第8章　Web页面测试

随着 Internet 和 Intranet 的快速发展，Web 技术已经对工商业、医疗业、教育、政府、娱乐以及我们的生活产生了深远的影响。Web 平台能支持几乎所有媒体类型的信息发布，容易为最终用户存取，更多传统的信息和数据系统正在逐渐迁移到互联网上：电子商务正迅速增长，范围广泛、复杂的云应用和云计算也正在 Web 环境中出现。基于 Web 的系统在变得越来越复杂和强大的同时，Web 应用软件的缺陷危机也越来越严重。早在 1998 年 Yogesh Deshpande 和 Steve Hansen 就提出了 Web 工程的概念。Web 工程提倡使用一个过程和系统的方法来开发高质量的基于 Web 的系统。在 Web 工程中，基于 Web 系统的测试、确认和验收是一项重要而富有挑战性的工作。

Web 环境具有浏览器平台不兼容、网络环境多样化、应用复杂化等诸多特性，所以，传统测试方法的某些方面不适用于网络测试。Web 的自动化测试方法包含几个方面，比如，测试脚本技术、人工测试过程自动化、验证自动化等等。在测试驱动开发模式中，测试已成为迭代开发过程中起推动作用的环节，但与此同时，大量的重复性的测试代码却造成了大量资源的浪费。

随着自动化测试技术的成熟和自动化测试工具的广泛应用，人们重新认识到了测试的价值：最优的质量成本，最高的质量保证。自动化测试的优势在软件领域很明显的：减少了测试时间，使测试程序统一化，便于管理，节约了质量保证的成本，提高了测试运行的效率，改善了软件产品的质量。

现在一般人都有使用浏览器浏览网页的经历，用户虽然不是专业人员但是对界面效果的印象是很重要的。如果开发人员注重这方面的测试，那么验证应用程序是否易于使用就非常重要了。很多人认为这是测试中最不重要的部分，但是恰恰相反，界面对不懂技术的客户来说都是相当关键，特别是在简洁、美观、易用等方面。

方法上可以根据设计文档，如果够专业的话可以由专业美工人员，来确定整体风格，特别是页面风格。然后根据这个设计好的页面，生成静态的 HTML、CSS 等甚至生成几套不同的方案来讨论，或者交给客户评审，最后形成统一风格的页面/框架。

页面测试的主要页面元素有：

- 页面元素的容错性列表(如输入框、时间列表或日历)。
- 页面元素清单(为实现功能，是否将所需要的元素全部都列出来了，如按钮、单选按钮、复选框、列表框、超链接、输入框等等)。
- 页面元素的容错性是否存在。

- 页面元素的容错性是否正确。
- 页面元素的基本功能是否实现(如文字特效、动画特效、按钮、超链接)。
- 页面元素的外形、摆放位置(如按钮、列表框、复选框、输入框、超链接等)。
- 页面元素是否显示正确(主要针对文字、图形、签章)。
- 元素是否显示(元素是否存在)。

页面测试主要包括以下几个方面的内容：

- 站点地图和导航条位置是否合理，是否可以导航等。
- 页面内容布局是否合理，文字是否准确、简洁，字体和字号是否符合多数读者习惯。
- 背景/色调是否合理、美观，是否符合多数用户审美要求。
- 页面在窗口中的显示是否正确、美观(在调整浏览器窗口大小时，屏幕刷新是否正确)，表单样式 大小、格式是否适宜。
- 是否对提交数据进行验证(如果在页面部分进行验证的话)等。
- 链接的形式、位置、是否易于理解等。

对 Web 应用的测试可以分为页内测试(IntraPageTest)和跨页测试(InterPageTest)两种。页内测试相当于单元测试，着重于测试单个页面的行为是否正确。根据模块化思想，在进行页面划分时，一般使每个页面具有单一、具体的功能，可以直接表达用户的一个目标。本章我们主要考虑 Web 页内测试的主要方法。

- 人工走查：①通过页面走查，浏览确定使用的页面是否符合需求。可以结合兼容性测试不同分辨率下的页面显示效果，如果有影响，则应该交给设计人员由他们提出解决方案。②可以结合数据定义文档查看表单项的内容、长度等信息。③对于动态生成的页面最好也能浏览查看。如 Servelet 部分可以结合编码规范，进行代码走查。是否支持中文，如果数据用 XML 封装，要做的工作可能会多一点。
- 使用 Web 页面测试工具：工具可以提供很多测试方法，可以用来模拟许多手工操作，如单击按钮、给文本框输入字符或数字、鼠标双击事件等，从而实现了测试的自动化。这对于需要输入大量信息的界面测试来说是十分重要的。

Web 页面测试的基本准则：符合页面/界面设计的标准和规范，满足灵活性、正确性、直观性、舒适性、实用性、一致性等要求。

直观性：①用户界面是否洁净、不唐突、不拥挤，界面不应该为用户制造障碍，所需功能或者期待的响应应该明显，并在预期的地方出现。②界面组织和布局合理吗?是否允许用户轻松地从一个功能转到另一个功能?下一步做什么明显吗?任何时刻都可以决定放弃或者退回、退出吗?输入得到承认了吗?菜单或者窗口是否深藏不露? ③有多余功能吗?软件整体抑或局部是否做得太多?是否因有太多特性而把工作复杂化了?是否感到信息太庞杂? ④如果其他所有努力失败，帮助系统真能帮忙吗?

