IT23024

13. How do Servlets and JSPs work together in a web application following the MVC (model-view-controller) architecture? provide a brief use case showing the servlet as a controller, JSP as a view, and Java as the model.

Answer:

Servlet : Handles client requests, interacts with the model, and forwards data to JSP.

JSP (view): Display data to the user.

Java class : Encapsulation business logic or data.

Code:

```
public class User {
        private String Name;
        public User (String name) { this.name = name; }
        public String getName() { return name; }
}
// Controller (servlet)
@WebServlet ("/greet")
public class UserServlet extends HttpServlet {
protected void doGet
    User user = new User ("Alice");
    req.setAttribute ("user", user);
    req.getRequestDispatcher("view.jsp").forward
     (req, res);
}
```

14. Explain the life cycle of a Java servlet. where are the roles of the init(), service(), and destroy() methods? Discuss how servlets handle concurrent requests and how thread safety issues may rise

Answer:

life Cycle Overview: A servlet goes through 3 main stages: initialization, request handling, and destruction.

1. init() – called once when the servlet is first loaded
2. service() – called for each client request
3. destroy() – called once before the servlet is unloaded.

Handing concurrent Request:

A single servlet instance handles multiple request via multiple threads
service() is called simultaneously in separate threads.

15. A single instance of a servlet handles multiple requests using threads. What problems can occur if shared resources are accessed by multiple threads? Illustrate your answer with an example and suggest a solution using synchronization

Answer:

Problem:

A single servlet instance handle multiples request using multiple threads. If shared resource are accessed / modified by multiple threads simultaneously, it can lead to data inconsistency or race condition.

Problematic code:

```
@WebServlet("/counter")
public class CounterServlet extends HttpServlet {
    private int count = 0;
    protected void doGet
        count++;
    res.getWriter().println("count: " + count);
}
```

Synchronization

```
@WebServlet("/counter")
public class CounterServlet extends HttpServlet {

    private int count = 0;

    protected synchronized void doGet

        count++;
        res.getWriter().println("count: " + count);

    }
}
```

Alternative solution:

i) Use local variables inside doGet()

ii) Use AtomicInteger for atomic operations

16. Deserive how the MVC (Model-view-controller) patterr separates conceerns in a Java web application. Explain the advantage of this structure in terms of maintainability and scalability, using a student register as an example.

Answer:

Separation of concerns:

1. Model: Encapsulates data and business logic
2. View: Handle presentation
3. Controller: Handles user request, interacts with the model, and forward data to the view.

Example:

model - student class stores student info, studentDAO handle database operation,

Controller: RegisterServlet process form input, calls studentDAO,

View: register.jsp displays the form. success.jsp show confirmation.

17. In a Java EE application, how does a servlet controller manage the flow between the model and the view. Provide a brief example that demonstrates forwarding data from a servlet to a JSP and rendering a response.

Answer:

In a Java EE (MVC) application, a servlet acts as the controller. Its main job is to:

1. Receive the client request. (HTTP request)
2. Invoke the model (business logic, service, DAO, entities)
3. Store results in request / session scope
4. Forward the request to a view (JSP) for presentation.

(The servlet does not generate HTML directly).

This keep business logic separate from UI, which is the core idea of MVC

Flow: Client → servlet (controllen) → Model → JSP (view)

Browsen
↓
servlet (controllen)
↓
Model (Business Logic/data)
↓
JSP (view)
↓
Responsie to browsen

1. servlet (controllen)

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jav.io.IOException;

public class studentServlet extends HttpServet{
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                        throws ServletException, IOException{

        // model data (normally comes from service/DB)
        String studentName = "Mimi";
        int manks = 85;
        // stone data in request scope
        request.setAttribute("Name", studentName);
        request.setAttribute("manks", manks);
```

```
// Forward to JSP (view)
RequestDispatcher rd =
        request.getRequestDispatcher("result.jsp");

rd.forward(request, response);

    }
}
```

The servlet controls the flow
No HTML is written here
Data is passing using request.setAttribute()

2. JSP (view) - result.jsp

