CONTENTS

Chapter 5. Session Management

The Hypertext Transfer Protocol (HTTP) is the network protocol that web servers and client browsers use to communicate with each other. HTTP is the language of the web. HTTP connections are initiated by a client browser that sends an HTTP request. The web server then responds with an HTTP response and closes the connection. If the same client requests another resource from the server, it must open another HTTP connection to the server. The server always closes the connection as soon as it sends the response, whether or not the browser user needs some other resource from the server.

This process is similar to a telephone conversation in which the receiver always hangs up after responding to the last remark/question from the caller. For example, a call goes something like this:

> Caller dials. Caller gets connected.
>
> Caller: "Hi, good morning."
>
> Receiver: "Good morning."
>
> Receiver hangs up.
>
> Caller dials again. Caller gets connected.
>
> Caller: "May I speak to Dr. Zeus, please?"
>
> Receiver: "Sure."
>
> Receiver hangs up.
>
> Caller dials again, and so on, and so on.

Putting this in a web perspective, because the web server always disconnects after it responds to a request, the web server does not know whether a request comes from a user who has just requested the first page or from a user who has requested nine other pages before. As such, HTTP is said to be *stateless*.

Being stateless has huge implications. Consider, for example, a user who is shopping at an online store. As usual, the process starts with the user searching for a product. If the product is found, the user then enters the quantity of that product into the shopping cart form and submits it to the server. But, the user is not yet checking out—she still wants to buy something else. So she searches the catalog again for the second product.The first product order has now been lost, however, because the previous connection was closed and the web server does not remember anything about the previous connection.

The good news is that web programmers can work around this, and this chapter discusses techniques for that. The solution is called *user session management*. The web server is forced to associate HTTP requests and client browsers.

What Is Session Management?

You know that HTTP statelessness has a deep impact on web application programming. To see the problem more clearly, consider the LoginServlet in Chapter 3, "Writing Servlet Applications," and Chapter 4, "Accessing Databases with JDBC." The skeleton is presented here:

```
public void doGet(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {
  sendLoginForm(response, false);
}

public void doPost(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {

  String userName = request.getParameter("userName");
  String password = request.getParameter("password");
```

```
  if (login(userName, password)) {
    // login successful, display the information
  }
  else {
    // login failed, re-send the Login form
    sendLoginForm(response, true);
  }
}
```

The Login servlet is used to require users to enter a valid user name and pass-word before they can see some information. When a user first requests the servlet, the Login form is displayed. The user then can enter a user name and password and submit the form. Assuming that the form uses the POST method, the user information is captured in the doPost method, which does the authentication by calling the login method. If the login was successful, the information is displayed. If not, the Login form is sent again.

What if you have another servlet that also only allows authorized users to view the information? This second servlet does not know whether the same user has successfully logged in to the first servlet. Consequently, the user will be required to log in again!

This is, of course, not practical. Every time a user goes to request a protected servlet, he or she has to login again even though all the servlets are part of the same application. This easily pushes the user to the edge of frustration and most likely results in a lost customer.

Fortunately, there are ways to get around this, using techniques for remembering a user's session. Once users have logged in, they do not have to login again. The application will remember them. This is called *session management*.

Session management, also called *session tracking*, goes beyond simply remembering a user who has successfully logged in. Anything that makes the application remember information that has been entered or requested by the user can be considered session management. Session management does not change the nature of HTTP statelessness—it simply provides a way around it.

By principle, you manage a user's session by performing the following to servlets/pages that need to remember a user's state:

1. When the user requests a servlet, in addition to sending the response, the servlet also sends a token or an identifier.
2. If the user does not come back with the next request for the same or a different servlet, that is fine. If the user does come back, the token or identifier is sent back to the server. Upon encountering the token, the next servlet should recognize the identifier and can do a certain action based on the token. When the servlet responds to the request, it also sends the same or a different token. This goes on and on with all the servlets that need to remember a user's session.

You will use four techniques for session management. They operate based on the same principle, although what is passed and how it is passed is different from one to another. The techniques are as follows:

- URL rewriting
- Hidden fields
- Cookies
- Session objects

Which technique you use depends on what you need to do in your application. Each of the techniques is discussed in the sections below. The section, "Knowing Which Technique to Use," concludes the chapter.

URL Rewriting

With URL rewriting, you append a token or identifier to the URL of the next servlet or the next resource. You can send parameter name/value pairs using the following format:

```
url?name1=value1&name2=value2&…
```

A name and a value is separated using an equal sign (=); a parameter name/value pair is separated from another parameter name/value pair using the ampersand (&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a servlet, you can use the HttpServletRequest interface's getParameter method to obtain a parameter value. For instance, to obtain the value of the second parameter, you write the following:

```
request.getParameter(name2);
```

The use of URL rewriting is easy. When using this technique, however, you need to consider several things:

- The number of characters that can be passed in a URL is limited. Typically, a browser can pass up to 2,000 characters.
- The value that you pass can be seen in the URL. Sometimes this is not desirable. For example, some people prefer their password not to appear on the URL.

- You need to encode certain characters—such as & and ? characters and white spaces—that you append to a URL.

As an example, you will build an application that you can use to administer all registered persons in a database. The application uses the Users table created in Chapter 4 and allows you to do the following:

1. Enter a keyword that will become the search key for the first name and the last name columns of the Users table.
2. Display all persons that match the keyword.
3. In addition to the data for each record, there are Delete and Update hyperlinks. The user id is included in the hyperlinks.
4. If the Delete hyperlink is clicked, the corresponding person will be deleted from the Users table.
5. If the Update hyperlink is clicked, the corresponding person's details will be displayed in a form. You then can change the data and submit the form to update that person's record.

The Administration application is shown in Figures 5.1, 5.2, and 5.3.

Figure 5.1. The Administration's Search page.

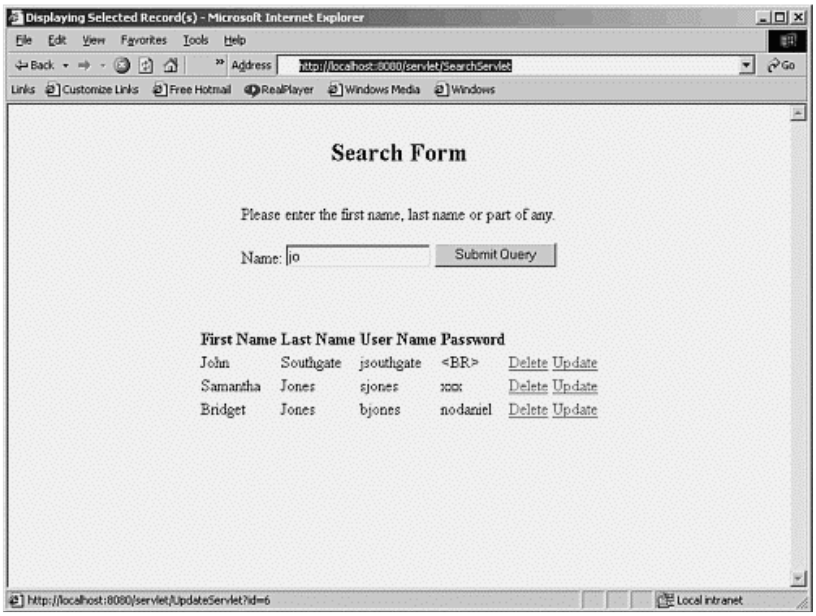


Figure 5.2. The Administrations Delete page.

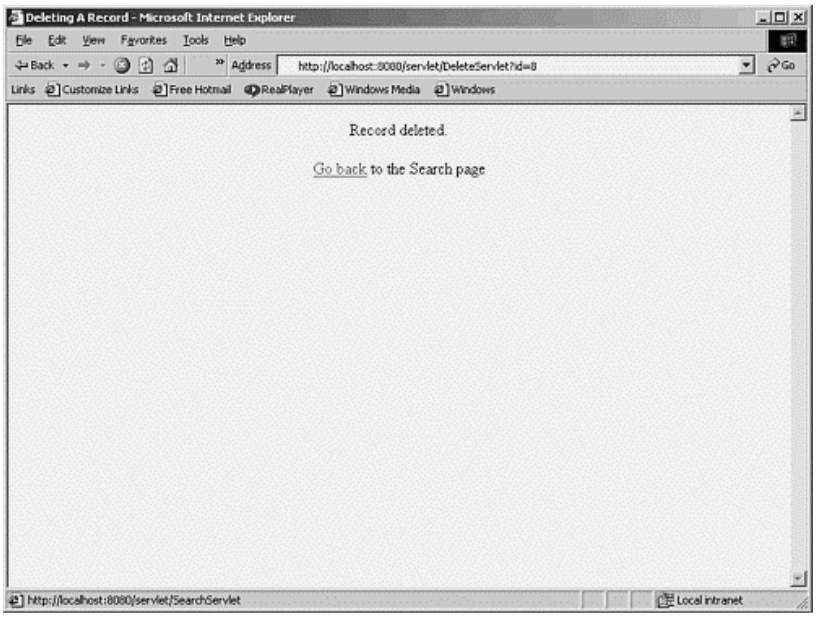
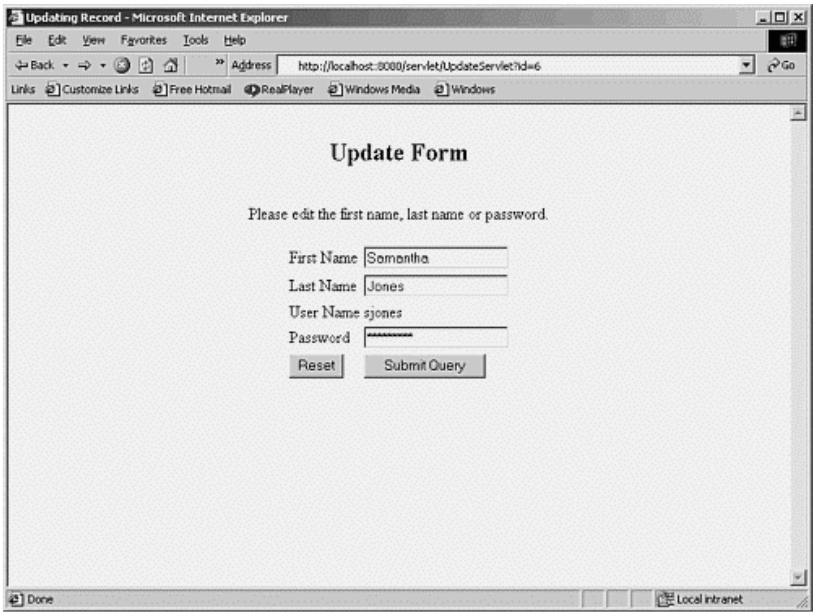


Figure 5.3. The Administration's Update page.



The first servlet, the SearchServlet, is similar to the one in Chapter 4. The doGet method sends the form for the user to type in a keyword. This keyword could be a first name, a last name, or part of a first name or a last name. To allow more flexibility, this example separates the form from the rest of the page. Therefore, you send the page header and page footer separately. The doGet method is given in Listing 5.1.