一致性：①快捷键和菜单选项，在 Windows 中按 F1 键总是得到帮助信息。②术语和命令，整个软件使用同样的术语吗?特性命名一致吗?例如，Find 是否一直叫 Find，而不是有时叫 Search? ③软件是否一直面向同一级别用户?带有花哨用户界面的趣味贺卡程序不应该显示泄

露技术机密的错误提示信息。④按钮位置和等价的按键。大家是否注意到对话框有 OK 按钮和 Cancel 按钮时，OK 按钮总是在上方或者左方，而 Cancel 按钮总是在下方或右方?同样原因，Cancel 按钮的等价按键通常是 Esc，而 OK 按钮的等价按钮通常是 Enter，要保持一致。

灵活性：①状态跳转，灵活的软件实现同一任务时通常会有多种选择方式。②状态终止和跳过，具有容错处理能力。③数据输入和输出，用户希望有多种方法输入数据和查看结果。例如，要在写字板中插入文字，可用键盘输入、粘贴、从 6 种文件格式读入、作为对象插入，或者用鼠标从其他程序拖动。

舒适性：①恰当，软件外观和感觉应该与所做的工作和使用者相符。②错误处理，程序应该在用户执行严重错误的操作之前提出警告，并允许用户恢复由于错误操作导致丢失的数据。正如大家认为 undo /redo 功能是理所当然应有的。③性能，快不见得是好事，要让用户看清程序在做什么，它是有反应的。

8.1　Web 页面测试工具介绍

基于 Web 的测试基本上采用两种思路和方法。一种可以称为“浏览器测试”，这种测试通常是模拟浏览器端的一些操作，比如在 TextBox 中输入一些文本，选择表单组件中的某个部件等。因为可以看见具体的操作界面，这种方法更多地被应用到 UI 和本地化方面的测试中。

另一种方法称为“协议测试”。这是建立在 HTTP 协议之上的 JavaScript 级别的测试，通过 JavaScript 伪协议或者 POST、Web Service 向服务器发送请求，然后对服务器响应回来的数据进行解析、验证。一些功能测试会更多地采用这种方法。最简单的应用就是检查链接的有效性：向服务器发送 URL 请求，检查响应回来的数据，来判断链接是否指向了正确的页面。

本章所介绍的 Web 页面测试工具事实上能够很好地支持上述测试方法，并且它们均是以 JUnit 为基础扩展出来的，但应用模式各有不同。

1. HttpUnit

HttpUnit 是在 JUnit 之上构建的测试框架，它支持 Web 应用的黑盒测试和 in-container 容器内测试。作为功能测试工具，可以用它来验证软件是否符合业务需求，以及是否在可视的级别符合预期行为。有趣的是，HttpUnit 的基础代码实际上跟测试没什么关系。开发 HttpUnit 库的目的是加强 HTTP 对 Web 应用的访问，它支持的特征包括状态管理(cookies)、请求提交、应答解析(HTML 解析)，以及网络爬虫(Web Spider)工具包需要的一些特征。HttpUnit 还有一个支持容器内测试的类 ServletUnit。在 JUnit 提供的断言功能和结果报告功能的基础上，HttpUnit 成了一个非常有用的测试 Web 应用的工具。可以在 www.httpunit.org 上找到 HttpUnit。

2. JWebUnit

JWebUnit 是在 HttpUnit 上创建的一个辅助工具包，它减少了测试 Web 程序所需要编

写的代码。简单地说，可以把它当作 HttpUnit 的宏程序库，提供到 HttpUnit 代码段的快捷方式，简化 Web 程序测试中的大多数行为。HttpUnit 提供的相对底层的接口可以让测试人员定制许多事情。如果用 HttpUnit 可以解决问题，那么用 JWebUnit 也可以。JWebUnit 的好处是它对代码有更好的控制。可以在 http://jwebunit.sourceforge.net/上找到 JWebUnit。

3. StrutsTestCase

StrutsTestCase 是为测试 Struts 应用而在 JUnit 基础上创建的测试框架。Struts 是用 Java 开发 Web 应用的程序员非常喜欢的模型-视图-控制器(MVC)平台，它简化了数据、表示和逻辑分离的易维护性组件式代码开发。Struts 使 Web 程序容器间(in-container)的功能测试和单元测试变得更复杂了，因为它们夹在 servlet 容器和程序之间。这就意味着这个测试框架要认识 Struts，能处理 Struts 的容器间测试。由于不需要知道 Web 程序的内部实现，所以尽管 HttpUnit 的黑盒测试仍然工作得很好，也仍然无法用 HttpUnit 做 Struts 应用的容器间测试，因为 HttpUnit 是独立地位于程序和 servlet 容器之间。StrutsTestCase 是专为 Struts 程序的容器间测试而设计的。StrutsTestCase 可以从 http://strutstestcase.sourceforge.net/获得。

4. Selenium

Selenium 是一个用于 Web 应用程序测试的工具，它将核心组件内插到浏览器中。Selenium 测试直接运行在浏览器中，就像真正的用户在操作一样，所有的测试都是可见的。支持的浏览器包括 IE、Mozilla、Firefox 等。这个工具的主要功能包括：测试与浏览器的兼容性，即测试应用程序是否能够很好地工作在不同浏览器和操作系统之上，测试系统功能，检验软件功能和用户需求。Selenium 可从 http://seleniumhq.org/download/上下载。

5. HtmlUnit

HtmlUnit 是一个具有浏览器基本功能的 Java 组件。HtmlUnit 是 JUnit 的扩展测试框架之一，它支持 HTML 文件，并提供了一些 API，允许访问网页、填写表格、点击链接等。HtmlUnit 将返回文档模拟成 HTML，这样我们便可以直接处理这些文档了。HtmlUnit 使用诸如 table、form 等标识符将测试文档作为 HTML 来处理。它同样需要遵循 JUnit 测试框架结构的 Java 测试程序。这里要注意的是：前面介绍的 HttpUnit 主要是在 http 的 request 和 response 层次上进行操作，而 HtmlUnit 则是在比 http 高一些的 HTML 层次上进行操作。HtmlUnit 可从 http://htmlunit.sourceforge.net/上下载。

