SESSION MANAGEMENT
and
DATABASE CONNECTIVITY

Session Management

HTTP protocol does not remember information when client communicates with the server. HTTP treats each request as a new request.

But sometimes clients are doing some important work such as online shopping, online banking, etc. In that situation, it is necessary to know about the client and remember the client's request accordingly.

For maintaining each client's session, we can use Session Management.

Session: The server should recognize a series of requests from the same user which form a single working 'session'. For example, in net banking application, each client can be differentiated from another client by associating a unique identifier in request and response within a specific working session.

State: The server should be able to remember information related to the previous request and other business transaction that are made for that request. For example, in net banking application, state comprises client information such as account number and amount transaction made within the particular session.

Four Ways to Manage the Sessions

HttpSession object, Cookies, URL rewriting, and Hidden Fields.

Any dynamic website or web application uses the concept of sessions and Session object to store data for a particular user.

For example, when you visit any web application for the first time, you have entered login name and password.

This information might be stored in session variable and maintained by the servlet container during your visit to web application so that it can be accessed when needed.

When your session is started, the requesting browser is allotted a unique id for each of the clients for identifies the client.

This Session object is denoted by *javax.http.HttpSession* interface.

Cookies

Cookies are a small part of information like a name, a single value and optional attributes such as comment, path and domain qualifiers, a maximum age, and a version number.

A servlet sends this cookie information to a web browser. It is saved by the browser and later sent back to the server.

A cookie is indicated by the Cookie class in the *javax.servlet.http* package. You can create a cookie by calling Cookie class like the following:

Cookie c = new Cookie("userid", "socis");

You can send the cookie to the client browser by using addCookie() method of the *HttpServletResponse* interface like the following:

response.addCookie(c);

You can retrieve cookies from request using getCookie() of the *HttpServletRequest* interface like the following:

request.getCookie(c);

URL Writting

The Token (parameter) is embedded in each URL. When client submits request using such URLs, the token is retransmitted to the server.

In each dynamically generated page, server embeds an extra query parameter or extra path information.

If a browser does not support cookies then in that case URL rewriting technique is the best alternative for session management.

Using URL Writing technique, you can send parameter name/value pairs like the following:

http://myserver.com?name=xyz&age=20

When you click on URL, the parameter name/value pair will transfer to the servlet. The servlet will fetch this parameter by using getParameter() method of *HttpServletRequest* interface from the requested URL and use it for session management.

Hidden Fields

When client submits a form, additional fields will also be sent in the request in the form of hidden fields to keep track of the session.

This method gives you the advantage to use this without depending on the browser whether the cookie is disabled or not.

It has a disadvantage also as it is not secure because anyone can view the hidden form field value from the HTML file and use it to hack the session.

Another disadvantage of using this method is that it needs extra form submission on each page.

You can create a unique hidden filed in the HTML form to keep track of the session like the following format:

<input type="hidden" name="userid" value="socis">

You can get this hidden field in Java Servlet using the getParameter() method of *HttpServletRequest* interface.

Servlet Collaboration

Servlet collaboration means sharing information among the servlets.

Sometimes, a situation arises in program coding when you may require passing the request from one servlet to another.

Also, you may want to include the content from HTML, JSP or Servlet into your servlet.

Java provides Servlet-API with two interfaces to accomplish Servlet Collaboration :

javax.servlet.RequestDispatcher and
javax.servlet.http.HttpServletResponse.

forward() and include()

RequestDispatcher interface of javax.servlet package has two methods :
forward () and *include()*
for dispatching requests to/from web resources.

forward() forwards a request from a servlet to another web resource like servlet, JSP file or HTML file on the server. The signature is as follows:

public void forward(ServletRequest request, ServletResponse response)
throws ServletException, java.io.IOException

include() includes the content of another servlet, JSP page, HTML file in the servlet response. The signature is as follows:

public void include(ServletRequest request, ServletResponse response)
throws ServletException, java.io.IOException

To use forward() or include() method, a RequestDispatcher object has to be obtained::

RequestDispatcher rd = request.getRequestDispatcher(String resource);

Servlet Context

Servlet context is the environment where the servlet runs.

The ServletContext's object is created by the web container at the time of deploying the project.

The ServletContext object is used to get configuration information from deployment descriptor file (web.xml).

ServletContext app = getServletContext();

You can set attributes of ServletContext object by using the *setAttribute()* method.

By using the *getAttribute()* method, other servlets can get the attribute from the ServletContext object.

Database Connectivity

Database access is one of the important features of Web application.

The JDBC technology of Java provides a standard library for accessing a wide range of relational databases like MS-Access, Oracle and Microsoft SQL Server.

Using JDBC API, you can access a wide variety of different SQL DBs.

The core functionality of JDBC is found in *java.sql* package.

Let's consider a table named as *Student* created in Microsoft SQL Server database with Roll No, Name and Program name and use it for storing/retrieving data into/from a Microsoft SQL Server using type-4 JDBC driver.

Insert data in Database

```
Connection con = null;
Statement stmt = null;

String url=
"jdbc:microsoft:sqlserver://127.0.0.1:1433;user=sa;password=poonam;Datab
aseName=SOCIS";

Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
con = DriverManager.getConnection(url);

if (con != null)
{
stmt = con.createStatement();
String rsql ="insert into student
values("+rollNo+",'"+StuName+"','"+prgName+"'"+")";

stmt.executeUpdate(rsql);
out.println("Your data is successfully stored in database");
}
```

Retrieving Data

```
if (con != null)
{
stmt = con.createStatement();
String rsql ="select * from Student";

rs = stmt.executeQuery(rsql);
out.println("<table border=1><tr><td>Roll Number</td><td>Student
Name</td><td>Programme</td></tr>");

while( rs.next() )
{
out.println("<tr><td>" + rs.getInt("RollNo") + "</td>");
out.println("<td>" + rs.getString("Student_Name") + "</td>");
out.println("<td>" + rs.getString("Programme") + "</td></tr>");
}

out.println("</table>");
```

Thank You….