Listing 5.1 The doGet Method of the SearchServlet

```
public void doGet(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {
    sendPageHeader(response);
    sendSearchForm(response);
    sendPageFooter(response);
  }
```

The page header contains the head part of the HTML page, including the page title and the opening <BODY> tag. To send the page header, you call the sendPageHeader method. The page footer contains the bottom part of the page, that is, the closing </BODY> and </HTML> tags. Calling the sendPageFooter method will send the page footer.

The sendSearchForm method sends the HTML form. This form uses the POST method and, when submitted, this form will send the keyword to the server. The server then invokes the doPost method whose code is given in Listing 5.2.

Listing 5.2 The doPost Method of the SearchServlet

```
public void doPost(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {

  keyword = request.getParameter("keyword");
  sendPageHeader(response);
  sendSearchForm(response);
  sendSearchResult(response);
  sendPageFooter(response);
}
```

The first thing doPost does is retrieve the keyword from the HttpServletRequest object and store it in a class-level variable keyword. It then sends the page header, the search form, the search result, and the page footer to the browser.

The sendSearchResult method forms an SQL select statement that incorporates the keyword, in the following syntax:

```
SELECT Id, FirstName, LastName, UserName, Password
FROM Users
WHERE FirstName LIKE '%keyword%'
OR LastName LIKE '%keyword%'
```

Note that the % character represents any text of zero length or more. This means that jo% will find jo, john, jones, and so on. Note, however, that this wildcard character is different from one database server to another and you should consult your database server documentation before using it.

Because the value entered by the user as the keyword can contain a single quote character, you use the StringUtil class's fixSqlFieldValue method to "fix" the keyword. StringUtil class is part of the com.brainysoftware.java package that can be found on the accompanying CD. You need to copy the StringUtil.java file into the com/brainysoftware/java/ directory under the directory where you put your source files. See Chapter 3 for more information on how to use the StringUtil class.

You compose the SQL statement using the following code:

```
String sql =
  "SELECT Id, FirstName, LastName, UserName, Password" +
  " FROM Users" +
  " WHERE FirstName LIKE '%" +
  StringUtil.fixSqlFieldValue(keyword) + "%'" +
  " OR LastName LIKE '%" +
```

Once the SQL statement is executed, the returned ResultSet can be looped through to get its cell data. Of particular interest is the call to the getString method passing the integer 1. This returns the Id for that person. The Id is important because it is used as the token for that person. The Id is passed in the URL to the DeleteServlet as well as to the UpdateServlet, as follows:

```
out.println("<TD><A HREF=DeleteServlet?id=" + id +
  ">Delete</A></TD>");
out.println("<TD><A HREF=UpdateServlet?id=" + id +
  ">Update</A></TD>");
```

Therefore, for a person with an Id of 6, the hyperlink to the DeleteServlet will be:

```
<A HREF=DeleteServlet?id=6>Delete</A>
```

And for a person with an Id of 8, the hyperlink to the UpdateServlet is

```
<A HREF=UpdateServlet?id=8>Delete</A>
```

See how information has been added to the URL?

The complete listing of the SearchServlet is presented in Listing 5.3.

Listing 5.3 The SearchServlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import com.brainysoftware.java.StringUtil;

public class SearchServlet extends HttpServlet {

  private String keyword = "";

  public void init() {
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      System.out.println("JDBC driver loaded");
    }
    catch (ClassNotFoundException e) {
      System.out.println(e.toString());
    }
  }

  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {
    sendPageHeader(response);
    sendSearchForm(response);
    sendPageFooter(response);
  }

  /**Process the HTTP Post request*/
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {

    keyword = request.getParameter("keyword");
```

```
    sendPageHeader(response);
    sendSearchForm(response);
    sendSearchResult(response);
    sendPageFooter(response);
  }

  void sendSearchResult(HttpServletResponse response)
    throws IOException {
    PrintWriter out = response.getWriter();
    try {
      Connection con =
        DriverManager.getConnection("jdbc:odbc:JavaWeb");
      System.out.println("got connection");

      Statement s = con.createStatement();

      out.println("<TABLE>");
      out.println("<TR>");
      out.println("<TH>First Name</TH>");
      out.println("<TH>Last Name</TH>");
      out.println("<TH>User Name</TH>");
      out.println("<TH>Password</TH>");
      out.println("<TH></TH>");
      out.println("<TH></TH>");
      out.println("</TR>");
      String sql =
        "SELECT Id, FirstName, LastName, UserName, Password" +
        " FROM Users" +
        " WHERE FirstName LIKE '%" +
        StringUtil.fixSqlFieldValue(keyword) + "%'" +
        " OR LastName LIKE '%" +
        StringUtil.fixSqlFieldValue(keyword) + "%'";
      ResultSet rs = s.executeQuery(sql);
      while (rs.next()) {
        String id = rs.getString(1);
        out.println("<TR>");
        out.println("<TD>" + StringUtil.encodeHtmlTag(rs.getString(2)) + "</TD>");
        out.println("<TD>" + StringUtil.encodeHtmlTag(rs.getString(3)) + "</TD>");
        out.println("<TD>" + StringUtil.encodeHtmlTag(rs.getString(4)) + "</TD>");
        out.println("<TD>" + StringUtil.encodeHtmlTag(rs.getString(5)) + "</TD>");
        out.println("<TD><A HREF=DeleteServlet?id=" + id + ">Delete</A></TD>");
        out.println("<TD><A HREF=UpdateServlet?id=" + id + ">Update</A></TD>");
        out.println("</TR>");
      }
      s.close();
      con.close();
    }
    catch (SQLException e) {
    }
    catch (Exception e) {
    }
    out.println("</TABLE>");
  }

  /**
  * Send the HTML page header, including the title
   * and the <BODY> tag
   */
  private void sendPageHeader(HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Displaying Selected Record(s)</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<CENTER>");
  }

  /**
```

```
   * Send the HTML page footer, i.e. the </BODY>
   * and the </HTML>
   */
  private void sendPageFooter(HttpServletResponse response)
    throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("</CENTER>");
    out.println("</BODY>");
    out.println("</HTML>");
  }

  /**Send the form where the user can type in
   * the details for a new user
   */
  private void sendSearchForm(HttpServletResponse response)
    throws IOException {

    PrintWriter out = response.getWriter();
    out.println("<BR><H2>Search Form</H2>");
    out.println("<BR>Please enter the first name, last name or part of any.");
    out.println("<BR>");
    out.println("<BR><FORM METHOD=POST>");
    out.print("Name: <INPUT TYPE=TEXT Name=keyword");
    out.print(" VALUE=\"" + StringUtil.encodeHtmlTag(keyword) + "\"");
    out.println(">");
    out.println("<INPUT TYPE=SUBMIT>");
    out.println("</FORM>");
    out.println("<BR>");
    out.println("<BR>");
  }
}
```

The DeleteServlet takes the value of id appended to the URL and deletes the record of the person having that id.

Retrieving the id is achieved by using the getParameter method of the HttpServletRequest interface:

```
String id = request.getParameter("id");
```

After you get the id, an SQL statement can be composed, as follows:

```
String sql = "DELETE FROM Users WHERE Id=" + id;
```

Next, you can create a Connection object and a Statement object, and use the latter to execute the SQL statement:

```
Connection con = DriverManager.getConnection("jdbc:odbc:JavaWeb");
Statement s = con.createStatement();
recordAffected = s.executeUpdate(sql);
```

The code for the DeleteServlet is given in Listing 5.4. Note that in the DeleteServlet, and in the UpdateServlet, there is no code for loading the JDBC driver. This has been done in the SearchServlets and the driver stays on for other servlets that need to connect to the same database.

Listing 5.4 The DeleteServlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;

public class DeleteServlet extends HttpServlet {
  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {
    int recordAffected = 0;
    try {
      String id = request.getParameter("id");
      String sql = "DELETE FROM Users WHERE Id=" + id;
      Connection con =
        DriverManager.getConnection("jdbc:odbc:JavaWeb");
```

```
      Statement s = con.createStatement();
      recordAffected = s.executeUpdate(sql);
      s.close();
      con.close();
    }
    catch (SQLException e) {
    }
    catch (Exception e) {
    }
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Deleting A Record</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<CENTER>");
    if (recordAffected==1)
      out.println("<P>Record deleted.</P>");
    else
      out.println("<P>Error deleting record.</P>");
    out.println("<A HREF=SearchServlet>Go back</A> to the Search page");
  }
}
```

The UpdateServlet gets the id passed in the URL and sends a form containing the user details of the person with that id. The user can update the first name, the last name, or the password for that person, but not the user name. Therefore, the first name, last name, and the password are represented in text boxes, whereas the user name is displayed as HTML text.

The business rule for this servlet states that the user name cannot be changed because it's been guaranteed unique at the time of data insertion. Allowing this value to change can break this restriction. Of course, how you implement a servlet depends on your own requirement and business rules. The UpdateServlet is called by clicking the URL in the SearchServlet. The URL always carries an id for the person whose details are to be changed. This request invokes the UpdateServlet's doGet method that sends the page header, the update form, and the page footer. The doGet method is presented in Listing 5.5.

Listing 5.5 The doGet Method of the UpdateServlet

```
public void doGet(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {
    sendPageHeader(response);
    sendUpdateForm(request, response);
    sendPageFooter(response);
  }
```

sendUpdateForm first retrieves the id from the URL using the getParameter method, as follows:

```
String id = request.getParameter("id");
```

The method then connects to the database to retrieve the details from the Users table. The SQL statement for that is simply the following:

```
SELECT FirstName, LastName, UserName, Password
FROM Users
WHERE Id=id
```

Then the method will send the details in an HTML form, similar to the one shown earlier in Figure 5.3.

The interesting part of the code is when the <FORM> tag is sent:

```
out.println("<BR><FORM METHOD=POST ACTION=" +
  request.getRequestURI() + "?id=" + id + ">");
```

Normally, you won't have an ACTION attribute in your form because the form is to be submitted to the same servlet. This time, however, you need to append the value of the id to the URL. Therefore, you use the getRequestURI method to get the current Uniform Resource Identifier (URI) and append the following information, such as

```
?id=6
```

for the user with an id of 6.

When the user submits the form, the doPost method of the UpdateServlet will be invoked. The doPost method is given in Listing 5.6.

Listing 5.6 The doPost Method of the UpdateServlet

```
public void doPost(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {

  sendPageHeader(response);
  updateRecord(request, response);
  sendPageFooter(response);
}
```

The important method called from doPost is the updateRecord method. The updateRecord method first retrieves the values of id, firstName, lastName, and password using the getParameter method. Note that the id is retrieved from the URL and the others from the request body, as follows:

```
String id = request.getParameter("id");
String firstName = request.getParameter("firstName");
String lastName = request.getParameter("lastName");
String password = request.getParameter("password");
```

With these values, you can compose the SQL that will update the record. It has the following syntax:

```
UPDATE Users
SET FirstName=firstName,
LastName=lastName,
Password=password
WHERE Id=id
```

Then you can execute the SQL as usual:

```
Connection con = DriverManager.getConnection(dbUrl);
Statement s = con.createStatement();
int i = s.executeUpdate(sql);
```

Now, the executeUpdate should return the number of records affected by the SQL statement. Because the id is unique, it should return 1 as the number of records affected. If it is 1, you simply send a message saying, "Record updated". If it is not 1 because an unexpected error occurred, say, "Error updating record".

```
if (i==1)
  out.println("Record updated");
else
  out.println("Error updating record");
```

The complete code for the UpdateServlet is given in Listing 5.7.