8.2　Web 页面测试工具之一——HttpUnit

HttpUnit 是 SourceForge 下面的一个开源项目，它是基于 JUnit 的一个测试框架，主要关注于测试 Web 应用，解决使用 JUnit 框架无法对远程 Web 内容进行测试的弊端。当前的最新版本是 1.7。为了使用 HtpUnit 能正常运行，您应该安装 JavaJDK1.3.1 或以上版本。

1. HttpUnit 的工作原理

HttpUnit 不需要使用浏览器。我们可以使用 HttpUnit 直接调用要测试的代码。从本质上来说，HttpUnit 是模拟 Web 浏览器，并且 HttpUnit API 可以模拟浏览器的许多行为，包括表单提交、JavaScript、HTTP 认证和 Cookie 等。我们也可以在装入 Web 页面时用 HttpUnit API 分析返回的内容。

HttpUnit 通过模拟浏览器的行为，处理页面帧(frames)、cookies、页面重定向(redirects)等。通过 HttpUnit 提供的功能，我们可以和服务器端进行信息交互，将返回的网页内容作为普通文本、XML DOM 对象或者是作为链接、页面框架、图像、表单、表格等的集合进行处理，然后使用 JUnit 框架进行测试；还可以导向一个新的页面，然后进行新页面的处理，这个功能使我们可以处理一组在一个操作链中的页面，轻松地测试 Web 页面。

HttpUnit 可以被分为两个核心组件：

- 一个发送请求并接收响应的 Web 客户机
- 一个分析并验证响应内容的方法集

在这里，我们要澄清一个概念——使用 HttpUnit 是不是相当于进行单元测试？

与其名称可能暗示的相反，HttpUnit 实际上并不做单元测试。实际上，HttpUnit 更适合做功能测试或“黑盒”测试。测试人员用 HttpUnit 编写的测试从外部查询 Web 服务器并使我们能够分析接收到的响应。功能测试在 XP 方法中起着重要的作用，这种方法强调功能测试，使开发者可以获取有关系统整体状态的反馈。使用单元测试，有时会丢失大方向。在将整个站点投入到实际使用的过程中，自动功能测试可以使开发者从手工检查站点区域的繁重工作中解脱出来。在 Web 环境中，有些专家认为每个“请求-响应”周期都是原子的，这些原子的周期可以依次被作为单独的代码单元，因此使用 HttpUnit 可以被认为是在进行这种意义上的单元测试。

2. HttpUnit 和其他商业工具的对比

商业工具一般使用录制、回放功能来实现测试，但是这里有个缺陷，就是当页面设计被修改以后，这些被记录的行为就不能重用了，需要重新录制才能继续测试。

举个例子：如果页面上有个元素最先的设计是采用单选框，这个时候你开始测试，那么这些工具记录的就是你的单项选择动作；但是如果你的设计发生了变化，比如说我改成了下拉选择，或者使用文本框来接收用户输入，那么这时候，你以前录制的测试过程就无效了，必须重新录制。

而 HttpUnit 因为关注点是这些控件的内容，所以不管控件的外在表现形式如何变化，都不会影响已确定测试的可重用性。

最重要的是，由于 Web 应用的快速发展以及 Web 编程错误容易产生的特征，市场上出现的许多商业测试产品都用详细的 GUI 来引导开发者进行测试。另一方面，开放源代码的 HttpUnit 无需许可费用，而且使用简单(我们将在下面看到这一点)。正是它的简单性，使它如此受欢迎。虽然没有漂亮的 GUI，但 HttpUnit Java 源代码的可用性和它的简单 API，使开发者能够创建他们自己的测试解决方案以满足特定组织的需要。以 HttpUnit 作为基础，

我们可以用最小的成本构建复杂的测试套件。

8.2.1　HttpUnit 环境建立

可以从HttpUnit项目网站——http://httpunit.sourceforge.net2上下载到最新版的HttpUnit，HttpUnit 应运行在支持 Java JDK 1.3 及更高版本的系统上。

将 HttpUnit 解压缩到 c:/httpunit(后面将使用“%httpunit_home%”引用该目录)，目录结构应该如下所示：

```
httpunit
   +--- jars //包含创建、测试及运行 HttpUnit 所必需的 jar
   |
   +--- lib   //  包含 HttpUnit jar
   |
   +--- doc   //文档
   |        |
   |        +--- tutorial   //基于 servlet Web 网站的测试优先开发的简单教程
   |        |
   |        +--- api          // javadoc
   |        |
   |        +--- manual      //  用户手册
   |
   +--- examples //  采用 HttpUnit 编写的一些示例程序
   |
   +--- src          // HttpUnit  源代码
   |
   +--- test        // HttpUnit 单元测试的一些很好的例子
```

只有 lib 和 jars 这两个目录对于运行 HttpUnit 来说是必需的。我们必须将 HttpUnit jar 添加到系统的 classpath，而其他的一些 jar 均为可选。

由于我们考虑 Web 页面是在 Eclipse 中开发、执行的，所以环境配置都是以 Eclipse 为例。如果使用其他的开发工具，则可根据以下步骤进行环境配置。

(1) 启动 Eclipse，建立一个 Java 项目。

(2) 将%httpunit_home%/lib/*.jar、%httpunit_home%/jars/*.jar 加入到该 Java 项目的 Java build Path 变量中。

8.2.2　HttpUnit 的工作方式

1. 在使用 HttpUnit 进行页面测试时，需要特别关注如下几个类

- WebConversation 是 HttpUnit 框架中最重要的类，它用于模拟浏览器的行为。
- WebRequest 模仿客户请求，通过它可以向服务器发送信息。

- WebResponse 模拟浏览器，获取服务器端的响应信息。

下面通过示例介绍上述类的具体应用：

1) 类的最基本应用

```
System.out.println("直接获取网页内容：");
// 建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
// 向指定的 URL 发出请求，获取响应
WebResponse wr = wc.getResponse( " http://localhost:6888/HelloWorld.html " );
// 用 getText 方法获取响应的全部内容
// 用 System.out.println 将获取的内容打印在控制台上
System.out.println( wr.getText() );
```