```
<html>
<head>
   <title> Result </title>
</head>
<body>
    <h2> student ResuH</h2>
    Name: ${name} <br>
    marks: ${marks} <br>
    <c:if test = "${marks> = 40}">
       <b> status: pass</b>
     </c:if>
  </body>
  </html>
```

1. JSP only ~~held~~ handles presentation
2. Uses Expression Language to access data
3. No business logic inside JSP

18. Compare and contrast cookies, URL rewriting and HttpSession as methods for session tracking in servlets. Discuss their advantages, limitations, and ideal use cases.

Answer:

1. Cookies:

i) How: Stores small data on the client browser

ii) Advantages: persistent across browser sessions

iii) Limitation: Disabled if client blocks cookies

iv) Use case: Remembering user preference login

2. URL Rewriting:

i) Appends session in URL

ii) Advantages: Works even if cookies are disabled

iii) URLs become messy, security risk

iv) Small application when cookies may be blocked.

3. HttpSession

i) Server-side storage of session objects each client gets a unique session ID.

ii) Secure, stores large objects, no dependency on client setting.

iii) Consumes server memory, expiry often timeout

iv) E-commerce cart, login sessions, sensitive data

Comparison table:

Method	storage	client Dependency	security	lifetime	Use case
Cookie	client	yes	medium	persistent or session	preferences login
URL Rewriting	URL	NO	low	session only	small app no cookies
HttpSession	server	NO	high	session	sensitive data, cart

19. A web Application stores user login information using Http session. Explain how the session tracking in servelets. D. Aerross requessts and how session timout on invalidation is handle seeurely

Answer:

Http session used in web applications to stores user login information and maintain user state acrog multiple nequest. Since Http is a stateless protoed, the server cannot remember user by default. Http session solves this problem

How. Http session le work

when a user log in, the server create an Http session and assigns a unique session ID.

session timeout and Invalidation

for seconity a ssesion has a timeout period. If the user remains inactive for a certain time the session expint automatically and the user must log in again.

20. Explain how spring MVC handle an HTTP request from a browser. Describe the role of the @controller, @RequestMapping, and model objects in separating business logic from presentation. Provide a brief flow example of a login from submission.

Answer:

Spring MVC HTTP Request Handling

i) Spring MVC follows the Model-view-controller

ii) The browser sends an HTTP request to the dispatcherservlet

Role of key component

i) @Controller: Handles requests and connects business logic with the view.

ii) @RequestMapping: Maps URLs and HTTP method to controller methods.

iii) Model: Transfer data from controller to view.

21. Spring MVC uses the DispatcherServlet as a front controller. Describe its role in the request processing workflow. How does it interact with view resolvers and handler mapping.

Answer:

DispatcherServlet acts as the front controller in Spring MVC: It receives all HTTP requests, uses HandlerMapping to find the appropriate controller, and then forwards the request to it. After the controller returns a view name, dispatcherServlet uses a ViewResolver to resolve the actual view and sends the rendered response back to the client.

22. How does Prepared Statement improve performance and security over Statement in JDBC? Write a short example to insert a record into a MySQL table using Prepared Statement.

Answer:

prepared statement improves performance and security compared to statement in the following ways:

Performance Improvement

1. Precompiled SQL

   i) The SQL query is compiled once and reused

   ii) Faster execution when the same query runs multiple times.

2. Efficient Parameter Handling

   i) Parameters are passed separately, avoiding repeated SQL parsing.

Security improvement

1. Prevents SQL Injection

   i) User input is treated as data, not executable SQL

   ii) Protects against malicious input like 'OR '1'='1

Statement limitation:

i) SQL is buit using string concatenation
ii) Vulnerable to SQL injection.
iii) Slower for repeated queries.

Insert Record Using PreparedStatement (MySQL)

```
impont java.sql.*;
public class IntertStudent{
    public static void main (String[] args){

    string url = "jdbc:mysql://localhost:3306/college";
    string user = "root";
    string password = "";
    String sql = "Insert INTO student (id, name, marks)
    VALUES (?, ?, ?)";

    try{
     // load driver
     Class.forName ("com.mysql.cj.jdbc.Driver");

     // create connection
     Connection con = DriverManagen.getconnection
      (url, user, password);
    // prepare statement
    PreparedStatement ps = con.prepareStatement(sql);
```

```
//set parameters
ps.setInt(1, 101);
ps.setString(2, "Karim");
ps.setInt(3, 88);

// Execute query
ps.executeUpdate();
System.out.println("Record inserted successfully"

// close resources
ps.close();
con.close();

} catch (Exception e){
        e.printStackTrace();

    }
}
}
```

Why Prepared Statement is better

Feature	Statement	Prepared Statement
SQL Injection Risk	High	None
Query Compilation	Every time	Once
performance	Slower	Faster
Parameter Support	No	Yes

23. What is resultset in JDBC and how is it used to retrive data from a MySQL database? Briefly explain the use of next(), getString(), and getInt() method with an example.

Answer:

A resultset in JDBC is an object that holds the data returned by executing a SELECT query on a database. It represents a table of data where the cursor initially point before the first row.

The data is retrieved row by row using cursor movement method.

Commonly Used Resultset Methods

- next()
  moves the cursor to the next row
  Return true if a row exists, otherwise false.
  Must be called before accessing any data.
- getString(columnName / columnIndex)
  Retrives String data from the current row.
  Used for VARCHAR, CHAR, TEXT columns.
- getInt(columnName / columnIndex)
  Retrieves integer data from the current row.
  Used for INT, NUMBER columns

Example: Retrieve Data from MySQL Using ResultSet

```
import java.sql.*;
public class FetchStudents {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/college";
        String user = "root";
        String password = "mimi@1701M";

        String sql = "SELECT id, name, marks FROM
            student";

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection con = DriverManager.getConnection(
            url, user, password);

    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(sql);

    // process ResultSet
```

```
while(rs.next()){
    int id = rs.getInt("id");    //getInt()
    String Name = rs.getString("name"); //getString()
    int marks = rs.getInt("marks");

    System.out.println(id+" "+name+" "+marks);
}
rs.close();
stmt.close();
con.close();
} catch (Exception e){
        e.printStackTrace();
    }
  }
}
```

ResultSet stores the result of a SELEct query

next() moves the cursor row by row

getString() retrives text data

getInt() retrives numeric data