Session Management and Database Connectivity

Session Management

- HTTP is a stateless protocol. It means that the HTTP protocol does not remember information when client or user communicates with the server.
- Whenever we send a request to the server through HTTP, the HTTP treats each request as a new request.
- So, there needs to be a technique that remembers information across multiple requests, This technique is called session management.

Session and State

- **Session**: A session refers to the period of time a user interacts with a web application. It represents a series of requests and responses between a user and the server that are considered part of a single interaction. Sessions are used to maintain user-specific data across multiple HTTP requests, which are stateless by nature.
- It maintains user context across requests.
- *Use cases for Session*: Authentication and Authorization, Shopping Cart Contents, User Preferences and settings during the session, Temporary storage of form data.
- **State**:State refers to the current condition or status of an application at any given time. It can be broken down into different types, such as *client-side state*, *server-side state*, and *application state*.
- It maintains the overall condition of the application
- *Use cases for State*: Maintaining the status of user interface elements (e.g., whether a menu is expanded or collapsed), Tracking the progress of a multi-step process (e.g., a checkout process),Holding data that needs to persist across different sessions (e.g., user profile information) etc.
- **Session Example**: When a user logs in, a session is created to keep track of their authentication status.As the user adds items to their cart, this information is stored in the session. When the user logs out or the session

expires, the session data (including the cart contents) is cleared.

Session Management

- Session management refers to the techniques and practices used to manage user sessions in web applications. A session is a period during which a user interacts with a web application, and session management is crucial for maintaining state and context across multiple requests.
- It does not change the nature of HTTP protocol i.e., stateless feature provides a way to remember the client information.

Four ways to manage sessions:

1. HttpSession Object
2. Cookies
3. URL rewriting
4. Hidden Fields

1. HttpSession Object

- A HttpSession object in Java is used to track session data between client and server during a user's interaction with a web application. Here's an example of how to use HttpSession in a servlet.

```
servletSession.java
```

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

public class ServletSession extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOExce
        // Set the content type of the response
        res.setContentType("text/html");

        // Get the PrintWriter to write the response
        PrintWriter out = res.getWriter();

        // Get the current session or create one if it doesn't exist
        HttpSession session = req.getSession(true);

        // Retrieve the visit count from the session attribute
        Integer visitCount = (Integer) session.getAttribute("visitCount");

        // If this is the first visit, initialize the count to 1
        if (visitCount == null) {
            visitCount = 1;
        } else {
            // Otherwise, increment the visit count
            visitCount = visitCount + 1;
        }

        // Store the updated visit count back into the session
        session.setAttribute("visitCount", visitCount);
        // Output session details
        out.println("Session Details: <br/>");
        out.println("You have visited this page: " + visitCount + (visitCount == 1 ? " time" : " times
        out.println("Session ID: " + session.getId() + "<br/>");
        out.println("New Session: " + session.isNew() + "<br/>");
        out.println("Creation Time: " + new Date(session.getCreationTime()) + "<br/>");
        out.println("Last Access Time: " + new Date(session.getLastAccessedTime()) + "<br/>");
    }
}
```

servletSession.html

```
<h1>Click here to go <a href="servletSession" >Session Servlet Program</a></h1>
```

2. Cookies

- Cookies are small pieces of text sent to your browser by a website you visit. They help that website remember information about your visit, which can both make it easier to visit the site again and make the site more useful to you.
- There are two types of cookies
  - **Non Persistent Cookies**: Effective for single session only.
  - **Persistent Cookie**:Effective for multiple sessions, only removed when user log outs.
- Creating a cookie - Using cookie class in `javax.servlet.http` package.
  - `Cookie c = new Cookie(“userid”, “socis”);`
- Send the cookie to browser
  - `response.addCookie(c);`
- Retrieve cookies
  - `request.getCookie(c);`

cookieForm.html