2) 通过 Get 方法访问页面并加入参数

```
System.out.println("向服务器发送数据，然后获取网页内容：");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//向指定的 URL 发出请求
WebRequest req = new GetMethodWebRequest(" http://localhost:6888/HelloWorld.jsp " );
//给请求加上参数
req.setParameter("username","姓名");
//获取响应对象
WebResponse resp = wc.getResponse(req);
//用 getText 方法获取响应的全部内容
//用 System.out.println 将获取的内容打印在控制台上
System.out.println(resp.getText());
```

3) 通过 Post 方法访问页面并加入参数

```
System.out.println("使用 Post 方式向服务器发送数据，然后获取网页内容：");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//向指定的 URL 发出请求
WebRequest req = new PostMethodWebRequest(" http://localhost:6888/HelloWorld.jsp ");
//给请求加上参数
req.setParameter("username","姓名");
//获取响应对象
WebResponse resp = wc.getResponse(req);
//用 getText 方法获取响应的全部内容
//用 System.out.println 将获取的内容打印在控制台上
```

```
System.out.println(resp.getText());
```

大家可以关注上面代码中打了下划线的两处内容，应该可以看到，使用 Get、Post 方法访问页面的区别就是使用的请求对象不同。

2. 处理页面中的链接

找到页面中的某一个链接，然后模拟用户的点击行为，获得它指向文件的内容。比如在页面 HelloWorld.html 中有一个链接，它显示的内容是 TestLink，它指向另一个页面 TestLink.html。TestLink.html 页面只显示 TestLink.html 页面的几个字符。

下面是处理代码：

```
System.out.println("获取页面中链接指向页面的内容: ");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//获取响应对象
WebResponse resp = wc.getResponse( " http://localhost:6888/HelloWorld.html " );
//获得页面链接对象
WebLink link = resp.getLinkWith( "TestLink" );
//模拟用户单击事件
link.click();
//获得当前的响应对象
WebResponse nextLink = wc.getCurrentPage();
//用 getText 方法获取响应的全部内容
//用 System.out.println 将获取的内容打印在控制台上
System.out.println( nextLink.getText() );
```

3. 处理页面中的表格

表格是用来控制页面显示的常规对象，HttpUnit 使用数组来处理页面中的多个表格，我们可以用 resp.getTables()方法获取页面中所有的表格对象。它们依照出现在页面中的顺序保存在一个数组里面。

注意：Java 中数组的下标是从 0 开始的，所以取第一个表格时应该是 resp.getTables()[0]，其他依此类推。

下面的示例演示了如何从页面中取出第一个表格的内容并将它们循环显示出来：

```
System.out.println("获取页面中表格的内容: ");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//获取响应对象
WebResponse resp = wc.getResponse( " http://localhost:6888/HelloWorld.html " );
//获得对应的表格对象
WebTable webTable = resp.getTables()[0];
```

```
//将表格对象的内容传递给字符串数组
String[][] datas = webTable.asText();
//循环显示表格内容
int i = 0 ,j = 0;
int m = datas[0].length;
int n = datas.length;
while (i<n){
    j=0;
    while(j<m){
        System.out.println("表格中第"+(i+1)+"行第"+(j+1)+"列的内容是：  "+datas[j]);
        ++j;
    }
    ++i;
}
```

4. 处理页面中的表单

表单用来接收用户输入，也可以向用户显示用户已输入的信息(如需要用户修改数据时，通常会显示他之前输入过的信息)。HttpUnit 使用数组来处理页面中的多个表单，我们可以用 resp.getForms()方法获取页面中所有的表单对象。它们依照出现在页面中的顺序保存在一个数组里面。

注意：Java 中数组的下标是从 0 开始的，所以取第一个表单时应该是 resp.getForms()[0]，其他依此类推。

下面的示例演示了如何从页面中取出第一个表单的内容并将它们循环显示出来：

```
System.out.println("获取页面中表单的内容：  ");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//获取响应对象
WebResponse resp = wc.getResponse( " http://localhost:6888/HelloWorld.html " );
//获得对应的表单对象
WebForm webForm = resp.getForms()[0];
//获得表单中所有控件的名称
String[] pNames = webForm.getParameterNames();
int i = 0;
int m = pNames.length;
//循环显示表单中所有控件的内容
while(i<m){
    System.out.println("第"+(i+1)+"个控件的名称是"+pNames+"，\
                        里面的内容是"+webForm.getParameterValue(pNames));
     ++i;
```

```
}
```

5. 如何使用 HttpUnit 进行测试

1) 对页面内容进行测试

HttpUnit 本身没有测试功能，事实上，它就是由一些类库组成，可以模拟出一个浏览器(WebConversation 类)，并可以模拟用户在网页上的多种行为。HttpUnit 要实现 Web 测试功能，需要结合 JUnit 才行。所以，HttpUnit 中的这部分测试完全采用了 JUnit 的测试方法，即直接将我们期望的结果和页面中的输出内容进行比较。这里只是字符串和字符串的比较。比如，我们期望的页面显示是其中有一个表格，并且是页面中的第一个表格，而且它的第一行第一列显示的数据应该是 username；那么，我们可以使用下面的代码进行自动化测试：

```
System.out.println("获取页面中表格的内容并且进行测试:  ");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//获取响应对象
WebResponse resp = wc.getResponse( " http://localhost:6888/TableTest.html " );
//获得对应的表格对象
WebTable webTable = resp.getTables()[0];
//将表格对象的内容传递给字符串数组
String[][] datas = webTable.asText();
//对表格内容进行测试
String expect = "中文";
Assert.assertEquals(expect,datas[0][0]);
```