Listing 5.7 The UpdateServlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import com.brainysoftware.java.StringUtil;

public class UpdateServlet extends HttpServlet {
  private String dbUrl = "jdbc:odbc:JavaWeb";

  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {
    sendPageHeader(response);
    sendUpdateForm(request, response);
    sendPageFooter(response);
  }

  /**Process the HTTP Post request*/
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {
    sendPageHeader(response);
    updateRecord(request, response);
```

```
    sendPageFooter(response);
  }


  /**
   * Send the HTML page header, including the title
   * and the <BODY> tag
   */
  private void sendPageHeader(HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Updating Record</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<CENTER>");
  }

  /**
   * Send the HTML page footer, i.e. the </BODY>
   * and the </HTML>
   */
  private void sendPageFooter(HttpServletResponse response)
    throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("</CENTER>");
    out.println("</BODY>");
    out.println("</HTML>");
  }

  /**Send the form where the user can type in
   * the details for a new user
   */
  private void sendUpdateForm(HttpServletRequest request, HttpServletResponse response)
    throws IOException {
    String id = request.getParameter("id");
    PrintWriter out = response.getWriter();
    out.println("<BR><H2>Update Form</H2>");
    out.println("<BR>Please edit the first name, last name or password.");
    out.println("<BR>");
    try {
      String sql = "SELECT FirstName, LastName," +
        " UserName, Password" +
        " FROM Users" +
        " WHERE Id=" + id;
        Connection con = DriverManager.getConnection(dbUrl);
        Statement s = con.createStatement();
        ResultSet rs = s.executeQuery(sql);
        if (rs.next()) {
          String firstName = rs.getString(1);
          String lastName = rs.getString(2);
          String userName = rs.getString(3);
          String password = rs.getString(4);

          out.println("<BR><FORM METHOD=POST ACTION=" +
            request.getRequestURI() + "?id=" + id + ">");
          out.println("<TABLE>");
          out.println("<TR>");
          out.println("<TD>First Name</TD>");
          out.print("<TD><INPUT TYPE=TEXT Name=firstName");
          out.print(" VALUE=\"" + StringUtil.encodeHtmlTag(firstName) + "\"");
          out.println("></TD>");
          out.println("</TR>");
          out.println("<TR>");
          out.println("<TD>Last Name</TD>");
          out.print("<TD><INPUT TYPE=TEXT Name=lastName");
          out.print(" VALUE=\"" + StringUtil.encodeHtmlTag(lastName) + "\"");
          out.println("></TD>");
          out.println("</TR>");
```

```
          out.println("<TR>");
          out.println("<TD>User Name</TD>");
          out.print("<TD>" + StringUtil.encodeHtmlTag(userName) + "</TD>");
          out.println("</TR>");
          out.println("<TR>");
          out.println("<TD>Password</TD>");
          out.print("<TD><INPUT TYPE=PASSWORD Name=password");
          out.print(" VALUE=\"" + StringUtil.encodeHtmlTag (password) + "\"");
          out.println("></TD>");
          out.println("</TR>");
          out.println("<TR>");
          out.println("<TD><INPUT TYPE=RESET></TD>");
          out.println("<TD><INPUT TYPE=SUBMIT></TD>");
          out.println("</TR>");
          out.println("</TABLE>");
          out.println("</FORM>");
        }
        s.close();
        con.close();
      }
      catch (SQLException e) {
        out.println(e.toString());
      }
    catch (Exception e) {
      out.println(e.toString());
    }
  }

  void updateRecord(HttpServletRequest request, HttpServletResponse response)
    throws IOException {
    String id = request.getParameter("id");
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    String password = request.getParameter("password");
    PrintWriter out = response.getWriter();
    try {
      String sql = "UPDATE Users" +
        " SET FirstName='" + StringUtil.fixSqlFieldValue(firstName) + "'," +
        " LastName='" + StringUtil.fixSqlFieldValue(lastName) + "'," +
        " Password='" + StringUtil.fixSqlFieldValue(password) + "'" +
        " WHERE Id=" + id;
      Connection con = DriverManager.getConnection(dbUrl);
      Statement s = con.createStatement();
      int i = s.executeUpdate(sql);
      if (i==1)
        out.println("Record updated");
      else
        out.println("Error updating record");
      s.close();
      con.close();
    }
    catch (SQLException e) {
      out.println(e.toString());
    }
    catch (Exception e) {
      out.println(e.toString());
    }
    out.println("<A HREF=SearchServlet>Go back</A> to the Search Page");
  }
}
```

Hidden Fields

Another technique for managing user sessions is by passing a token as the value for an HTML hidden field. Unlike the URL rewriting, the value does not show on the URL but can still be read by viewing the HTML source code. Although this method also is easy to use, an HTML form is always required.

Using Hidden Fields

As the first example to illustrate the use of this technique, you will modify the sendUpdateForm method in the UpdateServlet in Listing 5.7.

To try this example, you need to change the sendUpdateForm method in the UpdateServlet. This servlet should still be used with the SearchServlet and the DeleteServlet, given in Listings 5.3 and 5.4. The two servlets are not modified, and the code won't be repeated here.

The new sendUpdateForm method is given in Listing 5.8.

Listing 5.8 The Modified sendUpdateForm Method in UpdateServlet

```
private void sendUpdateForm(HttpServletRequest request, HttpServletResponse response)
    throws IOException {
    String id = request.getParameter("id");
    PrintWriter out = response.getWriter();
    out.println("<BR><H2>Update Form</H2>");
    out.println("<BR>Please edit the first name, last name or password.");
    out.println("<BR>");
    try {
      String sql = "SELECT FirstName, LastName," +
        " UserName, Password" +
        " FROM Users" +
        " WHERE Id=" + id;
      Connection con = DriverManager.getConnection(dbUrl);
      Statement s = con.createStatement();
      ResultSet rs = s.executeQuery(sql);
      if (rs.next()) {
        String firstName = rs.getString(1);
        String lastName = rs.getString(2);
        String userName = rs.getString(3);
        String password = rs.getString(4);

        out.println("<BR><FORM METHOD=POST>");
        out.print("<INPUT TYPE=HIDDEN Name=id VALUE=" + id + ">");
        out.println("<TABLE>");
        out.println("<TR>");
        out.println("<TD>First Name</TD>");
        out.print("<TD><INPUT TYPE=TEXT Name=firstName");
        out.print(" VALUE=\"" + StringUtil.encodeHtmlTag(firstName) + "\"");
        out.println("></TD>");
        out.println("</TR>");
        out.println("<TR>");
        out.println("<TD>Last Name</TD>");
        out.print("<TD><INPUT TYPE=TEXT Name=lastName");
        out.print(" VALUE=\"" + StringUtil.encodeHtmlTag(lastName) + "\"");
        out.println("></TD>");
        out.println("</TR>");
        out.println("<TR>");
        out.println("<TD>User Name</TD>");
        out.print("<TD>" + StringUtil.encodeHtmlTag(userName) + "</TD>");
        out.println("</TR>");
        out.println("<TR>");
        out.println("<TD>Password</TD>");
        out.print("<TD><INPUT TYPE=PASSWORD Name=password");
        out.print(" VALUE=\"" + StringUtil.encodeHtmlTag(password) + "\"");
        out.println("></TD>");
        out.println("</TR>");
        out.println("<TR>");
        out.println("<TD><INPUT TYPE=RESET></TD>");
        out.println("<TD><INPUT TYPE=SUBMIT></TD>");
        out.println("</TR>");
        out.println("</TABLE>");
        out.println("</FORM>");
      }
      s.close();
      con.close();
    }
    catch (SQLException e) {
      out.println(e.toString());
    }
    catch (Exception e) {
      out.println(e.toString());
    }
  }
```

If you run this application in a web browser, you will be able to see that the id is not appended to the URL. Therefore, you don't need the ACTION attribute in the form. The id is now written to a HIDDEN field as follows:

```
out.print("<INPUT TYPE=HIDDEN Name=id VALUE=" + id + ">");
```

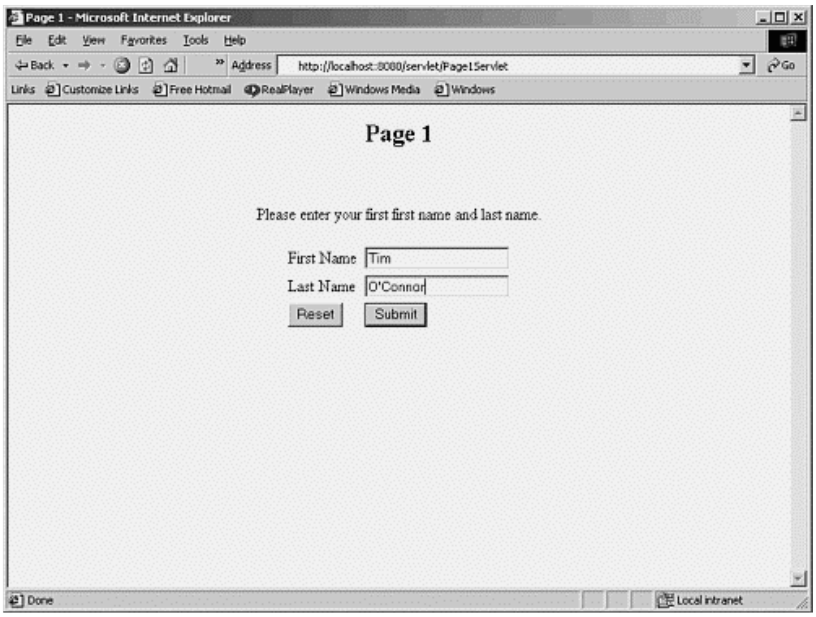
When the form is submitted, the hidden field is sent along with the other input elements' values of the form. On the servlet, the id still can be retrieved using the getParameter method.

Splitting Forms

The second example of the use of the hidden field occurs when you want to split a large form into several smaller ones for the sake of user-friendliness. Again, for this example, you will use the now familiar Users table. You will write an application that the user can use to insert a new record to the Users table, similar to the one in Chapter 4. Instead of having one form in which the user can enter the first name, the last name, the user name, and the password at the same time, however, the form is split into two smaller forms. The first form will accept the first name and the last name. The user name and password can be entered on the second form. When the second form is submitted, all four values must be sent to a third servlet that composes and executes an SQL insert statement.

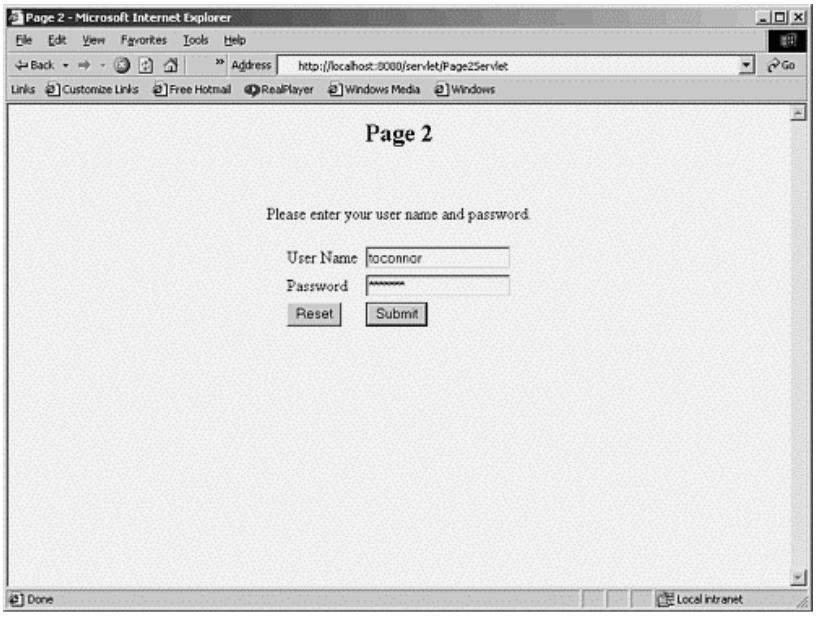
In this example, you will use three servlets for each page in the process. They are simply called Page1Servlet, Page2Servlet, and Page3Servlet. Page1Servlet does not do much except send an HTML form with two text fields. You can replace this servlet with a static HTML file if you want. Page1Servlet is shown in Figure 5.4.

Figure 5.4. Page 1.



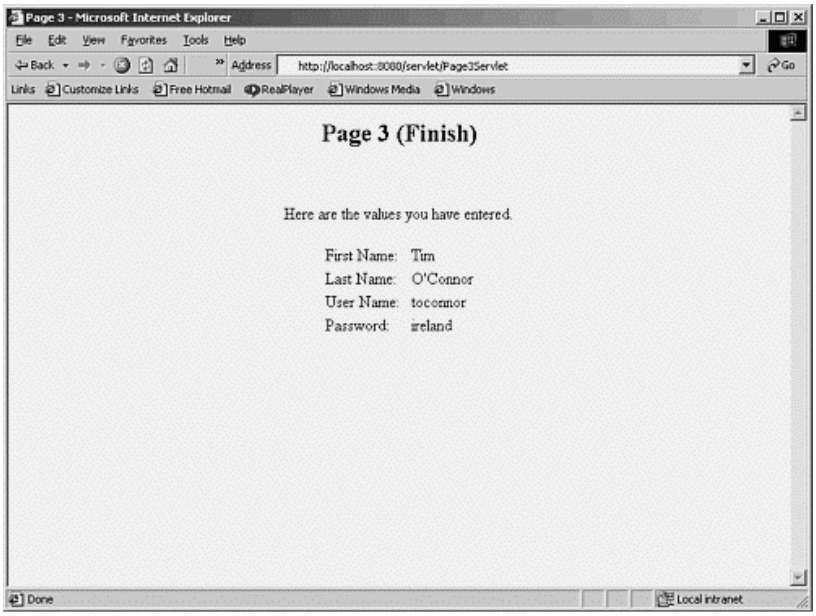
When the form sent by Page1Servlet is submitted, it goes to Page2Servlet. Page2Servlet sends the second form as well as the values from the first form. It is shown in Figure 5.5.

Figure 5.5. Page 2.



When the second form is submitted, all four values go to Page3Servlet. You could insert a new record to the database. For brevity, however, you will simply display the values without trying to access any database. The result from the Page3Servlet is shown in Figure 5.6.

Figure 5.6. Page 3.



Now, let's dissect the code.

As mentioned, Page1Servlet is a simple HTML form. It is given in Listing 5.9.

Listing 5.9 Page1Servlet

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

public class Page1Servlet extends HttpServlet {
  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {
    sendPage1(response);  }

  /**Process the HTTP Post request*/
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws
```

```
    ServletException, IOException {
    sendPage1(response);
  }

  void sendPage1(HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Page 1</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<CENTER>");
    out.println("<H2>Page 1</H2>");
    out.println("<BR>");
    out.println("<BR>");
    out.println("Please enter your first first name and last name.");
    out.println("<BR>");
    out.println("<BR>");
    out.println("<FORM METHOD=POST ACTION=Page2Servlet>");
    out.println("<TABLE>");
    out.println("<TR>");
    out.println("<TD>First Name </TD>");
    out.println("<TD><INPUT TYPE=TEXT NAME=firstName></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>Last Name </TD>");
    out.println("<TD><INPUT TYPE=TEXT NAME=lastName></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD><INPUT TYPE=RESET></TD>");
    out.println("<TD><INPUT TYPE=SUBMIT VALUE=Submit></TD>");
    out.println("</TR>");
    out.println("</TABLE>");
    out.println("</FORM>");
    out.println("</CENTER>");
    out.println("</BODY>");
    out.println("</HTML>");
  }
}
```

One thing to note is that you use the ACTION attribute for the form, as you see here:

```
out.println("<FORM METHOD=POST ACTION=Page2Servlet>");
```

The value for the ACTION attribute is Page2Servlet, which makes sure that the form will be submitted to Page2Servlet.

Page2Servlet retrieves the first name and last name from the form in Page1Servlet and retains them in hidden fields. These hidden fields are included in a form that also sends the two text input for user name and password. The code for this servlet is given in Listing 5.10.

Listing 5.10 Page2Servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.brainysoftware.java.StringUtil;

public class Page2Servlet extends HttpServlet {
  String page1Url = "Page1Servlet";
  String firstName;
  String lastName;

  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.sendRedirect(page1Url);
  }
```

```
  /**Process the HTTP Post request*/
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {
    firstName = request.getParameter("firstName");
    lastName = request.getParameter("lastName");
    if (firstName==null || lastName==null)
      response.sendRedirect(page1Url);

    sendPage2(response);
  }

  void sendPage2(HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Page 2</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<CENTER>");
    out.println("<H2>Page 2</H2>");
    out.println("<BR>");
    out.println("Please enter your user name and password.");
    out.println("<BR>");
    out.println("<BR>");
    out.println("<BR>");
    out.println("<FORM METHOD=POST ACTION=Page3Servlet>");
    out.println("<INPUT TYPE=HIDDEN NAME=firstName VALUE=\"" +
      StringUtil.encodeHtmlTag(firstName) + "\">");
    out.println("<INPUT TYPE=HIDDEN NAME=lastName VALUE=\"" +
      StringUtil.encodeHtmlTag(lastName) + "\">");
    out.println("<TABLE>");
    out.println("<TR>");
    out.println("<TD>User Name </TD>");
    out.println("<TD><INPUT TYPE=TEXT NAME=userName></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>Password </TD>");
    out.println("<TD><INPUT TYPE=PASSWORD NAME=password></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD><INPUT TYPE=RESET></TD>");
    out.println("<TD><INPUT TYPE=SUBMIT VALUE=Submit></TD>");
    out.println("</TR>");
    out.println("</TABLE>");
    out.println("</FORM>");
    out.println("</CENTER>");
    out.println("</BODY>");
    out.println("</HTML>");
  }
}
```

If you enter "Tim" and "O'Connor" as the first name and last name into the first form, the HTML source code passed back to Page 2 is as follows (look at the lines in bold where the values of the previous form are passed back to the browser):

```
<HTML>
<HEAD>
<TITLE>Page 2</TITLE>
</HEAD>
<BODY>
<CENTER>
<H2>Page 2</H2>
<BR>
<BR>
Please enter your user name and password.
<BR>
<BR>
<FORM METHOD=POST ACTION=Page3Servlet>
<INPUT TYPE=HIDDEN NAME=firstName VALUE="Tim">
<INPUT TYPE=HIDDEN NAME=lastName VALUE="O'Connor">
```

```
<TABLE>
<TR>
<TD>User Name </TD>
<TD><INPUT TYPE=TEXT NAME=userName></TD>
</TR>
<TR>
<TD>Password </TD>
<TD><INPUT TYPE=PASSWORD NAME=password></TD>
</TR>
<TR>
<TD><INPUT TYPE=RESET></TD>
<TD><INPUT TYPE=SUBMIT VALUE=Submit></TD>
</TR>
</TABLE>
</FORM>
</CENTER>
</BODY>
</HTML>
```

Finally, Listing 5.11 presents the Page3Servlet that retrieves all the values from the second form.

Listing 5.11 Page3Servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.brainysoftware.java.StringUtil;
public class Page3Servlet extends HttpServlet {
  String page1Url = "Page1Servlet";
  String firstName;
  String lastName;
  String userName;
  String password;

  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.sendRedirect(page1Url);
  }

  /**Process the HTTP Post request*/
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException,IOException {
    firstName = request.getParameter("firstName");
    lastName = request.getParameter("lastName");
    userName = request.getParameter("userName");
    password = request.getParameter("password");
    if (firstName==null || lastName==null ||
      userName==null || password==null)
      response.sendRedirect(page1Url);
    // display all the values from the previous forms
    displayValues(response);
  }

  void displayValues(HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Page 3</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<CENTER>");
    out.println("<H2>Page 3 (Finish)</H2>");
    out.println("<BR>");
    out.println("<BR>");
    out.println("Here are the values you have entered.");
    out.println("<BR>");
```

```
    out.println("<BR>");
    out.println("<TABLE>");
    out.println("<TR>");
    out.println("<TD>First Name:  </TD>");
    out.println("<TD>" + StringUtil.encodeHtmlTag(firstName) + "</TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>Last Name:  </TD>");
    out.println("<TD>" + StringUtil.encodeHtmlTag(lastName) + "</TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>User Name:  </TD>");
    out.println("<TD>" + StringUtil.encodeHtmlTag(userName) + "</TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>Password:  </TD>");
    out.println("<TD>" + StringUtil.encodeHtmlTag(password) + "</TD>");
    out.println("</TR>");
    out.println("</TABLE>");
    out.println("</CENTER>");
    out.println("</BODY>");
    out.println("</HTML>");
  }
}
```

Multiple Forms in One Servlet

The previous example demonstrated how you could retain values in hidden fields in three servlets. Using more than one servlet is probably the last thing you want to do for a simple application, considering the maintenance involved for each servlet. This example shows the same application using only one servlet.

The problem with using one servlet is that each form will submit to the same servlet and the same doPost method will be invoked. How do you know which form to display next? The solution is simply done by incorporating a hidden field called page with a value of the form number. In the first form, the page field will have the value of 1, and in the second form this field will have the value of 2.

When the servlet is first called, the doGet method is invoked. What it does is very predictable: send the first form to the browser using the sendPage1 method:

```
sendPage1(response);
```

When the first form is submitted, it will invoke the doPost method because the form uses the POST method. The doPost method will retrieve the value of the parameter called page using the getParameter method. It should always find a value. However, if for some reason it does not, the method simply sends the first form and returns, as you see here:

```
String page = request.getParameter("page");
if (page==null) {
  sendPage1(response);
  return;
}
```

If a value for page is found, it could be 1 or 2. If 1 is returned, the previous request is from the first page; therefore Page 2 should be sent. The request that submits the first form must be accompanied by the parameters firstName and lastName, however. In other words, both the getParameter("firstName") and getParameter("lastName") must not return null. The values themselves could be blank strings, as is the case if the user does not type anything in the text boxes. However, a valid request from the first page must carry these parameters:

```
if (page.equals("1")) {
  if (firstName==null || lastName==null)
    sendPage1(response);
  else
    sendPage2(response);
}
```

If either firstName or lastName is not found, the sendPage1 is called again because the request is not valid.

