**使用 Selenium 和 TestNG 进行编程式测试**

[开发者在线 Builder.com.cn](http://www.builder.com.cn) 更新时间:2007-08-31作者：Andrew Glover 来源:IBMDW

**本文关键词：** [**Selenium**](http://www.builder.com.cn/files/list-0-0-70444-1-1.htm)[**TestNG**](http://www.builder.com.cn/files/list-0-0-70442-1-1.htm)[**编程**](http://www.builder.com.cn/files/list-0-0-67355-1-1.htm)[**测试**](http://www.builder.com.cn/files/list-0-0-66516-1-1.htm)

Selenium 是一种 Web 测试框架，它搭建了验证 Web 应用程序的新途径。与大多数尝试模拟 HTTP 请求的 Web 测试工具不同，Selenium 执行 Web 测试时，就仿佛它本身就是浏览器。当运行自动的 Selenium 测试时，该框架将启动一个浏览器，并通过测试中描述的步骤实际驱动浏览器，用户将使用这种方式与应用程序交互。

由于开发人员和非开发人员都能够使用 Selenium 轻松地编写测试，使得它从众多测试框架应用程序中脱颖而出。在 Selenium 中，可以通过编程的方式编写测试，或者使用 Fit 样式的表，并且编写了测试后，可以使测试完全自动化。使用一个 Ant 构件（比方说）运行完整的 Selenium 套件非常简单，并且还可以在持续集成（Continuous Integration，CI）环境中运行 Selenium 测试。

这个月，我将介绍 Selenium，并逐一查看使它成为优秀 Web 测试框架的一些特性 —— 尤其是在结合使用 TestNG、DbUnit 和 Cargo 这样的软件时。

|  |  |  |
| --- | --- | --- |
| http://www.ibm.com/i/c.gif | |  | | --- | | **验收测试** 由于 Selenium 能够很好地模拟用户的行为，它常常用于进行验收测试，即在完成的系统上运行一整套测试。验收测试通常需要运行整个应用程序，以使测试发挥作用。如果您要测试一个 Web 应用程序，则需要访问应用程序数据库，以及一台 Web 服务器，一个容器和运行应用程序所需的任何配置元素。 | |

使用 Selenium 进行编程式测试

在 Selenium 中，您可以使用自己喜爱的语言或者 Fit 样式的表通过编程来编写测试。从测试的角度来说，不管使用什么语言，测试过程和结果都不会有显著的差别。在此，我希望研究 Selenium 的编程方法，因为在结合使用 TestNG 时，它提供了一些有趣的可行方法能性。

使用具有类似 TestNG 这样的框架的 Selenium 进行编程式测试具有这样一个优点，它允许您创建智能 fixture，而使用 Fit 样式的表则很难做到这一点。TestNG 尤其适合与 Selenium 结合使用，因为它使您能够完成其他框架无法做到的测试，例如使用依赖项进行测试，重新运行失败了的测试，以及使用单独文件中定义的参数进行参数化测试。所有这些特性结合在一起，当然能够使它在众多 Web 应用程序测试框架中脱颖而出，但是，正如您将看到的，在完全自动化的验收测试中使用这些特性令它更加出众。

**配置第一个测试**

Selenium 架构实际上由两个逻辑实体组成：您编写的代码以及能够简化与测试中的应用程序的交互的 Selenium 服务器。要成功地执行测试，必须要启动并运行 Selenium 服务器实例以及要测试的应用程序。（当然，测试结果取决于您编写的应用程序是否优秀！）

幸运的是，Selenium 服务器是一种轻量级程序，可以在实际的测试范围内通过编程启动和停止它。Selenium 服务器（使用 Selenium 对象嵌入）的启动和停止由一个 fixture 来执行。

要通过编程的方式启动 Selenium 服务器，必须创建一个新的 Selenium 对象，并告诉它要使用哪一种兼容的浏览器 —— 我在下面的示例中使用的是 Firefox。您还必须提供运行服务器实例的位置（通常是 localhost，但不是必须的），以及被测试的应用程序使用的基 URL。