2) 对 Servlet 进行测试

除了对页面内容进行测试外，有时候(比如开发复杂的 Servlet 的时候)，我们还需要对 Servlet 本身的代码块进行测试。可以选择 HttpUnit，它可以提供一个模拟的 Servlet 容器，让 Servlet 代码不需要发布到 Servlet 容器(如 tomcat)就可以直接测试。

使用 HttpUnit 测试 Servlet 时，需要创建一个 ServletRunner 实例，它负责模拟 Servlet 容器环境。如果只是测试一个 Servlet，那么可以直接使用 registerServlet 方法注册这个 Servlet；如果需要配置多个 Servlet，那么可以编写自己的 web.xml，然后在初始化 ServletRunner 的时候将它的位置作为参数传给 ServletRunner 的构造器。

在测试 Servlet 时，应该记得使用 ServletUnitClient 类作为客户端，它和前面用过的 WebConversation 差不多，都继承自 WebClient，所以它们的调用方式基本一致。要注意的是，在使用 ServletUnitClient 时，它会忽略 URL 中的主机地址信息，而是直接指向 ServletRunner 所实现的模拟环境。

下面的示例只是演示了如何简单地访问 Servlet 并获取它的输出信息。例子中 Servlet 在接到用户请求的时候只是返回一串简单的字符串“Hello World!”。

Servlet 的代码如下：

```
public class MyServlet extends HttpServlet {
public void service(HttpServletRequest req, HttpServletResponse resp)
throws IOException
{
        PrintWriter out = resp.getWriter();
        //向浏览器中写一个字符串 Hello World!
        out.println("<html><body>Hello World!</body></html>");
        out.close();
    }
}
```

测试的调用代码如下：

```
//创建 Servlet 的运行环境
ServletRunner sr = new ServletRunner();
//向环境中注册 Servlet
sr.registerServlet( "myServlet", MyServlet.class.getName() );
//创建访问 Servlet 的客户端
ServletUnitClient sc = sr.newClient();
//发送请求
WebRequest request = new GetMethodWebRequest( " http://localhost/myServlet " );
//获得模拟服务器的信息
WebResponse response = sc.getResponse( request );
//将获得的结果打印在控制台上
System.out.println(response.getText());
```

测试 Servlet 的内部行为：

对于开发者来说，仅仅测试请求和返回信息是不够的，所以 HttpUnit 提供的 ServletRunner 模拟器可以对被调用 Servlet 的内部行为进行测试。和简单测试中不同，这里使用了 InvocationContext 来获得该 Servlet 的环境，然后可以通过 InvocationContext 对象针对 request、response 等对象或者该 Servlet 的内部行为(非服务方法)进行操作。

下面的代码演示了如何使用 HttpUnit 模拟 Servlet 容器，并通过 InvocationContext 对象测试 Servlet 内部行为的大部分工作，比如控制 request、session、response 等。

```
//创建 Servlet 的运行环境
ServletRunner sr = new ServletRunner();
//向环境中注册 Servlet
sr.registerServlet( "InternalServlet", InternalServlet.class.getName() );
//创建访问 Servlet 的客户端
ServletUnitClient sc = sr.newClient();
```

```
//发送请求
WebRequest request = new GetMethodWebRequest( " http://localhost/InternalServlet " );
request.setParameter("pwd","pwd");
//获得该请求的上下文环境
InvocationContext ic = sc.newInvocation( request );
//调用 Servlet 的非服务方法
InternalServlet is = (InternalServlet)ic.getServlet();
is.myMethod();
//直接通过上下文获得 request 对象
System.out.println("request 中获取的内容: "+ic.getRequest().getParameter("pwd"));
//直接通过上下文获得 response 对象,并向客户端输出信息
ic.getResponse().getWriter().write("haha");
//直接通过上下文获得 session 对象，控制 session 对象
//给 session 赋值
ic.getRequest().getSession().setAttribute("username","timeson");
//获取 session 的值
System.out.println("session 中的值: "+ic.getRequest().getSession().getAttribute("username"));
//使用客户端获取返回信息，并且打印出来
WebResponse response = ic.getServletResponse();
System.out.println(response.getText());
```

注意在测试 Servlet 之前，必须通过 InvocationContext 完成 Servlet 中的 service 方法中完成的工作；因为通过 newInvocation 方法获取 InvocationContext 实例的时候，该方法并没有被调用。

8.3　Web 页面测试工具之二——JWebUnit

JWebUnit 是基于 Java 的用于测试网络程序的框架，架构在 HttpUnit 之上——即 JWebUnit 以HttpUnit和 JUnit单元测试框架为基础，适合做 Web 应用的验收测试。实际上也可以这么说，JWebUnit 是 HttpUnit 的更高一层 API 封装，提供了访问 Web 应用程序的高级 API，并组合了一组断言，用它们来验证链接导航、表单输入项和提交、表格内容以及其他典型业务类 Web 应用程序特性的正确性，避免了使用 HttpUnit 来编写繁琐复杂的测试用例。JWebUnit 是以 JAR 文件形式提供的，可以很容易地将它作为插件集成到大多数的 IDE 中，JWebUnit 还包含了其他必要的库。

除了底层的一些逻辑 API 外，JWebUnit 还轻量地集成了现有的测试框架 HtmlUnit 和 Selenium，并提供统一的一组简单易用的接口，可以很方便地与 HtmlUnit 或 Selenium 测试项目集成。所以对于目标测试用例来说，所有的接口都是透明且统一语义的，这将极大保证测试用例的通用性并降低了开发难度。

8.3.1　JWebUnit 测试环境建立

JWebUnit 可以通过点击链接 http://sourceforge.net/projects/jwebunit 来下载。JWebUnit 编译需要下面这几个包的支持：HttpUnit Java 库包，HttpUnit 用到的 Rhino JavaScript(仅在对 JavaScript 进行测试时需要)，HttpUnit 用到的 nekohtml 分析器和美化器，JUnit。另外，如用到 Tidy/HttpUnit 或 HttpUnit/neko 等功能，则还需下载 Apache XML API Common XML stuff 和 Xerces xml parser 等。