If the first page is okay, the doPost method calls the sendPage2 method, which sends the second form as well as the previous values from the first form.

If the value for page is 2, the previous page must come from the second page and four parameters must be present in the request: firstName, lastName, userName, and password. Missing one of the values is a sufficient reason to resend the first page, as shown here:

```
else if (page.equals("2")) {
```

```
  if (firstName==null || lastName==null ||
    userName==null || password==null)
    sendPage1(response);
  else
    displayValues(response);
}
```

If the value for page is 2 and all the other values are found, the displayValues method is called and it displays all the four values from the first and second forms.

The complete code is given in Listing 5.12.

Listing 5.12 MultipleFormsServlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.brainysoftware.java.StringUtil;
public class MultipleFormsServlet extends HttpServlet {
  String firstName;
  String lastName;
  String userName;
  String password;

  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    sendPage1(response);
  }

  /**Process the HTTP Post request*/
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException,IOException {
    String page = request.getParameter("page");
    firstName = request.getParameter("firstName");
    lastName = request.getParameter("lastName");
    userName = request.getParameter("userName");
    password = request.getParameter("password");
    if (page==null) {
      sendPage1(response);
      return;
    }
    if (page.equals("1")) {
      if (firstName==null || lastName==null)
        sendPage1(response);
      else
        sendPage2(response);
      }
      else if (page.equals("2")) {
        if (firstName==null || lastName==null ||
          userName==null || password==null)
          sendPage1(response);
        else
          displayValues(response);
      }
    }

    void sendPage1(HttpServletResponse response)
      throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HTML>");
      out.println("<HEAD>");
      out.println("<TITLE>Page 1</TITLE>");
      out.println("</HEAD>");
      out.println("<BODY>");
      out.println("<CENTER>");
      out.println("<H2>Page 1</H2>");
      out.println("<BR>");
      out.println("<BR>");
```

```
    out.println("Please enter your first first name and last name.");
    out.println("<BR>");
    out.println("<BR>");
    out.println("<FORM METHOD=POST>");
    out.println("<INPUT TYPE=HIDDEN NAME=page VALUE=1>");
    out.println("<TABLE>");
    out.println("<TR>");
    out.println("<TD>First Name </TD>");
    out.println("<TD><INPUT TYPE=TEXT NAME=firstName></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>Last Name </TD>");
    out.println("<TD><INPUT TYPE=TEXT NAME=lastName></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD><INPUT TYPE=RESET></TD>");
    out.println("<TD><INPUT TYPE=SUBMIT VALUE=Submit></TD>");
    out.println("</TR>");
    out.println("</TABLE>");
    out.println("</FORM>");
    out.println("</CENTER>");
    out.println("</BODY>");
    out.println("</HTML>");
  }

  void sendPage2(HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Page 2</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<CENTER>");
    out.println("<H2>Page 2</H2>");
    out.println("<BR>");
    out.println("<BR>");
    out.println("Please enter your user name and password.");
    out.println("<BR>");
    out.println("<BR>");
    out.println("<FORM METHOD=POST>");
    out.println("<INPUT TYPE=HIDDEN NAME=page VALUE=2>");
    out.println("<INPUT TYPE=HIDDEN NAME=firstName VALUE=\"" +
      StringUtil.encodeHtmlTag(firstName) + "\"></TD>");
    out.println("<INPUT TYPE=HIDDEN NAME=lastName VALUE=\"" +
      StringUtil.encodeHtmlTag(lastName) + "\"></TD>");
    out.println("<TABLE>");
    out.println("<TR>");
    out.println("<TD>User Name </TD>");
    out.println("<TD><INPUT TYPE=TEXT NAME=userName></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>Password </TD>");
    out.println("<TD><INPUT TYPE=PASSWORD NAME=password></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD><INPUT TYPE=RESET></TD>");
    out.println("<TD><INPUT TYPE=SUBMIT VALUE=Submit></TD>");
    out.println("</TR>");
    out.println("</TABLE>");
    out.println("</FORM>");
    out.println("</CENTER>");
    out.println("</BODY>");
    out.println("</HTML>");
  }

  void displayValues(HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
```

```
      out.println("<HTML>");
      out.println("<HEAD>");
      out.println("<TITLE>Page 3</TITLE>");
      out.println("</HEAD>");
      out.println("<BODY>");
      out.println("<CENTER>");
      out.println("<H2>Page 3 (Finish)</H2>");
      out.println("<BR>");
      out.println("<BR>");
      out.println("Here are the values you have entered.");
      out.println("<BR>");
      out.println("<BR>");
      out.println("<TABLE>");
      out.println("<TR>");
      out.println("<TD>First Name:  </TD>");
      out.println("<TD>" + StringUtil.encodeHtmlTag(firstName) + "</TD>");
      out.println("</TR>");
      out.println("<TR>");
      out.println("<TD>Last Name:  </TD>");
      out.println("<TD>" + StringUtil.encodeHtmlTag(lastName) + "</TD>");
      out.println("</TR>");
      out.println("<TR>");
      out.println("<TD>User Name:  </TD>");
      out.println("<TD>" + StringUtil.encodeHtmlTag(userName) + "</TD>");
      out.println("</TR>");
      out.println("<TR>");
      out.println("<TD>Password:  </TD>");
      out.println("<TD>" + StringUtil.encodeHtmlTag(password) + "</TD>");
      out.println("</TR>");
      out.println("</TABLE>");
      out.println("</CENTER>");
      out.println("</BODY>");
      out.println("</HTML>");
    }
  }
```

Cookies

The third technique that you can use to manage user sessions is by using cookies. A *cookie* is a small piece of information that is passed back and forth in the HTTP request and response. Even though a cookie can be created on the client side using some scripting language such as JavaScript, it is usually created by a server resource, such as a servlet. The cookie sent by a servlet to the client will be passed back to the server when the client requests another page from the same application.

Cookies were first specified by Netscape (see http://home.netscape.com/newsref/std/cookie_spec.html) and are now part of the Internet standard as specified in RFC 2109: The HTTP State Management Mechanism. Cookies are transferred to and from the client in the HTTP headers.

In servlet programming, a cookie is represented by the Cookie class in the javax.servlet.http package. You can create a cookie by calling the Cookie class constructor and passing two String objects: the name and value of the cookie. For instance, the following code creates a cookie object called c1. The cookie has the name "myCookie" and a value of "secret":

```
Cookie c1 = new Cookie("myCookie", "secret");
```

You then can add the cookie to the HTTP response using the addCookie method of the HttpServletResponse interface:

```
response.addCookie(c1);
```

Note that because cookies are carried in the request and response headers, you must not add a cookie after an output has been written to the HttpServletResponse object. Otherwise, an exception will be thrown.

The following example shows how you can create two cookies called userName and password and illustrates how those cookies are transferred back to the server. The servlet is called CookieServlet, and its code is given in Listing 5.13.

When it is first invoked, the doGet method of the servlet is called. The method creates two cookies and adds both to the HttpServletResponse object, as follows:

```
Cookie c1 = new Cookie("userName", "Helen");
Cookie c2 = new Cookie("password", "Keppler");
response.addCookie(c1);
response.addCookie(c2);
```

Next, the doGet method sends an HTML form that the user can click to send another request to the servlet:

```
response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Cookie Test</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("Please click the button to see the cookies sent to you.");
    out.println("<BR>");
    out.println("<FORM METHOD=POST>");
    out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
    out.println("</FORM>");
    out.println("</BODY>");
    out.println("</HTML>");
```

The form does not have any element other than a submit button. When the form is submitted, the doPost method is invoked. The doPost method does two things: It iterates all the headers in the request to show how the cookies are conveyed back to the server, and it retrieves the cookies and displays their values.

To display all the headers in the HttpServletRequest method, it first retrieves an Enumeration object containing all the header names. The method then iterates the Enumeration object to get the next header name and passes the header name to the getHeader method to display the value of that header, as you see here:

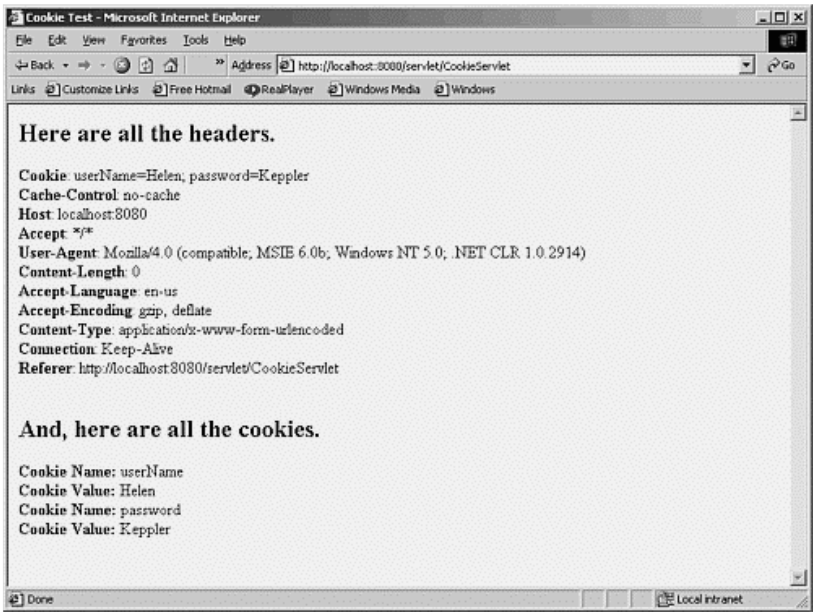
```
Enumeration enum = request.getHeaderNames();
while (enum.hasMoreElements()) {
  String header = (String) enum.nextElement();
  out.print("<B>" + header + "</B>: ");
  out.print(request.getHeader(header) + "<BR>");
}
```

To retrieve cookies, you use the getCookies method of the HttpServletRequest interface. This method returns a Cookie array containing all cookies in the request. It is your responsibility to loop through the array to get the cookie you want, as follows:

```
Cookie[] cookies = request.getCookies();
int length = cookies.length;
for (int i=0; i<length; i++) {
  Cookie cookie = cookies[i];
  out.println("<B>Cookie Name:</B> " +
    cookie.getName() + "<BR>");
  out.println("<B>Cookie Value:</B> " +
    cookie.getValue() + "<BR>");
}
```

The headers and cookies are displayed in Figure 5.7.

Figure 5.7. The headers containing cookies and the cookies' values.



Listing 5.13 Sending and Receiving Cookies

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

public class CookieServlet extends HttpServlet {

  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    Cookie c1 = new Cookie("userName", "Helen");
    Cookie c2 = new Cookie("password", "Keppler");
    response.addCookie(c1);
    response.addCookie(c2);

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Cookie Test</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("Please click the button to see the cookies sent to you.");
    out.println("<BR>");
    out.println("<FORM METHOD=POST>");
    out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
    out.println("</FORM>");
    out.println("</BODY>");
    out.println("</HTML>");
  }
  /**Process the HTTP Post request*/
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException,IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Cookie Test</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<H2>Here are all the headers.</H2>");

    Enumeration enum = request.getHeaderNames();
    while (enum.hasMoreElements()) {
      String header = (String) enum.nextElement();
      out.print("<B>" + header + "</B>: ");
      out.print(request.getHeader(header) + "<BR>");
    }

    out.println("<BR><BR><H2>And, here are all the cookies.</H2>");
    Cookie[] cookies = request.getCookies();
    int length = cookies.length;
    for (int i=0; i<length; i++) {
      Cookie cookie = cookies[i];
      out.println("<B>Cookie Name:</B> " + cookie.getName() + "<BR>");
      out.println("<B>Cookie Value:</B> " + cookie.getValue() + "<BR>");
    }
     out.println("</BODY>");
     out.println("</HTML>");
   }
}
```