在清单 1 中，我配置了一个本地 Selenium 实例，使用它在本地安装的 Web 应用程序上驱动 Firefox（http://localhost:8080/gt15/）。正如您从参数中推断的一样，Selenium 是作为被测试的应用程序的代理，并相应地促进测试。

**清单 1. 配置 SeleniumServer**

|  |
| --- |
| Selenium driver =  new DefaultSelenium("localhost", SeleniumServer.getDefaultPort(),  "\*firefox", "http://localhost:8080/gt15/");  driver.start();  //go to web pages and do stuff...  driver.stop(); |

创建了 Selenium 实例后，您可以 启动并在运行时 停止它。这意味着您可以通过编程与 Selenium 服务器交互，并通过一个测试程序使它驱动浏览器。

驱动应用程序

通过编程与 Web 页面进行交互是一种使用本地 id 的应用。（一些读者可能对这种源自 [本系列二月份关于 TestNG-Abbot 的文章](http://www.ibm.com/developerworks/cn/java/j-cq02277/index.html) 的概念比较熟悉）。与页面元素进行交互的第一步就是查找该元素，通常可以使用 HTML 元素 ID 进行查找。Selenium 还允许您使用 XPath、正则表达式，甚至是 JavaScript 来查找特定的元素（如果您希望这样做）。

清单 2 所示的 HTML 是使用 Groovlet 的简单 Web 应用程序的一部分。这段代码定义了包含输入和提交按钮的表单。如果希望 Selenium 与该表单交互，我必须为输入按钮提供 ID 以及相应的值。我还需要为提交按钮提供一个 ID，这样 Selenium 才能 “单击” 它。单击按钮后，表单将被提交给 Groovlet —— 本例中为 FindWidget.groovy。

**清单 2. 简单的 HTML 表单**

|  |
| --- |
| <form method=post action="./FindWidget.groovy">  <table border="0" style="border-style: dotted">  <tr>  <td class="heading">Widget:</td>  <td class="value"><input type="text" name="widget"></td>  </tr>  <tr>  <td></td>  <td class="value"><input type="submit" value="Find Description" name="submit"></td>  </tr>  </table>  </form> |

现在就可以通过使用 ID widget（输入值）和 submit（单击按钮）与该 HTML 表单进行编程式交互，如清单 3 所示：

**清单 3. 驱动简单的 Web 页面**

|  |
| --- |
| driver.type("widget", "pg98-01");  driver.click("submit");  driver.waitForPageToLoad("10000");  //assert some return value... |

Selenium 中用于和 Web 页面元素进行交互的 API 非常的直观。对于输入字段，我可以使用 type() 方法将值与 ID 关联起来。如果需要的话，可以通过编程 click 按钮。在清单 3 中，我将 click 设置为 10 秒的等待时间 —— 足够表单提交请求完成处理。当 FindWidget.groovy 中的代码运行其内容并返回响应后，我可以使用该响应来查找特定页面元素，并验证所有内容是否正常工作。

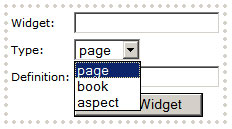
Selenium 和 TestNG

TestNG 以其灵活性和参数化 fixture 成为定义 Selenium 的驱动验收测试的首选。TestNG 能够定义测试依赖项并返回失败的测试，以及其易用性，使得 Selenium-TestNG 成为吸引人的组合。

让我们首先从一个能够允许用户创建、查找、更新或删除小部件的 Web 应用程序开始。创建一个小部件需要三个属性：名称、类型和定义。图 1 显示了创建小部件的表单：

**图 1. 创建小部件的 Web 表单**  


请注意：表单元素的类型是具有三个不同选项的下拉列表，如图 2 所示：

**图 2. 包含下拉列表的 Web 表单**  


单击 Create Widget 将促使 Groovlet 处理这一请求。如果所有内容正确的话（即名字和定义不为空，并且数据库中不存在该实例），Groovlet 将创建一个新的小部件实例并类似图 3 所示的状态页面：

**图 3. 返回的 Web 页面显示状态**  
返回的 Web 页面显示状态

结合使用 Selenium 和 TestNG 验证简单的 Create Widget 用例是一种可管理的应用：

1. 配置并启动 Selenium 服务器的实例。
2. 与 Create Widget Web 表单交互并提交它。
3. 检验结果页面是否包含具有小部件名称的成功信息。
4. 停止 Selenium 服务器实例。