下载完 JWebUnit 之后，请按以下步骤在 Eclipse 平台上配置 JWebUnit 库：

(1) 把下载的 JWebUnit 文件释放到临时目录中(假设是 C:\temp)。JWebUnit 当前的最新版本为 2.2。

(2) 在 Eclipse(这里用到的测试环境为 MyEclipse 6.0.1)中创建新 Java 项目，将其命名为 JWebUnitTest。

(3) 右击 Package Explorer 视图中的 JWebUnitTest 项目，然后选择“Properties”。

(4) 单击“Java Build Path”。单击“Libraries”标签中的“Add External JARs”。

(5) 浏览到 C:\temp\jwebunit-2.2\lib 目录，选择这个目录中的所有 JAR 文件。

(6) 单击“OK“。

现在可以在 Eclipse 中的 JWebUnitTest 项目下开发 jWebUnit 测试用例了，如图 8-1 所示。

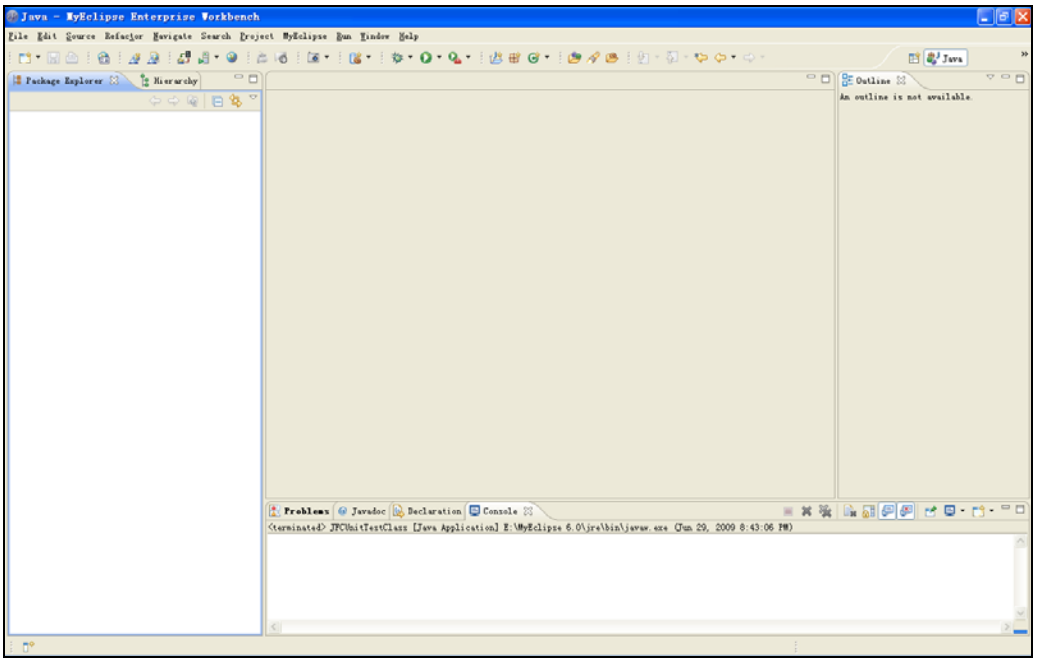

图 8-1　建立 JWebUnit 测试环境

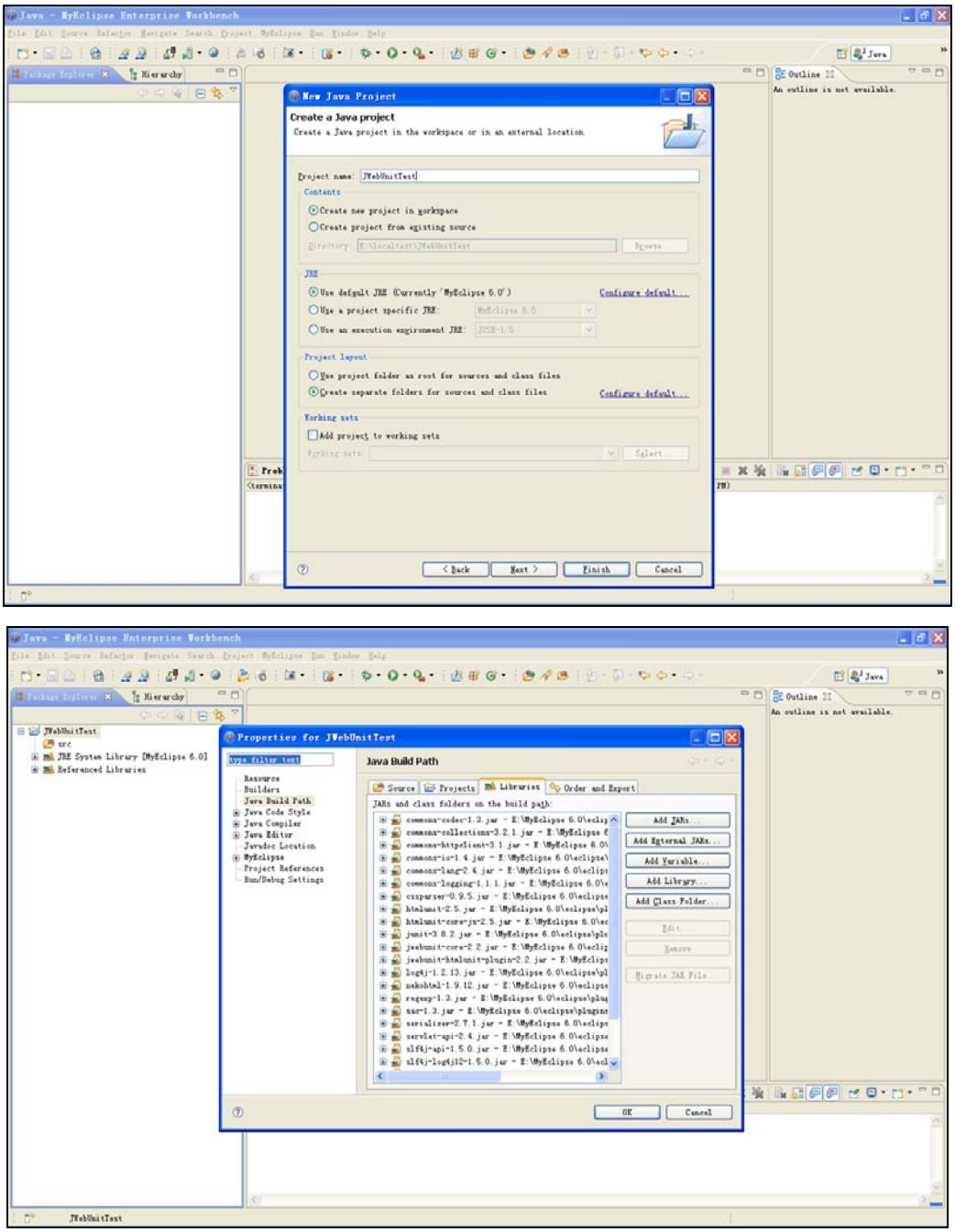

图 8-1　建立 JWebUnit 测试环境(续表)

8.3.2　JWebUnit 应用方法

1. 快速应用

JWebUnit 大致有两种方式来建立测试用例：继承模式和委托模式。下面这种是使用继

承模式继承 JWebUnit 提供的 WebTestCase 类，该类继承于 junit.framework.TestCase 类，代码示例如下：

```
import net.sourceforge.jwebunit.WebTestCase;
public class ExampleWebTestCase extends WebTestCase {
  public ExampleWebTestCase(String name) {
    super(name);
  }
}
```

委托模式的实现比上面的实现方式麻烦一点，代码示例如下：

```
import junit.framework.TestCase;
import net.sourceforge.jwebunit.WebTester;
public class ExampleWebTestCase extends TestCase {
  private WebTester tester;
  public ExampleWebTestCase(String name) {
    super(name);
    tester = new WebTester();
  }
}
```

用户测试用例代码继承了 junit.framework.TestCase 类，并在该类的全局变量里声明了一个 WebTester 类，该类封装了一些基本的 Web 操作动作，它属于 JWebUnit。在 ExampleWebTestCase 构造函数中将 WebTester 实例化。委托模式有个好处，代码表现上很简洁，测试动作和被测试业务代码将被表现得异常清晰，有利于后期的测试开发迭代。

2. HttpUnit 和 JWebUnit 测试方法对比

