3.5 Simple HTML-Building Utilities

As you probably already know, an HTML document is structured as follows:

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>...</TITLE>...</HEAD>
<BODY ...>...</BODY>
</HTML>
```

When using servlets to build the HTML, you might be tempted to omit part of this structure, especially the `DOCTYPE` line, noting that virtually all major browsers ignore it even though the HTML specifications require it. We strongly discourage this practice. The advantage of the `DOCTYPE` line is that it tells HTML validators which version of HTML you are using so they know which specification to check your document against. These validators are valuable debugging services, helping you catch HTML syntax errors that your browser guesses well on but that other browsers will have trouble displaying.

The two most popular online validators are the ones from the World Wide Web Consortium (http://validator.w3.org/) and from the Web Design Group (http://www.htmlhelp.com/tools/validator/). They let you submit a URL, then they retrieve the page, check the syntax against the formal HTML specification, and report any errors to you. Since, to a client, a servlet that generates HTML looks exactly like a regular Web page, it can be validated in the normal manner unless it requires `POST` data to return its result. Since `GET` data is attached to the URL, you can even send the validators a URL that includes `GET` data. If the servlet is available only inside your corporate firewall, simply run it, save the HTML to disk, and choose the validator's File Upload option.

Core Approach

Use an HTML validator to check the syntax of pages that your servlets generate.

Admittedly, it is sometimes a bit cumbersome to generate HTML with `println` statements, especially long tedious lines like the `DOCTYPE` declaration. Some people address this problem by writing lengthy HTML-generation utilities, then use the utilities throughout their servlets. We're skeptical of the usefulness of such an extensive library. First and foremost, the inconvenience of generating HTML programmatically is one of the main problems addressed by JavaServer Pages (see Chapter 10, "Overview of JSP Technology"). Second, HTML generation routines can be cumbersome and tend not to support the full range of HTML attributes (`CLASS` and `ID` for style sheets, JavaScript event handlers, table cell background colors, and so forth).

Despite the questionable value of a full-blown HTML generation library, if you find you're repeating the same constructs many times, you might as well create a simple utility class that simplifies those constructs. After all, you're working with the Java programming language; don't forget the standard object-oriented programming principle of reusing, not repeating, code. Repeating identical or nearly identical code means that you have to change the code lots of different places when you inevitably change your approach.

For standard servlets, two parts of the Web page (`DOCTYPE` and `HEAD`) are unlikely to change and thus could benefit from being incorporated into a simple utility file. These are shown in Listing 3.5, with Listing 3.6 showing a variation of the `HelloServlet` class that makes use of

this utility. We'll add a few more utilities throughout the book.

Listing 3.5 coreservlets/ServletUtilities.java

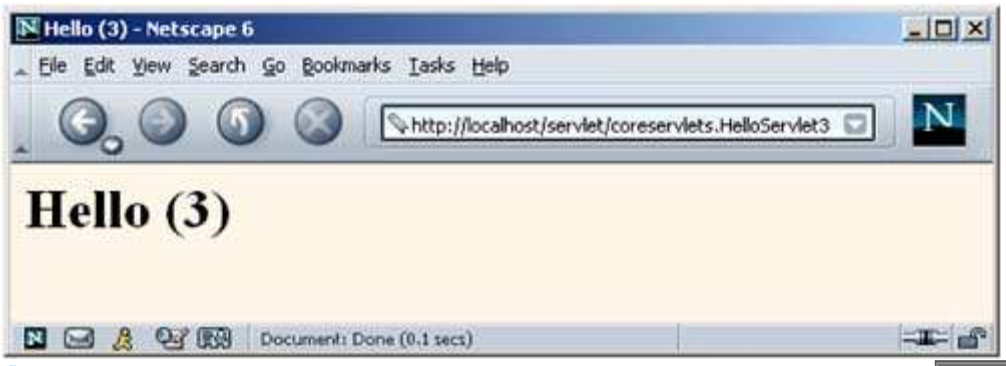
```
package coreservlets;

import javax.servlet.*;
import javax.servlet.http.*;

/** Some simple time savers. Note that most are static methods. */

public class ServletUtilities {
  public static final String DOCTYPE =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">";

 public static String headWithTitle(String title) {
    return(DOCTYPE + "\n" +
           "<HTML>\n" +
           "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
  }

  ...
}
```

Listing 3.6 coreservlets/HelloServlet3.java

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Simple servlet for testing the use of packages
 *  and utilities from the same package.
 */

public class HelloServlet3 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Hello (3)";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>" + title + "</H1>\n" +
                "</BODY></HTML>");
  }
}
```

After you compile `HelloServlet3.java` (which results in `ServletUtilities.java` being compiled automatically), you need to move the two class files to the `coreservlets` subdirectory of the server's default deployment location (`.../WEB-INF/classes`; review Section 2.8 for details). If you get an "Unresolved symbol" error when compiling `HelloServlet3.java`, go back and review the `CLASSPATH` settings described in Section 2.7 (Set Up Your Development Environment), especially the part about including the top-level development directory in the `CLASSPATH`. Figure 3-5 shows the result when the servlet is invoked with the default URL.

Figure 3-5. Result of `http://localhost/servlet/coreservlets.HelloServlet3`.



[Team LiB]
PREVIOUS NEXT