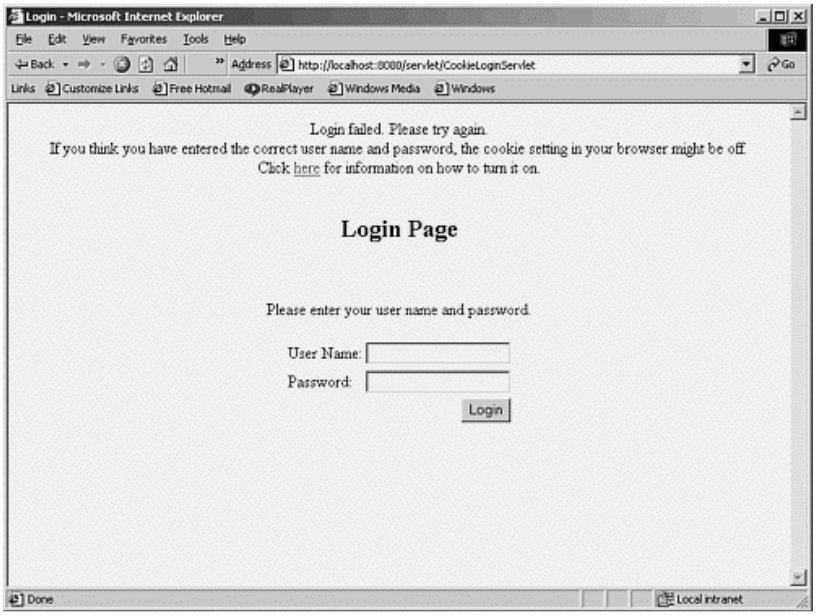
Another example of a servlet that uses cookies is a Login servlet that utilizes cookies to carry the user name and password information. The use of cookies is more appropriate than both URL rewriting and hidden values. First, unlike URL rewriting, the values of the cookies are not directly visible (you don't want this sensitive information to be seen by anyone). Second, you don't need to use any form, which is the requirement of using hidden fields.

Using cookes has a disadvantage, however: The user can choose not to accept them. Even though browsers leave the factories with the cookie setting on, any user can (accidentally) change this setting. The normal practice is therefore to use cookies with warnings to the user if the

application does not work as expected. The warning could be a simple message telling the user to activate his cookie setting, or it could be a hyperlink to a page that thoroughly describes how to set the cookie setting in various browsers.

So, here it is: the CookieLoginServlet that modifies the previous LoginServlet in Chapter 4. The complete code is given in Listing 5.14. If the cookie setting is not on, the servlet will send a message. The servlet also will send a message if the authentication fails (see Figure 5.8).

Figure 5.8. CookieLoginServlet.bmp.



An important part of the listing that deserves an explanation is the part that redirects the user to another resource when the login is successful. First, you need to create two cookies called userName and password and add them to the HttpServletResponse object. The cookies will always go back to the server when the user gets redirected to another resource. This resource can then retrieve the cookies and do the authentication again against the same database. This way, the user does not have to log in more than once.

Next, you need the code that redirects the user. Normally, to redirect a user, you would use the sendRedirect method. When you need to send cookies at the same time, however, redirecting using the sendRedirect method will not make the cookies get passed back to the server. As an alternative, you use a META tag of the following syntax:

```
<META HTTP-EQUIV=Refresh CONTENT=x;URL=ContentServlet>
```

This META tag will make the browser request another resource as indicated in the URL part. x indicates the number of seconds the browser will wait before the redirection occurs.

The code that does these two things is given here:

```
    if (login(userName, password)) {
      //send cookie to the browser
      Cookie c1 = new Cookie("userName", userName);
      Cookie c2 = new Cookie("password", password);
      response.addCookie(c1);
      response.addCookie(c2);
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      //response.sendRedirect does not work here.
      // use a Meta tag to redirect to ContentServlet
      out.println(
"<META HTTP-EQUIV=Refresh CONTENT=0;URL=ContentServlet>");
    }
```

Listing 5.14 CookieLoginServlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import com.brainysoftware.java.StringUtil;

public class CookieLoginServlet extends HttpServlet {
```

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
  sendLoginForm(response, false);
}

public void doPost(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {

  String userName = request.getParameter("userName");
  String password = request.getParameter("password");
  if (login(userName, password)) {
    //send cookie to the browser
    Cookie c1 = new Cookie("userName", userName);
    Cookie c2 = new Cookie("password", password);
    response.addCookie(c1);
    response.addCookie(c2);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    //response.sendRedirect does not work here.
    // use a Meta tag to redirect to ContentServlet
    out.println("<META HTTP-EQUIV=Refresh CONTENT=0;URL=ContentServlet>");
  }
  else {
    sendLoginForm(response, true);
  }
}

private void sendLoginForm(HttpServletResponse response, boolean withErrorMessage)
  throws ServletException, IOException {
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  out.println("<HTML>");
  out.println("<HEAD>");
  out.println("<TITLE>Login</TITLE>");
  out.println("</HEAD>");
  out.println("<BODY>");
  out.println("<CENTER>");

  if (withErrorMessage) {
    out.println("Login failed. Please try again.<BR>");
    out.println("If you think you have entered the correct user name" +
      " and password, the cookie setting in your browser might be off." +
      "<BR>Click <A HREF=InfoPage.html>here</A> for information" +
      " on how to turn it on.<BR>");
  }
  out.println("<BR>");
  out.println("<BR><H2>Login Page</H2>");
  out.println("<BR>");
  out.println("<BR>Please enter your user name and password.");
  out.println("<BR>");
  out.println("<BR><FORM METHOD=POST>");
  out.println("<TABLE>");
  out.println("<TR>");
  out.println("<TD>User Name:</TD>");
  out.println("<TD><INPUT TYPE=TEXT NAME=userName></TD>");
  out.println("</TR>");
  out.println("<TR>");
  out.println("<TD>Password:</TD>");
  out.println("<TD><INPUT TYPE=PASSWORD NAME=password></TD>");
  out.println("</TR>");
  out.println("<TR>");
  out.println("<TD ALIGN=RIGHT COLSPAN=2>");
  out.println("<INPUT TYPE=SUBMIT VALUE=Login></TD>");
  out.println("</TR>");
  out.println("</TABLE>");
  out.println("</FORM>");
  out.println("</CENTER>");
  out.println("</BODY>");
  out.println("</HTML>");
}
```

```
  public static boolean login(String userName, String password) {
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con = DriverManager.getConnection("jdbc:odbc:JavaWeb");
      Statement s = con.createStatement();
      String sql = "SELECT UserName FROM Users" +
        " WHERE UserName='" + StringUtil.fixSqlFieldValue(userName) + "'" +
        " AND Password='" + StringUtil.fixSqlFieldValue(password) + "'";

      ResultSet rs = s.executeQuery(sql);

      if (rs.next()) {
        rs.close();
        s.close();
        con.close();
        return true;
      }
      rs.close();
      s.close();
      con.close();
    }
    catch (ClassNotFoundException e) {
      System.out.println(e.toString());
    }
    catch (SQLException e) {
      System.out.println(e.toString());
    }
    catch (Exception e) {
      System.out.println(e.toString());
    }
    return false;
  }
}
```

The second resource (such as another servlet) has to check the presence of the two cookies before displaying its supposedly important content. A servlet called ContentServlet is created to demonstrate this. The CookieLoginServlet will redirect the user to this servlet upon a successful login. The ContentServlet is given in Listing 5.15.

Listing 5.15 ContentServlet

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

public class ContentServlet extends HttpServlet {

  public String loginUrl = "CookieLoginServlet";

  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {
    Cookie[] cookies = request.getCookies();
    int length = cookies.length;
    String userName = null;
    String password = null;

    for (int i=0; i<length; i++) {
      Cookie cookie = cookies[i];
      if (cookie.getName().equals("userName"))
        userName = cookie.getValue();
      else if (cookie.getName().equals("password"))
        password = cookie.getValue();
    }

    if (userName==null || password==null || !CookieLoginServlet.login(userName,
➥password))
      response.sendRedirect(loginUrl);

    // This is an authorized user, okay to display content
```

```
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Welcome</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("Welcome.");
    out.println("</BODY>");
    out.println("</HTML>");
  }