对 Web 应用程序自动进行测试意味着跳过 Web 浏览器，通过程序来处理 Web 站点。首先，我们看看 HttpUnit(JWebUnit 的构建块之一)是如何简化这项工作的。HttpUnit 可以模拟帧、JavaScript、页面重定向 cookie 等等。在将 HttpUnit 用于 JUnit 时，它可以迅速地对 Web 站点的功能进行验证。

下面代码显示了一个用 HttpUnit 编写的测试用例，它试图单击 HttpUnit 主页上的“Cookbook”链接：

```
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   }
```

这段代码的第 6 行用 your.proxy.com 连接 Internet。如果存在直接 Internet 连接，那么可以把这条语句注释掉。第 8 行的语句用于在页面中搜索包含文本 Cookbook 的 Web 链接。第 9 行的语句用于单击这个链接。如果找到链接，用户将会看到 Test Successful!这条消息。

下面代码是用 JWebUnit 改写后的上述 HttpUnit 的测试代码，看上去更简单了。

```
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 }
```

如果没注意特定于 JUnit 的代码，那么我们可以看到，测试用例现在变得相当整洁、简练。需要查看的重要的行是第 6 行、第 11 行和第 12 行。

在第 6 行，基本 URL 被设置到 HttpUnit 的主页中。第 11 行用相对路径/index.html 连接站点。第 12 行用于单击页面上具有文本 Cookbook 的链接。如果链接有效，那么 JUnit 会报告成功；否则，JUnit 会报告异常。

3. JWebUnit API

每个 JWebUnit 测试的核心都是 net.sourceforge.jwebunit.WebTestCase 类，它代表测试用例。每个测试用例都必须是从这个类扩展而来的(net.sourceforge. jwebunit.WebTestCase 类本身则是从 junit.framework.TestCase 类扩展而来的，它在 JUnit 中代表测试用例)。WebTestCase 类的重要方法参见表 8-1。

表 8-1　net.sourceforge.jwebunit.WebTestCase 类的重要方法

方　　法	说　　明
public TestContext getTestContext()	得到测试用例的上下文。可以用它访问诸如地区、基本 URL 和 cookie 之类的项目
public void beginAt(String relativeURL)	在相对于基本 URL 的 URL 处开始对话
public void setWorkingForm(String nameOrId)	与指定的表单开始交互。如果当前页面只有一个表单，就不需要调用这个方法
protected void submit()	提交表单——等同于单击表单的“提交”按钮
public void gotoFrame(String frameName)	激活命名帧

另一个重要的类是 net.sourceforge.jwebunit.TestContext，它用于为测试创建上下文。可以用这个类来处理诸如 cookie、会话和授权之类的信息。表 8-2 显示了这个类的一些重要方法。