```
<form action="../servlet/CreateCookieServlet" method="post">
    Name:<input type="text" name="uname"/><br/>
    <input type="submit" value="Submit"/>
</form>
```

CreateCookieServlet.java

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

public class CreateCookieServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response) {
        try {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            // Retrieve the name parameter from the form submission
            String name = request.getParameter("uname");

            out.print("Welcome " + name);
            out.print(", Submit your data for GetCookieServlet");

            // Create a cookie object with name "uname" and value as the submitted name
            Cookie c = new Cookie("uname", name);

            // Add the cookie to the response
            response.addCookie(c);

            // Create a form to submit data to GetCookieServlet
            out.println("<form action='../servlet/GetCookieServlet' method='post'>");
            out.println("<input type='submit' value='Submit data'>");
            out.println("</form>");
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

GetCookieServlet.java

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

public class GetCookieServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response) {
        try {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();

            // Retrieve all cookies from the request
            Cookie c[] = request.getCookies();

            // Display a welcome message using the value of the "uname" cookie
            out.println("Welcome in SOCIS " + c[0].getValue());
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

3. URL Rewriting

- URL rewriting in Java Servlets is a technique used to maintain session state across multiple HTTP requests by encoding session information in URLs. This is often used when cookies are disabled or when you need to support very old browsers that don't handle cookies well.

URLWritingForm.html

```
<html>
<head>
    <title>URL Writing Session FORM</title>
</head>
<body>
    <form action="../servlet/CreateURLServlet" method="post">
        <fieldset>
            <legend>Student Details</legend>
            Student Name: <input type="text" name="uname"> <br>
            Password: <input type="password" name="pwd"> <br>
            <input type="submit" value="Submit">
        </fieldset>
    </form>
</body>
</html>
```

createURLServlet.java

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

public class CreateURLServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String name = request.getParameter("uname");
        String password = request.getParameter("pwd");

        if (password.equals("socis")) {
            // Redirect to FetchURLServlet with username as a parameter
            response.sendRedirect("FetchURLServlet?username=" + name);
        } else {
            out.println("Password is incorrect");
        }
    }
}
```

FetchURlServlet.java

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

public class FetchURLServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String name = request.getParameter("username");

        out.println("Welcome to SOCIS, " + name);
    }
}
```

4. Hidden Fields

- Using hidden fields in Java Servlets is another technique for passing information between different requests or servlets. Hidden fields are HTML input fields that are not displayed to the user but are submitted along with other form data when the form is submitted. Here's how you can implement hidden fields in Java Servlets:

`form.html`

```
<html>
<head>
    <title>Hidden Fields Example</title>
</head>
<body>
    <form action="../servlet/ProcessHiddenFieldsServlet" method="post">
        <fieldset>
            <legend>Student Details</legend>
            <input type="hidden" name="studentId" value="12345">
            Student Name: <input type="text" name="uname"> <br>
            Password: <input type="password" name="pwd"> <br>
            <input type="submit" value="Submit">
        </fieldset>
    </form>
</body>
</html>
```

`ProcessHiddenFieldsServlet.java`

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

public class ProcessHiddenFieldsServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String studentId = request.getParameter("studentId");
        String name = request.getParameter("uname");
        String password = request.getParameter("pwd");

        out.println("<html><body>");
        out.println("<h2>Hidden Fields Example</h2>");
        out.println("<p>Student ID: " + studentId + "</p>");
        out.println("<p>Student Name: " + name + "</p>");
        out.println("<p>Password: " + password + "</p>");
        out.println("</body></html>");
    }
}
```

Database Connectivity

- Connecting a servlet to a database involes the following steps:
    - Setup the Database ex: postgresql
    - Include Postgresql jdbc driver
    - Servlet code for database connectivity
        - Register JDBC driver
        - Open a connection
        - Execute query
- Example: Inserting and Retreiving Data from Database

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

public class UserServlet extends HttpServlet {

    private static final String JDBC_URL = "jdbc:postgresql://localhost:5432/mydb";
    private static final String JDBC_USER = "your_username";
    private static final String JDBC_PASSWORD = "your_password";

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        Connection conn = null;
        Statement stmt = null;

        try {
            // Step 1: Register JDBC driver
            Class.forName("org.postgresql.Driver");

            // Step 2: Open a connection
            conn = DriverManager.getConnection(JDBC_URL, JDBC_USER, JDBC_PASSWORD);

            // Step 3: Execute a query to fetch existing users
            stmt = conn.createStatement();
            String selectSql = "SELECT * FROM users";
            ResultSet rs = stmt.executeQuery(selectSql);