请注意：用例中的每一步都是通过 Selenium 完成的 —— 所以说，TestNG 仅仅帮助进行查找。现在，我们来实践一下。

Create Widget 测试用例

我希望对 Selenium 服务器进行灵活的配置，所以我将编写一个*参数化 fixture*（TestNG-Selenium 样式），一般可以使用它来为不同浏览器、不同位置甚至混合的 Web 应用程序地址（类似 localhost 和产品）创建 Selenium 服务器。清单 4 定义了我所配置的灵活的 Selenium 服务器 fixture：

**清单 4. 灵活的 Selenium fixture**

|  |
| --- |
| @Parameters()  @BeforeClass  private void init(String selenSrvrAddr, String bpath,  String appPath) throws Exception {  driver = new DefaultSelenium(selenSrvrAddr,  SeleniumServer.getDefaultPort(), bpath, appPath);  driver.start();  }  //....  @AfterClass  private void stop() throws Exception {  driver.stop();  } |

必须将参数名与 TestNG 的 testng.xml 文件中的值链接起来；因此，我定义了如清单 5 所示的三个参数。（默认情况下为 Firefox 定义了 brwsr-path 参数，但是我可以同样轻松地定义一组新的使用 Internet Explorer 的测试。）

**清单 5. TestNG testng.xml 文件中的参数值**

|  |
| --- |
| <parameter name="selen-svr-addr" value="localhost"/>  <parameter name="aut-addr" value="http://localhost:8080/gt15/"/>  <parameter name="brwsr-path" value="\*firefox"/> |

接下来，我将定义清单 6 所示的测试用例，它也包含一个参数，用于进行测试的应用程序的基 URL。该测试将促使浏览器在 Web 应用程序内打开特定页面，并操作 [图 1](http://www.ibm.com/developerworks/cn/java/j-cq04037/index.html#fig1) 所示的表单。

**清单 6. 一个良好的测试用例**

|  |
| --- |
| @Parameters()  @Test  public void verifyCreate(String appPath) throws Exception {  driver.open(appPath + "/CreateWidget.html");  driver.type("widget", "book-01");  driver.select("type", "book");  driver.type("definition", "book widget type book");  driver.click("submit");  driver.waitForPageToLoad("10000");  assertEquals(driver.getText("success"),  "The widget book-01 was successfully created.",  "test didn't return expected message");  } |

通过调用 driver.click("submit") 提交表单后，Selenium 将等待响应的加载，然后我将断言成功的创建信息。（注意：响应 Web 页面具有一个 ID 为 *success* 的元素。）

结果产生一个灵活的文本类，它将检验两种场景：一种是良好的场景，而另一种是没有提供定义的边界用例，如清单 7 所示：

**清单 7. 使用 TestNG 进行全部的处理**

|  |
| --- |
| public class CreateWidgetUATest {  private Selenium driver;  @Parameters()  @BeforeClass  private void init(String selenSrvrAddr, String bpath,  String appPath) throws Exception {  driver = new DefaultSelenium(selenSrvrAddr,  SeleniumServer.getDefaultPort(), bpath, appPath);  driver.start();  }  @Parameters()  @Test  public void verifyCreate(String appPath) throws Exception {  driver.open(appPath + "/CreateWidget.html");  driver.type("widget", "book-01");  driver.select("type", "book");  driver.type("definition", "book widget type book");  driver.click("submit");  driver.waitForPageToLoad("10000");  assertEquals(driver.getText("success"),  "The widget book-01 was successfully created.",  "test didn't return expected message");  }  @Parameters()  @Test  public void verifyCreationError(String appPath) throws Exception {  driver.open(appPath + "/CreateWidget.html");  driver.type("widget", "book-02");  driver.select("type", "book");  //definition explicitly set to blank  driver.type("definition", "");  driver.click("submit");  driver.waitForPageToLoad("10000");  assertEquals(driver.getText("failure"),  "There was an error in creating the widget.",  "test didn't return expected message");  }  @AfterClass  private void stop() throws Exception {  driver.stop();  }  } |

