Webアプリケーション用のテスト・ケースを作る

jWebUnitフレームワークでWebアプリケーションのテストが容易に

Amit Tuli
Software Engineer
IBM India Software Labs

2005年 5月 31日

Web開発用の自動テストをお探しですか？ 走り回って探す必要はありません。大部分のJava™ IDEに簡単にプラグインできるjWebUnitは、Webアプリケーション用のテスト・ケースを作るための、オープンソースのフレームワークです。この記事では、ソフトウェア技術者のAmit Tuliが、jWebUnitを紹介します。サンプル・アプリケーションを使いながら、簡潔なテスト・ケースを生成する方法を、順を追って説明します。

同じテスト・ステップを繰り返し行う場合、自動テストがあると、時間と手間が節約できます。この記事では、Webアプリケーション用のテスト・ケース開発に利用できるJavaクラス・セット、jWebUnitを紹介します。jWebUnitはオープンソースのプロジェクトであり、BSDライセンスの下で、自由に使用することができます。この記事では、jWebUnitライブラリーのダウンロード方法やjWebUnitテスト・ケースを開発するためのEclipseプラットフォームの構成方法、サンプル・テスト・ケースの構築方法などを説明します。

jWebUnitの紹介

jWebUnitは、Webアプリケーション自動テスト用のJavaライブラリーであるHttpUnitと、JUnitユニット・テスト・フレームワークに基づいています（参考文献を見てください）。jWebUnitは、Webアプリケーションのナビゲーション用に高位レベルのAPIを備えており、また、ナビゲーションの正しさをリンク経由で検証するための一連のアサーション、フォーム入力や送信、テーブル内容その他、ビジネス用Webアプリケーションに通常必要とされる機能も備えています。jWebUnitはJARファイルの形で提供されるため、大部分のIDEに容易にプラグインできますが、その他にも、必要とされるライブラリーを含んでいます。

HttpUnitでテストする

Webアプリケーションを自動化するということは、Webブラウザーをバイパスし、プログラムを通してWebサイトを操作することを意味します。まず、JWebUnitの構成ブロックであるHttpUnitによって、これが単純にできることを示しましょう。HttpUnitは、フレームやJavaScript、ページ・リダイレクション・クッキーなどをエミュレートすることができます。HttpUnitはJUnitと組み合わされると、Webサイトが動作しているかどうかを素早く検証することができます。

リスト1は、HttpUnitホームページにある「Cookbook」リンクをクリックするための、HttpUnitで書いたサンプル・テスト・ケースです。

リスト1. HttpUnitホームページにあるCookbookリンクをクリックするHttpUnitコード

```
1 public class HttpUnitTest {
2  public static void main(String[] args) {
3    try {
4      WebConversation wc = new WebConversation();
5      WebRequest request =
            new GetMethodWebRequest("http://httpunit.sourceforge.net/index.html");
6      wc.setProxyServer( "your.proxy.com", 80 );
7      WebResponse response = wc.getResponse(request);
8      WebLink httpunitLink =
            response.getFirstMatchingLink(WebLink.MATCH_CONTAINED_TEXT,"Cookbook");
9      response = httpunitLink.click();
10      System.out.println("Test successful !!");
11    } catch (Exception e) {
12      System.err.println("Exception: " + e);
13    }
14  }
15 }
```

リスト1のコードは、`your.proxy.com`（6行目）を使ってインターネットに接続します。インターネットに直接接続できる場合には、このステートメントはコメント行にすることができます。8行目のステートメントは、Cookbook という文字列を含む、Webリンク用のページを検索します。9行目のステートメントは、そのリンクをクリックします。リンクが見つかると、`Test Successful !!`というメッセージが表示されます。

jWebUnitを使った、さらに簡単なテスト

リスト2のテスト・ケースは、jWebUnit APIを使って、リスト1と同じことを行います。

リスト2. HttpUnitホームページにあるCookbookリンクをクリックするjWebUnitコード

```
1 public class JWebUnitTest extends WebTestCase{
2  public static void main(String[] args){
3    junit.textui.TestRunner.run(new TestSuite(JWebUnitTest.class));
4  }
5  public void setUp(){
6    getTestContext().setBaseUrl("http://httpunit.sourceforge.net");
7    getTestContext().setProxyName("webproxy.watson.ibm.com");
8    getTestContext().setProxyPort(8080);
9  }
10  public void testSearch(){
11    beginAt("/index.html");
12    clickLinkWithText("Cookbook");
13  }
14 }
```