            // Display existing users
            out.println("<h2>Existing Users:</h2>");
            while (rs.next()) {
                int id = rs.getInt("id");
                String username = rs.getString("username");
                String password = rs.getString("password");

                out.println("ID: " + id + ", Username: " + username + ", Password: " + password + "<br
            }
            rs.close();

            // Step 4: Insert a new user
            String newUsername = "new_user";
            String newPassword = "new_password";
            String insertSql = "INSERT INTO users (username, password) VALUES (?, ?)";
            PreparedStatement pstmt = conn.prepareStatement(insertSql);
            pstmt.setString(1, newUsername);
            pstmt.setString(2, newPassword);
            pstmt.executeUpdate();
            pstmt.close();

            out.println("<h2>New User Inserted:</h2>");
            out.println("Username: " + newUsername + ", Password: " + newPassword + "<br/>");

        } catch (SQLException se) {
            // Handle errors for JDBC
            se.printStackTrace();
        } catch (Exception e) {
            // Handle errors for Class.forName
            e.printStackTrace();
        } finally {
            // finally block used to close resources
            try {
                if (stmt != null) stmt.close();
            } catch (SQLException se2) {
                se2.printStackTrace();
            }
            try {
                if (conn != null) conn.close();
            } catch (SQLException se) {
                se.printStackTrace();
            }
        }
    }
}
```

Servelet Collaboration

- Servlet collaboration refers to the process where servlets within the same web application share information or work together to process requests. This is facilitated by various techniques and interfaces provided by the Servlet API.

RequestDispatcher interface

- The **RequestDispatcher** interface, found in the **javax.servlet** package, allows servlets to dispatch requests to other resources such as servlets, JSP files, or HTML files within the same application.
- **Methods**
  i. ***forward() method***:
  - Used to forward the request from one servlet to another resource.
  - Signature:

```
void forward(ServletRequest request, ServletResponse response)
throws ServletException, IOException
```

    - Control is transferred from the current servlet to the target resource without involving the client browser.

ii. ***include() method***:

    - Includes the content of another resource (servlet, JSP, or HTML) in the response of the calling servlet.
    - Signature:

```
void include(ServletRequest request, ServletResponse response)
throws ServletException, IOException
```

    - The response of the included resource is included in the response of the calling servlet.

  Example: In the example provided, ***RequestDispatcherServlet.java*** forwards the request to ***WelcomeStudentServlet.java*** upon successful validation of user credentials. If the validation fails, it includes the ***LoginForm.html*** again for re-entering credentials.

- **HttpServletResponse Interface**
    - The HttpServletResponse interface extends ServletResponse and provides methods specific to HTTP responses. Key methods include setContentType() for specifying the content type of the response and getWriter() for obtaining a PrintWriter to send data to the client.

  Example:

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("Welcome in IGNOU, "+name+"!");
```

ServletContext Interface

- The `ServletContext` interface represents the servlet's view of the web application within the server. It provides methods to interact with the servlet container, such as getting initialization parameters and sharing attributes among servlets.
- **Methods**:
    - ***setAttribute()*** and ***getAttribute()***:
        - `setAttribute(String name, Object obj)` : Stores an attribute in the servlet context.
        - `getAttribute(String name)` : Retrieves an attribute from the servlet context.
- Example:
    - In the provided example, *ServletDemo1.java* sets an attribute using setAttribute() and retrieves it in ServletDemo2.java using getAttribute() after forwarding the request based on certain conditions.

Differences Between forward() and sendRedirect()

- **Forward**
    - Internal redirection within the server.
    - Control remains within the server.
    - `HttpServletRequest` and `HttpServletResponse` objects are forwarded.
- **sendRedirect()**
    - External redirection.
    - Control goes to the client browser, which then makes a new request.
    - Only the URL is passed as a parameter.

Check your Progress - 1

- What is session management? What are the different techniques of session management in servlets? Why is session management needed for HTTP protocol? Explain how cookies can be used for session management.
- Write a servlet program by using HttpSession Object of session management. Servlet program will display creation time when user will visit web page for the first time else it displays last access time. Also display session ID in both the conditions.
- What is the difference between a session and cookie?

Check your Progress - 2

- What is servlet collaboration? How many ways can servlet communicate with each other? What is the purpose of RequestDispatcher Interface?
- What is the difference between forward() and sendRedirect() method?
- Write a servlet program for opening the IGNOU website by using sendRedirect() method.