表 8-2　net.sourceforge.jwebunit.TestContext 类的重要方法

方　　法	说　　明
public void addCookie(String name, String value)	向测试上下文中添加 cookie。在 HttpUnitDialog 开始时，添加的 cookie 被设置到 WebConversation 上
public void setResourceBundleName(String name)	为测试上下文设置一个使用的资源绑定。用于按照 WebTester 中的键查找期望的值
public void setProxyName(String proxyName)	为测试上下文设置代理服务器名称
public void setBaseUrl(String url)	为测试上下文设置基本 URL

8.3.3　JWebUnit 测试应用举例

现在我们通过例子来介绍 JWebUnit API 的实际应用。我们考察的应用程序是一个测试用例，用于打开一个 Google 搜索页面并搜索文本 *HttpUnit*。应用程序需要测试以下场景：

- 打开 Google 主页 http://www.google.com。
- 确定该页包含一个名为 q 的表单元素(在 Google 的主页上，名为 q 的文本框用于接收用户的查询输入条件)。应用程序用这个元素输入搜索参数。
- 在搜索文本框中输入字符串“HttpUnit Home“，并提交表单。
- 获得结果页，并确定该页面包含的链接中包含文本 *HttpUnit Home*。
- 单击包含文本 HttpUnit Home 的链接。

现在测试场景已经就绪，可以编写 Java 应用程序，用 JWebUnit 实现这些需求了。

第一步是声明一个从 WebTestCase 扩展而来的类，如(1)所示：

(1) 声明测试用例类：

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

正如我们在前面提到过的，JWebUnit 要求每个测试用例都是从 WebTestCase 中扩展而来的。searchLink 保存传入的搜索参数。这个值以命令行参数的形式传递给测试用例。

下一步是声明入口点——main()方法，如(2)所示。

(2) 声明 main()方法：

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

main()方法调用 junit.textui.TestRunner.run()来执行测试用例。因为需要运行 GoogleTest 测试用例，所以作为参数传递给 run()方法的测试套件采用 GoogleTest.class 作为参数。

接下来，浏览用例调用 setUp()方法来设置基本 URL 和代理，如(3)所示。

(3) 设置基本 URL 和代理：

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

上面把基本 URL 设置为 http://www.google.com，这意味着测试用例的启动是相对于这个 URL 的。接下来的两条语句设置连接到 Internet 的代理主机和代理端口，如果是直接连接到 Internet，那么可以忽略代理设置语句。

现在开始浏览站点并输入搜索参数。

(4) 显示了访问 Web 页面，然后测试所有场景的代码：

(5) 测试所有场景

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

上述代码连接到基本 URL，并相对于“/”开始浏览。然后它断定页面中包含一个名为 q 的表单元素——q 是 Google 主页上查询输入文本框的名称。下一条语句用值 HttpUnit

来设置名为 q 的文本框。再下一条语句提交表单上名为 btnG 的提交按钮(在 Google 主页上，名为 btnG 的按钮是标签为“Google Search”的按钮)。表单是在这个对话中提交的，下一页列出搜索结果。在结果页面上，代码首先检查是否有一个链接的文本是 *HttpUnit Home*。如果该链接不存在，那么测试就以 AssertionFailedError 失败。如果该链接存在，则测试执行的下一个操作是单击链接。

我们在测试环境中编写上述的完整 Java 测试程序：

```
package com.jweb.test;
import junit.framework.TestSuite;
import junit.textui.TestRunner;
import net.sourceforge.jwebunit.TestContext;
import net.sourceforge.jwebunit.WebTestCase;
public class GoogleTest extends WebTestCase
{
    static String searchLink = "";
    public static void main(String[] args) {
        searchLink = args[0];
        TestRunner.run(new TestSuite(GoogleTest.class));
    }
    public void setUp() {
        getTestContext().setBaseUrl("http://www.google.com");
        getTestContext().setProxyName("proxy.host.com");
        getTestContext().setProxyPort(80);
    }
    public void testSearch() {
        beginAt("/");
        assertFormElementPresent("q");
        setFormElement("q", "HttpUnit");
        submit("btnG");
        assertLinkPresentWithText(searchLink);
        clickLinkWithText(searchLink);
    }
}
```

在控制台执行 java com.jweb.test.GoogleTest "HttpUnit Home"命令，运行程序。

在执行了测试用例之后，会在命令行输出一个测试用例报告。如果测试失败，报告将看起来如(6)中所示。

(6) 带有断言失败的输出：

```
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
```

正如在(6)中可以看到的，可以以 HttpUnit Hwee 作为参数来执行测试用例。这个测试用例遇到断言的地方会失败，因为结果页面中不包含带有这个文本的链接。由此也就产生了 junit.framework.AssertionFailedError。

在(7)中执行时以 HttpUnit Home 作为参数。测试用例找到了一个带有这个文本的链接，所以测试通过了。

(7) 成功测试的输出：

```
C:\temp>java com.jweb.test.GoogleTest "HttpUnit Home"
.
Time: 6.991
OK (1 test)
```

8.3.4　JWebUnit 应用小结

前面通过讨论 JWebUnit 框架的一些突出特性和最重要的类，介绍了如何用它创建简洁的测试用例，使大家能够对 JWebUnit 框架有一个认识。JWebUnit 还有更多特性可以用在测试用例中。它支持测试 Web 页面中的链接行数，可以对字符串、表或者带有指定标签的表单输入元素是否存在于页面上进行断言。此外，JWebUnit 还可以处理 cookie(例如断言存在某个 cookie、删除 cookie 等)。测试可以对某个文本之后出现的特定文本的链接进行单击。如果想为 Web 应用程序构建快而有效的测试用例，JWebUnit 可能是最好的工具。

实验习题

1. 应用 HtmlUnit 对 Web 应用进行测试，并给出具体的测试过程，同时与 HttpUnit 进行比较。

2. 在 Eclipse 环境下建立 swtbot 的 Web 应用测试环境，并对具体的 Web 应用进行测试，详细描述测试过程。