リスト2で、JUnit特有のコードを無視すれば、テスト・ケースが非常にきれいに、簡潔になっていることが分かるでしょう。注目すべき重要な行は、6行目、11行目、12行目です。6行目では、ベースURLがHttpUnitのホームページに設定されています。11行目は、相対パス`/index.html`にあるサイトに接続します。12行目は、そのページにある、文字列Cookbookを持つリンクをクリッ

クします。クリックが動作すれば成功をレポートし、動作しない場合には、例外をレポートします。

jWebUnit APIを詳しく見る

どのjWebUnitテスト・ケースにも、その中心には、テスト・ケースを表すnet.sourceforge.jwebunit.WebTestCaseクラスがあります。テスト・ケースはすべて、このクラスから継承する必要があります。（net.sourceforge.jwebunit.WebTestCaseクラス自体は、JUnitでテスト・ケースを表現するjunit.framework.TestCaseから継承されています。）表1は、このクラスにあるメソッドのうち、一般的に使われるものを示しています。

表1. net.sourceforge.jwebunit.WebTestCaseクラスの中の重要メソッド

メソッド	説明
public TestContext getTestContext()	テスト・ケースのコンテキストを取得します。これを使って、ロケールやベースURL、クッキーなどのアイテムにアクセスすることができます。
public void beginAt(String relativeURL)	ベースURLに対する相対URLで、会話を開始します。
public void setWorkingForm(String nameOrId)	規定されたフォームとのやり取りを開始します。現在のページに1つしかフォームがない場合には、このメソッドを呼ぶ必要はありません。
protected void submit()	フォームを送信します。フォームのSubmitボタンをクリックすることと同等です。
public void gotoFrame(String frameName)	指定された名前のフレームをアクティブにします。

もう1つ重要なクラスは、net.sourceforge.jwebunit.TestContextです。このクラスは、テスト用のコンテキストを確立します。このクラスを使って、クッキーやセッション、認証などの情報を操作することができます。表2は、このクラスにあるメソッドのうち、重要なものの幾つかを示しています。

表2. net.sourceforge.jwebunit.TestContextクラスの中の重要メソッド

メソッド	説明
public void addCookie(String name, String value)	テスト・コンテキストにクッキーを追加します。HttpUnitDialogが開始されると、追加されたクッキーはWebConversationに対して設定されます。
public void setResourceBundleName(String name)	テスト・コンテキスト用に使うリソース・バンドルを設定します。期待される値を参照するために、WebTesterでのキーで使われます。
public void setProxyName(String proxyName)	テスト・コンテキスト用のプロキシー・サーバー名を設定します。
public void setBaseUrl(String url)	テスト・コンテキスト用のベースURLを設定します。

EclipseでjWebUnitをダウンロードし、設定する

jWebUnitは純粋なJavaコードとして実装されているため、JARファイルとして入手できます（ダウンロード・リンクについては参考文献を見てください）。ダウンロードしたら、次のステップに従って、Eclipseプラットフォーム上でjWebUnitライブラリーを設定します。

1. ダウンロードしたファイル（jwebunit-1.2.zip）を、一時ディレクトリー（ここでは、C:\tempとします）で解凍します。
2. Eclipseで、jWebUnitという名前の新しいJavaプロジェクトを作ります。

3. Package ExplorerビューでjWebUnitプロジェクトを右クリックし、Propertiesを選択します。
4. Java Build Pathをクリックします。Librariesタブで、Add External JARsをクリックします。
5. C:\temp\jwebunit-1.2\libディレクトリーに行き、このディレクトリーにある、全てのJARファイルを選択します。
6. OKをクリックします。

これで、jWebUnitというプロジェクトの下に、EclipseでjWebUnitテスト・ケースの開発を開始することができます。

サンプル・アプリケーションを作る

今度は、jWebUnit APIの実際の動作を見る番です。jWebUnitの真の力をよく理解できるように、順を追ってサンプル・アプリケーションを説明しましょう。このアプリケーションは、Googleの検索ページを開き、文字列HttpUnitを検索するテスト・ケースです。このアプリケーションで、次のようなシナリオをテストします。