目前为止，我已经定义了两种足够灵活的 Selenium 测试，可以对多个浏览器进行测试，并且还可以对多个位置进行测试，这对初学者非常有利。尽管如此，我还想获得更高级点的应用，我开始考虑测试中的逻辑是否可重复使用。比如，如果对一行运行两次 CreateWidgetUATest 测试类会怎样？如何确保我的 Web 应用程序运行的是本地机器（或其他机器）上最新版本的代码？

可重复的验收测试

在执行 Selenium 测试时，必须运行 Selenium 服务器以及要检验的 Web 应用程序。言外之意，还必须运行应用程序中所有相关的架构依赖关系 —— 对于大多数 Java™ Web 应用程序来说，即 Servlet 容器和相关的数据库。

正如在我的另一篇文章 [repeatable system tests](http://www.ibm.com/developerworks/cn/java/j-cq04037/resources.html) 中解释的一样，DbUnit 和 Cargo 是两种我最喜欢的技术，可以在依赖数据库的 Web 应用程序中实现逻辑重复。DbUnit 管理数据库中的数据，而 Cargo 使容器管理以通用的方式实现自动化。下面几节将向您展示如何结合使用 Selenium 和 TestNG 从而确保实现逻辑重复的验收测试。

DbUnit 再次登场

您可能回想起，DbUnit 通过有效地管理测试场景中的数据简化了使用数据库的工作。通过使用 DbUnit，可以在测试前将一组已知的数据加载到数据库中，这意味着您可以依赖这些在测试过程中呈现的数据。此外，在完成测试后，还可以从数据库中删除测试结果产生的数据。DbUnit 作为一种方便的 fixture（JUnit 或 TestNG）简化了所有这些工作，它能够读取包含测试数据的种子文件，逻辑插入、删除数据，或更新数据到相应的数据库表中。

由于这里使用了 TestNG 驱动 Selenium，我将创建一个 DbUnit fixture，它将在*测试* 级别上运行。TestNG 支持在五种粒度级别上运行 fixture。最低的两种级别，方法和类是最常见的 —— 用于每个测试方法的 fixture 或者用于整个类的 fixture。之后，TestNG 为一个测试*集合*（定义在 TestNG 配置文件中并由 test 元素指定）定义了一个 fixture，为一*组* 测试（定义在 TestNG 的 Test 注释中）定义了一个 fixture。

**测试细节**

创建一个 DbUnit fixture 并在测试级别上运行，这意味着运行任何测试之前，测试类的集合将共享相同的逻辑，为数据库正确地播种。在本文的示例中，在运行每个逻辑测试集合前，我希望数据库具有一组干净的数据。使用 DbUnit 的 CLEAN\_INSERT 命令确保在先前运行的测试中创建的行被删除掉 —— 因此，我可以重新运行测试，该测试可以不断创建数据并且不用考虑数据库约束。

此外，我希望 fixture 能够依赖参数化数据，这使我在运行某个测试之前，能够灵活地切换种子文件，甚至是特定数据库的位置。将 TestNG 与参数相关联起来再简单不过了：我所需做的仅仅是使用 Parameters 注释装饰 fixtrue，声明方法签名中相应的参数，并提供 TestNG 配置文件中的值。

清单 8 定义了一个简单的 DbUnit fixture，它使用所需的种子文件播种数据库。请注意：该 fixture 被定义为包含*五个* 参数。（这可能非常多，但是在 fixture 中包含参数不是很好吗？）

**清单 8. 测试集合的 DbUnit fixture**

|  |
| --- |
| public class DatabaseFixture {  @Parameters()  @BeforeTest  public void seedDatabase(String seedpath, String driver,  String url, String user, String pssword) throws Exception {  IDatabaseConnection conn = this.getConnection(driver, url, user, pssword);  IDataSet data = this.getDataSet(seedpath);  try {  DatabaseOperation.CLEAN\_INSERT.execute(conn, data);  }finally {  conn.close();  }  }  private IDataSet getDataSet(String path) throws IOException, DataSetException {  return new FlatXmlDataSet(new File(path));  }  private IDatabaseConnection getConnection(String driver,  String url, String user, String pssword ) throws ClassNotFoundException,  SQLException {  Class.forName(driver);  Connection jdbcConnection =  DriverManager.getConnection(url, user, pssword);  return new DatabaseConnection(jdbcConnection);  }  } |