  /**Process the HTTP Post request*/
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {
    doGet(request, response);
  }
}
```

After studying the previous example, you might wonder why you need to create two cookies that hold the value for the user name and password, respectively. Can't you just use a cookie that contains the value of true to indicate a previous successful login? The ContentServlet can just find this cookie and doesn't have to authenticate the user name and password again, hence saving a database manipulation.

This might sound better; however, a clever user can create a cookie at the client side. Knowing that a successful cookie carries a special flag, the user can create the cookie and get authenticated without having to know a valid user name and password.

Anticipating a Failed Redirection

When you need to do redirection while using cookies for managing user sessions—even when everything looks perfect—a redirection might fail. In the code that is supposed to redirect the user, you should provide a link that the user can manually click should the automatic redirection fail.

Note

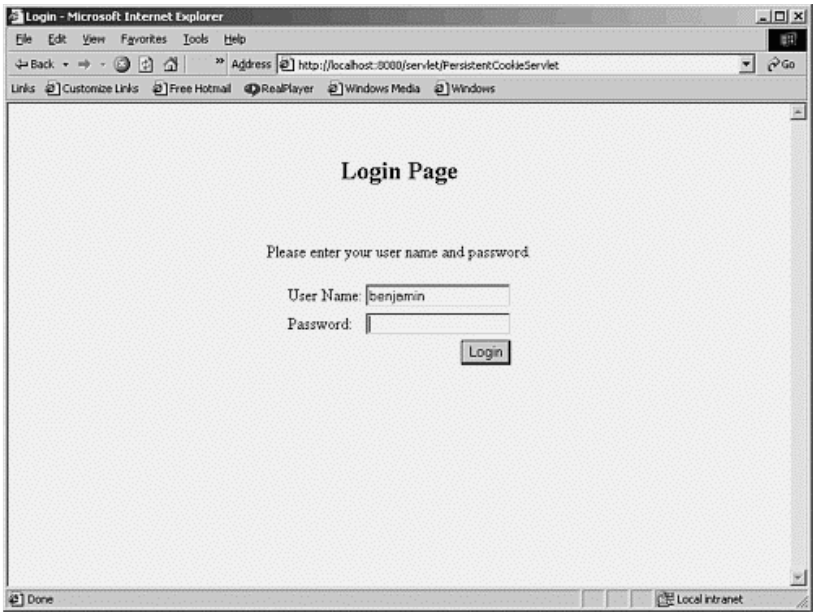
If you are testing a cookie and find that the code does not work as expected, close your browser to delete the cookie.

Persisting Cookies

The cookies you created in the previous example last as long as the browser is open. When the browser is closed, the cookies are deleted. You can choose to persist cookies so that they last longer. The javax.servlet.http.Cookie class has the setMaxAge method that sets the maximum age of the cookie in seconds.

In the next example, you will create a Login servlet that uses a cookie that persists for 10,000 seconds. If the user closes the browser but comes back within 10,000 seconds, he or she does not have to enter a user name again. Figure 5.9 shows the Login page in which the user name has been supplied by the server.

Figure 5.9. Persistent Cookie servlet.



The servlet is called PersistentCookieServlet and is given in Listing 5.16.

Listing 5.16 PersistentCookieServlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import com.brainysoftware.java.StringUtil;

public class PersistentCookieServlet extends HttpServlet {
  String persistedUserName;

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    Cookie[] cookies = request.getCookies();
    int length = cookies.length;
    for (int i=0; i<length; i++) {
      Cookie cookie = cookies[i];
      if (cookie.getName().equals("userName"))
        persistedUserName = cookie.getValue();
    }
    sendLoginForm(response, false);
  }

  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String userName = request.getParameter("userName");
    String password = request.getParameter("password");
    if (login(userName, password)) {
      //send cookie to the browser
      Cookie c1 = new Cookie("userName", userName);
      Cookie c2 = new Cookie("password", password);
      c1.setMaxAge(10000);

      response.addCookie(c1);
      response.addCookie(c2);
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      //response.sendRedirect does not work here.
      // use a Meta tag to redirect to ContentServlet
      out.println("<META HTTP-EQUIV=Refresh CONTENT=0;URL=ContentServlet>");
    }
    else {
      sendLoginForm(response, true);
    }
  }

  private void sendLoginForm(HttpServletResponse response, boolean withErrorMessage)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Login</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<CENTER>");

    if (withErrorMessage) {
      out.println("Login failed. Please try again.<BR>");
      out.println("If you think you have entered the correct user name" +
        " and password, the cookie setting in your browser might be off." +
        "<BR>Click <A HREF=InfoPage.html>here</A> for information" +
        " on how to turn it on.<BR>");
    }
    out.println("<BR>");
    out.println("<BR><H2>Login Page</H2>");
    out.println("<BR>");
```

```
    out.println("<BR>Please enter your user name and password.");
    out.println("<BR>");
    out.println("<BR><FORM METHOD=POST>");
    out.println("<TABLE>");
    out.println("<TR>");
    out.println("<TD>User Name:</TD>");
    out.print("<TD><INPUT TYPE=TEXT NAME=userName");

    if (persistedUserName!=null)
      out.print(" VALUE=\"" + persistedUserName + "\"");
    out.print("></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>Password:</TD>");
    out.println("<TD><INPUT TYPE=PASSWORD NAME=password></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD ALIGN=RIGHT COLSPAN=2>");
    out.println("<INPUT TYPE=SUBMIT VALUE=Login></TD>");
    out.println("</TR>");
    out.println("</TABLE>");
    out.println("</FORM>");
    out.println("</CENTER>");
    out.println("</BODY>");
    out.println("</HTML>");
  }

  public static boolean login(String userName, String password) {
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con = DriverManager.getConnection("jdbc:odbc:JavaWeb");
      Statement s = con.createStatement();
      String sql = "SELECT UserName FROM Users" +
        " WHERE UserName='" + StringUtil.fixSqlFieldValue(userName) + "'" +
        " AND Password='" + StringUtil.fixSqlFieldValue(password) + "'";
      ResultSet rs = s.executeQuery(sql);

      if (rs.next()) {
        rs.close();
        s.close();
        con.close();
        return true;
      }
      rs.close();
      s.close();
      con.close();
    }
    catch (ClassNotFoundException e) {
      System.out.println(e.toString());
    }
    catch (SQLException e) {
      System.out.println(e.toString());
    }
    catch (Exception e) {
      System.out.println(e.toString());
    }
    return false;
  }
}
```

When a user requests the PersistentCookieServlet, either the first time or a subsequent time, the doGet method is invoked. What this method does is browse through the Cookie collection obtained from the getCookies() method of the javax.servlet.http.HttpServletResponse interface. For each Cookie, the code tests whether the cookie name is "userName". If it is, the cookie value is assigned to the persistentUserName field as follows:

```
Cookie[] cookies = request.getCookies();
int length = cookies.length;
for (int i=0; i<length; i++) {
  Cookie cookie = cookies[i];
  if (cookie.getName().equals("userName"))
    persistedUserName = cookie.getValue();
```

```
}
```

Then, the doGet method calls the sendLoginForm method to send the Login form that the user can use to log in:

```
sendLoginForm(response, false);
```

When the user submits the Login form, the doPost method in the PersistentCookieServlet is invoked. The doPost method first tries to retrieve the userName and password values entered by the user, as you see here:

```
String userName = request.getParameter("userName");
String password = request.getParameter("password");
```

It then sends the userName and password values to the login method. This method will return true if the userName and password are valid. Upon successful login, two cookies, c1 and c2, will be created. c1 is called userName, and c2 is named password:

```
if (login(userName, password)) {
  //send cookie to the browser
  Cookie c1 = new Cookie("userName", userName);
  Cookie c2 = new Cookie("password", password);
```

The c1 cookie is set to last for 10,000 seconds using the setMaxAge method, as follows:

```
c1.setMaxAge(10000);
```

Then, the doPost method sends both c1 and c2 to the browser and redirects the browser to another servlet, as the following code shows:

```
response.addCookie(c1);
response.addCookie(c2);
response.setContentType("text/html");
PrintWriter out = response.getWriter();
//response.sendRedirect does not work here.
// use a Meta tag to redirect to ContentServlet
out.println(
  "<META HTTP-EQUIV=Refresh CONTENT=0;URL=ContentServlet>");
```

If the user comes back within 10,000 seconds, the browser will send the userName cookie back to the server and the cookie will be found in the doGet method.

The login method called from the doPost method sets up a database connection and sends the following SQL statement:

```
SELECT UserName FROM Users
WHERE UserName='userName'
AND Password='password'
```

If the SQL statement execution returns a non-empty ResultSet object, a user account by that name and password is found and the login method returns true:

```
public static boolean login(String userName, String password) {
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con = DriverManager.getConnection("jdbc:odbc:JavaWeb");
      Statement s = con.createStatement();
      String sql = "SELECT UserName FROM Users" +
        " WHERE UserName='" + StringUtil.fixSqlFieldValue(userName) + "'" +
        " AND Password='" + StringUtil.fixSqlFieldValue(password) + "'";
      ResultSet rs = s.executeQuery(sql);

      if (rs.next()) {
        rs.close();
        s.close();
        con.close();
        return true;
      }
      rs.close();
      s.close();
      con.close();
    }
    catch (ClassNotFoundException e) {
      System.out.println(e.toString());
    }
```

```
    catch (SQLException e) {
      System.out.println(e.toString());
    }
    catch (Exception e) {
      System.out.println(e.toString());
    }
    return false;
  }
}
```

Checking Whether Cookie Setting Is On

All the cookie-related examples assume that the user browser's cookie setting is on. Even though browsers leave the factory with this setting on, the user can turn this off. One approach to solving this problem is to send a warning message if the application does not work as expected. The other option is to check this setting automatically. This option can be explained with an example of a servlet called CheckCookieServlet.

What the servlet does is simple enough. It sends a response that forces the browser to come back for the second time. With the first response, it sends a cookie. When the browser comes back for the second time, the servlet checks whether the request carries the cookie sent previously. If the cookie is there, it can be concluded that the browser setting for cookies is on. Otherwise, it could be that the user is using a very old browser that does not recognize cookies at all, or the cookie support for that browser is off.

The CheckCookieServlet is given in Listing 5.17.

Listing 5.17 CheckCookieServlet

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

public class CheckCookieServlet extends HttpServlet {
  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    if (request.getParameter("flag")==null) {
      // the first request
      Cookie cookie = new Cookie("browserSetting", "on");
      response.addCookie(cookie);
      String nextUrl = request.getRequestURI() + "?flag=1";
      out.println("<META HTTP-EQUIV=Refresh CONTENT=0;URL="+ nextUrl +">");
    }
    else {
      // the second request
      Cookie[] cookies = request.getCookies();
      if (cookies!=null)  {
      int length = cookies.length;
      boolean cookieFound = false;
      for (int i=0; i<length; i++) {
        Cookie cookie = cookies[i];
        if (cookie.getName().equals("browserSetting") &&
          cookie.getValue().equals("on")) {
          cookieFound = true;
          break;
        }
      }
      if (cookieFound) {
        out.println("Your browser's cookie setting is on.");
      }
      else {
        out.println("Your browser does not support cookies or" +
          " the cookie setting is off.");
      }
    }
  }
  /**Process the HTTP Post request*/
  public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    doGet(request, response);
  }
```

```
}
```

Session Objects

Of the four techniques for session management covered in this chapter, the Session object, represented by the javax.servlet.http.HttpSession interface, is the easiest to use and the most powerful. For each user, the servlet can create an HttpSession object that is associated with that user only and can only be accessed by that particular user. The HttpSession object acts like a Hashtable into which you can store any number of key/object pairs. The HttpSession object is accessible from other servlets in the same application. To retrieve an object previously stored, you need only to pass the key.

An HttpSession object uses a cookie or URL rewriting to send a token to the client. If cookies are used to convey session identifiers, the client browsers are required to accept cookies.

Unlike previous techniques, however, the server does not send any value. What it sends is simply a unique number called the *session identifier*. This session identifier is used to associate a user with a Session object in the server. Therefore, if there are 10 simultaneous users, 10 Session objects will be created in the server and each user can access only his or her own HttpSession object.

The way an HttpSession object is created for a user and retrieved in the next requests is illustrated in Figure 5.10.

Figure 5.10. How session tracking works with the HttpSession object.

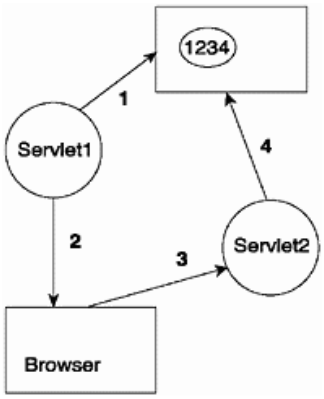


Figure 5.10 shows that there are four steps in session tracking using the HttpSession object:

1. An HttpSession object is created by a servlet called Servlet1. A session identifier is generated for this HttpSession object. In this example, the session identifier is 1234, but in reality, the servlet container will generate a longer random number that is guaranteed to be unique. The HttpSession object then is stored in the server and is associated with the generated session identifier. Also the programmer can store values immediately after creating an HttpSession.

2. In the response, the servlet sends the session identifier to the client browser.

3. When the client browser requests another resource in the same application, such as Servlet2, the session identifier is sent back to the server and passed to Servlet2 in the javax.servlet.http.HttpServletRequest object.

4. For Servlet2 to have access to the HttpSession object for this particular client, it uses the getSession method of the javax.servlet.http.HttpServletRequest interface. This method automatically retrieves the session identifier from the request and obtains the HttpSession object associated with the session identifier.

Note

What if the user never comes back after an HttpSession object is created? Then the servlet container waits for a certain period of time and removes that HttpSession object. One worry about using Session objects is scalability. In some servlet containers, Session objects are stored in memory, and as the number of users exceeds a certain limit, the server eventually runs out of memory.

One solution to this memory problem when using Session objects is to save Session objects to the database or disk. However, the Servlet 2.3 Specification does not clearly state how a servlet container should do this. If you are using Tomcat 4, however, your Session objects will be moved to secondary storage once the number of Session objects has exceeded some value.

The getSession method of the javax.servlet.http.HttpServletRequest interface has two overloads. They are as follows:

- HttpSession getSession()
- HttpSession getSession(boolean create)

The first overload returns the current session associated with this request, or if the request does not have a session identifier, it creates a new one.

The second overload returns the HttpSession object associated with this request if there is a valid session identifier in the request. If no valid session identifier is found in the request, whether a new HttpSession object is created depends on the create value. If the value is true, a new HttpSession object is created if no valid session identifier is found in the request. Otherwise, the getSession method will return null.

Now that you know how session management works using the HttpSession object, let's have a look at the javax.servlet.http.HttpSession interface in more detail.

The javax.servlet.http.HttpSession Interface

This interface has the following methods:

- getAttribute
- getAttributeNames
- getCreationTime
- getId
- getLastAccessedTime
- getMaxInactiveInterval
- getServletContext
- getSessionContext
- getValue
- getValueNames
- invalidate
- isNew
- putValue
- removeAttribute
- removeValue
- setAttribute
- setMaxInactiveInterval

Each of the methods is discussed below.

getAttribute

This method retrieves an attribute from the HttpSession object. The return value is an object of type Object; therefore you may have to downcast the attribute to its original type. To retrieve an attribute, you pass the name associated with the attribute. This method returns an IllegalStateException if it is called upon an invalidated HttpSession object.

The signature for this method is as follows:

```
public Object getAttribute(String name) throws IllegalStateException
```

getAttributeNames

The getAttributeNames method returns a java.util.Enumeration containing all attribute names in the HttpSession object. This method returns an IllegalStateException if it is called upon an invalidated HttpSession object.

The signature is as follows:

```
public java.util.Enumeration getAttributeNames() throws IllegalStateException
```

getCreationTime

The getCreationTime method returns the time that the HttpSession object was created, in milliseconds since January 1, 1970 00:00:00 GMT. This method returns an IllegalStateException if it is called upon an invalidated HttpSession object.

The signature for this method is as follows:

```
public long getCreationTime() throws IllegalStateException
```

getId

The getID method returns the session identifier. The signature for this method is as follows:

```
public String getId()
```

getLastAccessedTime

The getLastAccessedTime method returns the time the HttpSession object was last accessed by the client. The return value is the number of milliseconds lapsed since since January 1, 1970 00:00:00 GMT. The following is the method signature:

```
public long getLastAccessedTime()
```

getMaxInactiveInterval

The getMaxInactiveInterval method returns the number of seconds the HttpSession object will be retained by the servlet container after it is last accessed before being removed. The signature of this method is as follows:

```
public int getMaxInactiveInterval()
```

getServletContext

The getServletContext method returns the javax.servlet.ServletContext object the HttpSession object belongs to. The signature is as follows:

```
public javax.servlet.ServletContext getServletContext
```

getSessionContext

This method is deprecated.

getValue

This method is deprecated.

getValueNames

This method is deprecated.

invalidate

The invalidate method invalidates the HttpSession object and unbinds any object bound to it. This method throws an IllegalStateException if this method is called upon an already invalidated HttpSession object. The signature is as follows:

```
public void invalidate() throws IllegalStateException
```

isNew

The isNew method indicates whether the HttpSession object was created with this request and the client has not yet joined the session tracking. This method returns an IllegalStateException if it is called upon an invalidated HttpSession object.

Its signature is as follows:

```
public boolean isNew() throws IllegalStateException
```

putValue

This method is deprecated.

removeAttribute

The removeAttribute method removes an attribute bound to this HttpSession object. This method returns an IllegalStateException if it is called upon an invalidated HttpSession object.

Its signature is as follows:

```
public void removeAttribute(String name) throws IllegalStateException
```

removeValue

This method is deprecated.

setAttribute

The setAttribute method adds a name/attribute pair to the HttpSession object. This method returns an IllegalStateException if it is called upon an invalidated HttpSession object. The method has the following signature:

```
public void setAttribute(String name, Object attribute) throws
IllegalStateException
```

setMaxInactiveInterval

The setMaxInactiveInterval method sets the number of seconds from the time the HttpSession object is accessed the servlet container will wait before removing the HttpSession object. The signature is as follows:

```
public void setMaxInactiveInterval(int interval)
```

Passing a negative number to this method will make this HttpSession object never expire.

Using HttpSession Object

The following example modifies the previous Login page and uses HttpSession objects. You don't need to create two cookies for both the user name and password. Because only the server can store and retrieve values from a HttpSession object, you can create a key called loggedIn with the value of "true" if the user has successfully logged in before.

The code is given in Listing 5.18.

Listing 5.18 The SessionLoginServlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import com.brainysoftware.java.StringUtil;

public class SessionLoginServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    sendLoginForm(response, false);
  }

  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String userName = request.getParameter("userName");
    String password = request.getParameter("password");
    if (login(userName, password)) {
      //send cookie to the browser
      HttpSession session = request.getSession(true);
      session.setAttribute("loggedIn", new String("true"));
      response.sendRedirect("Content2Servlet");
    }
    else {
      sendLoginForm(response, true);
    }
  }

  private void sendLoginForm(HttpServletResponse response, boolean withErrorMessage)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
```

```
    out.println("<TITLE>Login</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<CENTER>");

    if (withErrorMessage) {
      out.println("Login failed. Please try again.<BR>");
      out.println("If you think you have entered the correct user name" +
        " and password, the cookie setting in your browser might be off." +
        "<BR>Click <A HREF=InfoPage.html>here</A> for information" +
        " on how to turn it on.<BR>");
    }
    out.println("<BR>");
    out.println("<BR><H2>Login Page</H2>");
    out.println("<BR>");
    out.println("<BR>Please enter your user name and password.");
    out.println("<BR>");
    out.println("<BR><FORM METHOD=POST>");
    out.println("<TABLE>");
    out.println("<TR>");
    out.println("<TD>User Name:</TD>");
    out.println("<TD><INPUT TYPE=TEXT NAME=userName></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>Password:</TD>");
    out.println("<TD><INPUT TYPE=PASSWORD NAME=password></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD ALIGN=RIGHT COLSPAN=2>");
    out.println("<INPUT TYPE=SUBMIT VALUE=Login></TD>");
    out.println("</TR>");
    out.println("</TABLE>");
    out.println("</FORM>");
    out.println("</CENTER>");
    out.println("</BODY>");
    out.println("</HTML>");
  }

  public static boolean login(String userName, String password) {
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con = DriverManager.getConnection("jdbc:odbc:JavaWeb");
      Statement s = con.createStatement();
      String sql = "SELECT UserName FROM Users" +
        " WHERE UserName='" + StringUtil.fixSqlFieldValue(userName)+ "'" +
        " AND Password='" + StringUtil.fixSqlFieldValue(password) + "'";

      ResultSet rs = s.executeQuery(sql);

      if (rs.next()) {
        rs.close();
        s.close();
        con.close();
        return true;
      }
      rs.close();
      s.close();
      con.close();
    }
    catch (ClassNotFoundException e) {
      System.out.println(e.toString());
    }
    catch (SQLException e) {
      System.out.println(e.toString());
    }
    catch (Exception e) {
      System.out.println(e.toString());
    }
    return false;
  }
}
```

```
Listing x: Content2Servlet
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class Content2Servlet extends HttpServlet {

  public String loginUrl = "SessionLoginServlet";

  /**Process the HTTP Get request*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException,IOException {
    HttpSession session = request.getSession();
    if (session==null)
      response.sendRedirect(loginUrl);
    else {
      String loggedIn = (String) session.getAttribute("loggedIn");
      if (!loggedIn.equals("true"))
        response.sendRedirect(loginUrl);
    }
    // This is an authorized user, okay to display content
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Welcome</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("Welcome.");
    out.println("</BODY>");
    out.println("</HTML>");
  }

  /**Process the HTTP Post request*/
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws
➥ServletException, IOException {
    doGet(request, response);
  }
}
```

Session Tracking with URL-Rewriting

As mentioned previously, by default a session identifier is sent to the client browser and back to the server using a cookie. This means that the client browser must have its cookie support enabled. Therefore, you need to perform browser cookie testing as explained in the previous section, "Checking Whether Cookie Setting Is On."

Alternatively, you can avoid the use of cookies by sending session identifiers in the URL using the URL-rewriting technique. Fortunately, the Servlet API provides an easy way to append session identifier to the URL—using the encodeURL method of the javax.servlet.http.HttpServletResponse interface.

To use the URL-rewriting technique, pass any URL referenced to in a servlet into the encodeURL method of the javax.servlet.http.HttpServletResponse interface. For example, the following code:

```
out.println("<a href=Servlet2>Click here</a>");
```

must be changed into the following:

```
out.println("<a href=" + response.encodeURL("Servlet2") +
  ">Click here</a>");
```

This will add the session identifier to the end of the string, resulting in a URL similar to the following:

```
http://localhost:8080/myApp/servlet/Servlet2;jsessionid=AB34898928342948 9234At
```

where the number after jsessionid= is the session identifier.

One disadvantage of using URL rewriting as opposed to cookies in sending your session identifiers is that URL rewriting does not survive static pages. For example, if the user has to browse through some HTML page during his or her session, the session identifier will be lost.

The not-so-practical solution is of course to convert all static pages into servlets or JSP pages.

Knowing Which Technique to Use

Having learned the four techniques for managing user sessions, you may be wondering which one you should choose to implement.

Clearly, using Session objects is the easiest and you should use this if your servlet container supports swapping Session objects from memory to secondary storage. If you are using Tomcat 4, this feature is available to you.

One concern when using the Session objects is whether cookie support is enabled in the user browser. If it is, you have two options:

- You can test the cookie support setting by using the technique described in the section "Checking Whether Cookie Setting Is On."
- You can use URL-rewriting.

Appending your session identifier to the URL is a good technique, even though this creates some additional work for the programmer. However, this relieves you of having to rely on cookies.

Using cookies is not as flexible as using Session objects. However, cookies are the way to go if you don't want your server to store any client-related information or if you want the client information to persist when the browser is closed.

Finally, hidden fields are probably the least-often-used technique. If you need to split a form into several smaller ones, however, using hidden fields is the cheapest and most efficient method. You don't need to consume server resources to temporarily store the values from the previous forms, and you don't need to rely on cookies. I would suggest hidden fields over Session objects, URL-rewriting, or cookies in this case.

Summary

In this chapter, you have learned that HTTP is a stateless protocol. You also have been shown the implications of this statelessness. Additionally you learned about four techniques you can use to manage user sessions. Each technique has been described, and examples have been built to demonstrate the technique.

 CONTENTS