- Googleのホームページ、http://www.google.comを開きます。
- ページは名前`q`を持つフォーム要素を含む、とアサートします。（Googleのホームページでは、`q`という名前のテキスト・ボックスは、ユーザー・クエリーを入力とするものを意味します。）アプリケーションは、この要素を使って検索基準を入力します。
- 検索テキスト・ボックスに文字列`HttpUnit Home`を入力し、フォームを送信します。
- 結果ページを取得し、そのページが文字列HttpUnit Homeを含むリンクを含んでいるとアサートします。
- 文字列HttpUnit Home上にあるリンクをクリックします。

テスト・シナリオの準備ができたので、jWebUnitを使ってこうした要求事項を実装する、Javaアプリケーションを書きます。

最初のステップは、リスト3に示すように、`WebTestCase`から継承するクラスを宣言することです。

リスト3. testcaseクラスを宣言する

```
public class GoogleTest extends WebTestCase {
   static String searchLink = '';
}
```

先に触れたように、jWebUnitは、全てのテスト・ケースが`WebTestCase`から継承するように要求します。`searchLink`は、検索用に渡された引数を保管します。この値を、コマンドライン・パラメーターとして、テスト・ケースに渡します。

次のステップは、エントリー・ポイント、つまり`main()`メソッド、を宣言します。これをリスト4に示します。

リスト4. `main()`メソッド

```
public static void main(String[] args) {
   searchLink = args[0];
   junit.textui.TestRunner.run(new TestSuite(GoogleTest.class));
}
```

`main()`メソッドは、`junit.textui.TestRunner.run()`を呼ぶことによって、`JTest`テスト・ケースを実行します。ここでは`GoogleTest`テスト・ケースを実行する必要があるため、パラメーターとして`run()`メソッドに渡すテスト・スイートは、引数として`GoogleTest.class`を取ります。

`JTest`は次に`setUp()`メソッドを呼び、ベースURLとプロキシーを設定します。これをリスト5に示します。

リスト5. Setup

```
public void setUp() {
   getTestContext().setBaseUrl("http://www.google.com");
   getTestContext().setProxyName("proxy.host.com");
   getTestContext().setProxyPort(80);
}
```

リスト5は、ベースURLを`http://www.google.com`に設定します。これによって、このURLを相対的な基準としてテスト・ケースが開始することになります。次の2つのステートメントは、インターネットに接続するためのプロキシー・ホストと、プロキシー・ポートを設定します。直接インターネットに接続している場合には、プロキシー設定のステートメントを省略することができます。

今度は、サイトのブラウズと、サーチ基準の入力を始めます。リスト6は、Webページにアクセスし、全てのシナリオをテストするためのコードです。

リスト6. シナリオをテストする

```
public void testSearch() {
   beginAt("/");
   assertFormElementPresent("q");
   setFormElement("q", "HttpUnit");
   submit("btnG");
   assertLinkPresentWithText(searchLink);
   clickLinkWithText(searchLink);
}
```

リスト6のコードは、ベースURLに接続し、`/`を相対基準としてブラウズを開始します。次に、このページが`q`という名前のフォーム要素を含むことをアサートします（qは、Googleのホームページ上の、クエリー入力テキスト・ボックスの名前です）。次のステートメントは、`q`という名前を持つテキスト・ボックスを、値`HttpUnit`に設定します。その次のステートメントは、`btnG`という名前のフォーム送信ボタンを送信します。（Googleのホームページでは、`btnG`という名前のボタンは、Google Searchというラベルの付いたボタンです。）この会話でフォームは送信され、その次のページで検索結果がリストアップされます。結果ページでは、コードはまず、文字列HttpUnit Home を含むリンクがあるかどうかをチェックします。リンクがない場合には、テストは`AssertionFailedError`でフェールします。リンクがある場合には、テストはそのリンクをクリックします。