要将实际的值与清单 8 中的参数相关联，我必须在 TestNG 的 testng.xml 文件中定义它们，如清单 9 所示：

**清单 9. TestNG 的 testng.xml 文件中定义的特定于 DbUnit 的参数**

|  |
| --- |
| <parameter name="seed-path" value="test/conf/gt15-seed.xml"/>  <parameter name="db-driver" value="org.hsqldb.jdbcDriver"/>  <parameter name="db-url" value="jdbc:hsqldb:hsql://127.0.0.1"/>  <parameter name="db-user" value="sa"/>  <parameter name="db-psswrd" value=""/> |

**通用参数值**

现在我已经定义了一个灵活的 fixture，它将处理数据库状态和相应测试。现在可以准备使用 TestNG 将所有内容连接起来。通常，第一步是了解希望实现的内容。在本例中，我想完成以下任务：

* 我希望在运行任何逻辑测试集合前，DbUnit fixture 能够完成自己任务。
* 我希望将相同的测试集合运行两次：一次用于 Firefox，一次用于 Internet Explorer。

TestNG 的 parameter 元素的作用域是局部的，这对我来说是件好事。这样，我可以很容易地在 TestNG 配置文件中定义通用参数值，并且当需要时在 TestNG 的 test 组元素中重写它们。

比如，要运行两组测试，简单创建两个 test 元素。我可以通过 TestNG 的 package 元素将我的 fixture 和相关测试包括进来，package 元素能够使包结构中所有测试（或 fixture）的查找变得简单。接着，我可以在两个定义了的 test 组中将 Firefox 和 Internet Explorer 的 brwsr-path 参数关联起来。所有这些都显示在了 testng.xml 文件中，如清单 10 所示：

**清单 10. 使 DbUnit 运行的灵活的 testng.xml 文件**

|  |
| --- |
| <suite name="User Acceptance Tests" verbose="1" >    <!-- required for DbUnit fixture -->  <parameter name="seed-path" value="test/conf/gt15-seed.xml"/>  <parameter name="db-driver" value="org.hsqldb.jdbcDriver"/>  <parameter name="db-url" value="jdbc:hsqldb:hsql://127.0.0.1"/>  <parameter name="db-user" value="sa"/>  <parameter name="db-psswrd" value=""/>    <!-- required for Selenium fixture -->  <parameter name="selen-svr-addr" value="localhost"/>  <parameter name="aut-addr" value="http://localhost:8080/gt15/"/>  <test name="GT15 CRUDs- Firefox" >    <parameter name="brwsr-path" value="\*firefox"/>  <packages>  <package name="test.com.acme.gt15.Web.selenium" />  <package name="test.com.acme.gt15.Web.selenium.fixtures" />  </packages>  </test>  <test name="GT15 CRUDs- IE" >    <parameter name="brwsr-path" value="\*iexplore"/>  <packages>  <package name="test.com.acme.gt15.Web.selenium" />  <package name="test.com.acme.gt15.Web.selenium.fixtures" />  </packages>  </test>  </suite> |

我很高兴地宣布，我已经完成了创建一套可重复验收测试所需的所有事情。剩下的工具就是处理 Web 应用程序容器本身。幸运地是，我可以使用 Cargo 来完成。

Cargo 执行加载

Cargo 是一个创新的以通用方式自动化容器管理的开源项目，比如，用于将 WAR 文件部署到 JBoss 的相同 API 还可以启动和停止 Tomcat。Cargo 还可以自动下载并安装容器 —— Cargo API 的用途很广泛，从 Java 代码到 Ant 任务，甚至是 Maven。

诸如 Cargo 这样的工具将处理编写逻辑重复测试用例所面对的一个大的挑战，它避免一种潜在的假设，即*运行* 的容器具有最新最好的应用程序代码。此外，还可以构造一个利用 Cargo 的能力自动完成以下任务的编译过程（例如在 Ant 内）：

