName); }
  ...
  // The actual "unit test" for a scenario.
  public void testHoroLookup() throws Exception { ... }
}
```

Implementing the Test Case Scenario

- See the implementation of the `testHoroLookup()` method in the class `HoroTestingWithHttpUnit`.

1. Create a `WebConversation` object which communicates with the server.
   - A request (`WebRequest`) is supplied to the `WebConversation` object.
   - It submits the request to the remote server, and returns any response (`WebResponse`) it receives from the server.

```
WebConversation wc = new WebConversation();
WebRequest request = new GetMethodWebRequest(horo_url); // (1)
WebResponse response = wc.getResponse(request);         // (2)
```

   - Lines (1) and (2) can be replaced by the following code:

```
WebResponse response = wc.getResponse(horo_url);
```

2. Manipulating the response and asserting facts.
   - The `WebResponse` object can be inspected for the *HTTP response* it represents.
   - Specific methods can be used to inspect the *HTML elements* (links, tables, forms, frames) on the virtual web page response.
   - Facts about the HTML content can be verified using the `assert`*SomeFact*`()` methods in the JUnit framework.

```
assertEquals("No. of links should be 9", 9, response.getLinks().length);
```

3. Following a HTML Link.
   - Links in a response can be retrieved as `WebLink` objects and "clicked" to obtain a new web page response.
   - Allows the navigation on a web site to be tested.

```
// Find the link which contains the string
// "Simple Horoscope II (tested with HttpUnit)"
WebLink httpunitLink = response.getFirstMatchingLink(
    WebLink.MATCH_CONTAINED_TEXT, "Simple Horoscope II" +
    "(tested with HttpUnit)");
// Click the link to obtain the response
httpunitLink.click();
response = wc.getCurrentPage();
```

4. Manipulating a HTML Form.
   - The `WebResponse` class provides the method `getForms()` to retrieve all the forms in the order they appear in a web page.
   - A HTML form (`WebForm`) can be checked for its controls and default values.
   - Form parameters can also be set and the form submitted to get a new response.

```
// Simulate a lookup on the form in the page
WebForm form = response.getForms()[0]; // Get the first form.
assertEquals("Default checked value is aquarius", "aquarius",
             form.getParameterValue("sign"));

// Setting form parameters and submitting the form
form.setParameter("sign", "libra");
form.submit();

// Can now examine the new response
response = wc.getCurrentPage();  // The response
```

5. Using a HTML Table.
   - The `WebResponse` class provides the method `getTables()` to retrieve all top-level tables in the order they appear in a web page.
   - A table (`WebTable`) can also be retrieved by the content in the first non-empty cell and also by its ID attribute.
   - A `WebTable` can be verified for its dimensions and its cell content.
   - An entire `WebTable` can be converted to a two-dimensional array of `String` objects, ignoring all formatting tags:

     ```
     String[][] strCells = response.getTableStartingWith("Sign").asText();
     ```

   - Each cell in a `WebTable` can be treated as a `String` object or a DOM object, and any nested HTML element in the cell can be retrieved and examined.

     ```
     // Check information in a HTML table.
     WebTable wt = response.getTables()[0];  // Get the first table
     assertEquals("No. of rows", 2, wt.getRowCount());
     assertEquals("No. of columns", 2, wt.getColumnCount());
     assertEquals("Wrong star sign", "LIBRA", wt.getTableCell(1,0).getText());
     assertNotNull("Not empty", wt.getTableCell(1,1).getText());
     ```