サンプル・アプリケーションを実行する

では、サンプル・アプリケーションを動かしてみましょう。

1. サンプル・アプリケーション、j-webunitsample.jarをダウンロードします（ダウンロードのセクションを見てください）。

2. どこかのディレクトリーでj-webunitsample.jarを解凍します。例えばC:\tempで解凍すると、ソースファイルとクラスファイルはC:\temp\com\jweb\testに置かれ、そしてsetclasspth.batがC:\tempに置かれます。
3. setclasspath.batを編集します。`JAR_BASE`を、必要なJAR全てを含むディレクトリーを指すように設定します。例えば、C:\tempの下でjwebunit-1.2.zipファイルを解凍した場合は、JAR_BASEをC:\temp\jwebunit-1.2\libに設定します。
4. コマンド・プロンプトを開き、ディレクトリーをC:\tempに変更します。
5. `setclasspath.bat`を実行します。これによって、テスト・ケースを実行するために必要な`CLASSPATH`が設定されます。
6. アプリケーションを、コマンド、`java com.jweb.test.GoogleTest "HttpUnit Home"`で実行します。

サンプル出力

テスト・ケースを実行すると、コマンド・プロンプトにテスト・ケース・レポートが出力されます。テストが失敗すると、レポートはリスト7のようなものになります。

リスト7. アサーションがフェールした場合の出力

```
C:\temp>java com.jweb.test.GoogleTest "HttpUnit Hwee"
.F
Time: 5.338
There was 1 failure:
1) testSearch(com.jweb.test.GoogleTest)junit.framework.AssertionFailedError: Link
  with text [HttpUnit Hwee] not found in response.
at net.sourceforge.jwebunit.WebTester.assertLinkPresentWithText(WebTester.java:618)
at net.sourceforge.jwebunit.WebTestCase.assertLinkPresentWithText(WebTestCase.java:244)
        at com.jweb.test.GoogleTest.testSearch(GoogleTest.java:36)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at com.jweb.test.GoogleTest.main(GoogleTest.java:19)

FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0
```

リスト7を見ると分かる通り、テスト・ケースは`HttpUnit Hwee`をパラメーターとして実行されています。このテスト・ケースはアサーションでフェールしますが、これは結果ページに、この文字列を持つリンクが無いためです。従って`junit.framework.AssertionFailedError`となります。

リスト8は、`HttpUnit Home`をパラメーターとして実行します。テスト・ケースは、この文字列を持つリンクを見つけるので、テストをパスし、次を出力します。

リスト8. 成功したテストの出力

```
C:\temp>java com.jweb.test.GoogleTest "HttpUnit Home"
.
Time: 6.991

OK (1 test)
```

まとめ

この記事では、jWebUnitフレームワークの注目すべき機能や、最も重要なクラスなどを議論しながら、簡単なテスト・ケースの作り方を通して、jWebUnitフレームワークの一端を簡単に紹介

しました。jWebUnitには、テスト・ケースで使えるような機能が、他にも数多くあります。例えば、Webページ上にあるリンクの数をカウントすることができます。Webページの中に文字列やテーブル、指定のラベルを持つフォーム入力要素などがあるかどうかをアサートすることができます。クッキーがあるとアサートしたりクッキーを削除したりするなど、クッキーをいじることもできます。あるいは、別の文字列の後に現れる特定な文字列に対するリンクを、テストにクリックさせることもできます。Webアプリケーション用のテスト・ケースを、素早く効率良く構築したい場合には、jWebUnitが最適だと言えるでしょう。

ダウンロード

内容	ファイル名	サイズ
Sample code	j-webunitsample.jar	2 KB

著者について

Amit Tuli

Amit Tuli は、インドの Gurgaon にある、IBM India Software Labs (ISL) の staff software engineer です。現在は、ISL の Solutions Group での IBM Websphere Business Integration に関する業務を行っています。様々なプラットフォーム上での Java や、サーバー側プログラミングで 5 年間の技術経験があり、また DB2 UDB や Oracle を含むリレーショナル・データベース・システムに関する業務も行ってきました。IBM WebFountain SDK プロジェクトで、India Research Lab での業務にも携わったこともあります。専門領域は、スタンドアローンから n 層までの分散アプリケーションの設計開発です。彼は Hisar にある、Guru Jambheshwar University にて、コンピューター・サイエンスで修士号を取得しています。