1. 下载所需的容器。
2. 安装该容器。
3. 启动容器。
4. 将选择的 WAR 或 EAR 文件部署到容器上。

稍后，您还可以使 Cargo 停止所选的容器。（并且，不需要对下载和安装容器发出警告，或者，如果本地机器中已经存在了正确的版本，Cargo 将跳过步骤 1 和 2。）

我希望使用 Cargo 来确保启动并运行最新和最好的 Web 应用程序。并且，我不需要考虑在哪里部署 WAR 文件，或者必须确保正在使用的是最新的 WAR 文件。我真正想达到的目的是使用户验收测试实现*无事件* —— 我仅需要发出*一个* 命令，然后坐下来等待结果。甚至可以更好，在一个 CI 环境中，我不用等待；当测试完成后我将获得一个通知！

**测试容器管理**

要在 Ant 内设置 Cargo，我需要定义一个任务，它将下载特定版本的 Tomcat 并将其安装到本地机器上的临时目录。接下来，将最新版本的代码部署到 Tomcat 上，如清单 11 所示：

**清单 11. 设置 Cargo 的任务**

|  |
| --- |
| <target name="ua-test" depends="compile-tests,war">  <taskdef resource="cargo.tasks">  <classpath>  <pathelement location="$/$" />  <pathelement location="$/$" />  </classpath>  </taskdef>    <cargo containerId="tomcat5x" action="start" wait="false" id="$">  <zipurlinstaller installurl="$" />  <configuration type="standalone" home="$">  <property name="cargo.remote.username" value="admin" />  <property name="cargo.remote.password" value="" />  <deployable type="war" file="$/$" />  </configuration>  </cargo>  <antcall target="\_start-selenium" />  <cargo containerId="tomcat5x" action="stop" refid="$" />  </target> |

清单 11 中的 target 使用 antcall 调用另一个 target。实际上，清单 11 中最后的 cargo 任务封装了 \_start-selenium target，并且确保运行测试后停止 Tomcat。

在清单 12 中定义的 \_start-selenium target 中，我需要启动（并稍后停止）Selenium 服务器。在此过程中，我的测试还将连接到其 Selenium fixture 中的服务器实例。请注意：该 target 是如何引用另一个 target ——

**清单 12. 启动和停止 Selenium 服务器**

|  |
| --- |
| <target name="\_start-selenium">  <java jar="$/$" fork="true" spawn="true" />  <antcall target="\_run-ua-tests" />  <get dest="$/results.txt"  src="$/selenium-server/driver/?cmd=shutDown" />  </target> |

最后，该组中最后的 target 将通过 TestNG 实际运行我的编程式 Selenium 测试。注意，我是如何通过使用清单 13 中的 \_run-ua-tests target 的 xmlfileset 元素，使 TestNG 使用我的 testng.xml 文件。

**清单 13. 运行 TestNG testng.xml 文件中的测试**

|  |
| --- |
| <target name="\_run-ua-tests">  <taskdef classpathref="build.classpath" resource="testngtasks" />  <testng outputDir="$"  classpath="$;$" haltonfailure="true">  <xmlfileset dir="./test/conf" includes="testng.xml" />  <classpath>  <path refid="build.classpath" />  </classpath>  </testng>  </target> |

结束语

正如您看到的一样，Selenium 极大地简化了用户验收测试，尤其当使用 TestNG 驱动的时候。虽然编程式测试并不适用于所有人（非开发人员可能更喜欢 Selenium 的 Fit 样式的表），它确实让您了解到了 TestNG 非凡的灵活性。编程式测试还允许您使用 DbUnit 和 Cargo 构建自己的测试框架，从而确保测试的逻辑可重复性。

开源 Web 测试框架的发展绝不会停止，这对于追求代码质量的完美主义者是个好消息。Selenium 是驱动浏览器的开源 Web 测试框架中新出现的工具之一，它能够使用户验收测试自动化 —— 因此，它非常优秀。结合使用 Selenium 和 TestNG，正如我在本文中演示的一样，您将获得一个非常好的测试驱动，并从依赖性测试以及参数测试中获得巨大的优势。尝试使用 Selenium 和 TestNG 吧，您的用户将为此感谢您。