Java Server Pages

 -------------------

* servlets is powerful technology for handling the business logic, but when comes to diplaying the content more HTML code should be mixed with Java code.
* Because of this, servlets are becoming complex and hard-to-read, because you need to use many out.println() statements for HTML output.
* So, along with servlets, JSP technology also provided. So, we can use both the technologies to handle the requests and for creating dynamic web pages.
* JSP is more user-friendly and it allows embedding Java code directly in HTML pages using some tags, so that it is easier for creating dynamic web pages.
* JSP is more flexible and even web designers or front-end developers can able to write JSP pages, because they only need to include small snippets of Java code.
* For efficient web application development, both the technologies are used together.

JSP page:

 . A JSP page contains

1. html tags
2. jsp tags

. html tags are for presentation and jsp tags are for generating the dynamic content.

. a jsp file will be translated to a java file and then the java file is complied into class file, by the web container.

. a page translator is used for converting a jsp file to java file, and then java compiler is used for compiling the java file into a class file.



when a jsp file received first request:



when a jsp file received next request:



when a jsp file received first request after modification:



JSP life cycle methods:

----------------------

* Once the object is created, it goes through 3 life cycle methods.
1. jspInit() (Initialization):

 . This method is called only once, right after the instantiation of JSP Servlet object.

 . In this you can put one-time initialization code like estabilishing database connections.

1. \_jspService(req, resp) (Request processing):

 . This is method is called every time a request is made to the JSP page.

 . It handles the request and generates the response.

 . You can’t override this method directly. The container automatically generated this method.

1. jspDestroy() (destruction):

 .This method is called when the JSP page is about to be removed from the memory.

 . In this you can put releasing the resources like closing database connections or any other cleaning up of the resources.

JSP scripting tags:

-------------------

* scripting tags allows you to insert java code directly into an HTML page.
* scripting tags are 3 types.
1. scriptlet tag
2. expression tag
3. declaration tag

 scriptlet tag:

 . we can write this scriptlet tag as <% %>

 . The code written in this scriptlet tag is executed every

 time the page is requested.

 . scriptlet code is placed in the \_jspService() method, so

 the code is executed for every request.



 expression tag:

 . You can write expression tag as, <%= %>

 . You can define a single expression in this at a time.

 . The output of the expression will be directly inserted in

 to the HTML content, by converting the output into a

 string.

 . Each expression is translated into out.print() statement

 and the statement is placed in \_jspService() method.

 So, the expression tag is executed for each request.

 

 declaration tag:

. You can write this declaration tag as, <%! %>

. It is used to declare the variables and the methods that are available for the entire JSP page.

. The code inside the declaration tag is placed directly into the JSP Servlet class, but not into the \_jspService() method.

. The code inside the declaration tag will execute only once, but not for each request.

 

How to write comments in a jsp file?

 . comments you can write in two styles.

 1. jsp style comment

 2. html style comment

 . jsp style comment you can write as, <%-- comment --%>

 . html style comment you can write as, <!-- comment -->

 . jsp style comment is not included in the generated HTML, so on the browser, if source code is opened for a page then this jsp style comments are invisible.

 . html style comment is included in the generated HTML. So, on the browser, if source code is opened for a page, then html style comments are visible.

counter example:

---------------

 . create a new dynamic web project in the eclipse ide, with the name JspCounterApp.

 . add servlet-api.jar and jsp-api.jar to the build path

 . Expand src folder, the main folder. You can see webapp folder.

 . right click on the webapp folder, and create a new jsp file with the filename index.jsp, with below code.

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

 pageEncoding=*"UTF-8"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

 <%-- This declaration tag --%>

 <%!

 **int** counter = 0;

 %>

 <!-- scriptlet tag -->

 <%

 counter++;

 %>

 <font color=*'green'* size=*8*> The counter value : <%= counter %></font>

</body>

</html>

. execute the application, on Tomcat server.

. open the browser, and enter the below url.

 <http://localhost:8080/JspCounterApp/index.jsp>

. every time when you reload the page, the counter value will be incremented.

login example:

 . create a new dynamic web project in the eclipse ide, with the name JspLoginApp.

 . add servlet-api.jar and jsp-api.jar to the build path

 . Expand src folder, the main folder. You can see webapp folder.

 . right click on the webapp folder, and create a new jsp file with the filename login.jsp, with below code.

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

 pageEncoding=*"UTF-8"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

 <!-- Login form design -->

 <form action = *"./login.jsp"* method = *"post"* >

 Username : <input type=*"text"* name=*"username"*> <br>

 Password : <input type=*"password"* name=*"password"*> <br>

 <button type=*"submit"*>submit</button>

 </form>

 <hr color=*'green'*>

 <%-- sciptlet tag --%>

 <%

 String message = "";

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

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

 **if** ( authenticate(username, password) ) {

 message = "Hello " + username + "<br> Login success";

 }

 **else** {

 message = "Hello " + username + "<br> Login failed";

 }

 %>

 <%-- declaration tag --%>

 <%!

 **public** **boolean** authenticate(String username, String password)

 {

 **if** ("John".equals(username) && "john@123".equals(password))

 **return** **true**;

 **else**

 **return** **false**;

 }

 %>

 <%-- expression tag --%>

 <font color=*'blue'* size=*6*> <%= message %> </font>

</body>

</html>

JSP directives:

 . Directives are special instructions processed by the container and they are used to configure global information.

 . Directives do not produce any output into the HTML response, but these are used to configure the behavior of the JSP page.

 . There are 3 directives in JSP.

1. include directive
2. page directive
3. taglib directive

include directive( <%@ include %> ):

 . It includes the content of other file into the current JSP file at the page translation time.

 . The included content becomes part of the current JSP file, before it is compiled.



. The include directive has only one attribute called file and the

 included file must be either a jsp file or a html file.

page directive ( <%@ page %> ):

 . This page directive is used to define page-level attributes to a JSP page.

 . some commonly used attributes are,

 i)language: specifies the language to be used in the JSP page.

 The default value is “java”.

 <%@ page language = “java” %>

ii) contentType : specifies the MIME type and character encoding of the response generated by the JSP file.

 <%@ page contentType=”text/html; charset=UTF-8” %>

iii) import: Specifies the Java classes or packages that should be imported into JSP page.

 <%@ page import=”java.util.List, java.time.LocalTime” %>

iv) session: Specifies whether a JSP page is a part of session or

 not.

 The default value is true, it means, the JSP page has

 access to the session object.

 If you set false, the JSP page will not have access to

 the session object.

 ex1:

 <%@ page session=”false” %>

 <%= session.getId() %> --🡪 gives an error

 ex2:

 <%@ page session=”true” %>

 <%= session.getId() %> -🡪 prints session id

 v) errorPage: Specifies the URL of the other JSP page, that should

 be used when an exception occurs in the current JSP

 page.

 <%@ page errorPage = “error.jsp” %>

 vi) isErrorPage: Specifies whether the current page is ready to act

 as an error page to handle the exceptions or not.

 The default value is false.

 If you set the value to true, then the page acts

 as an error page and it can use the exception

 object.

 <%@ page isErrorPage = “true” %>

taglib directive ( <%@ taglib %> ):

 . It is used to import a tag library into a JSP page.

 . A tag library is a group of tags, similar a Java package which

 is a group of classes.

 . Like a Java package has a name, a tag library also has a name

 and it is called uri.

 . If a tag library is imported then the JSP page is allowed to

 use the tags from that library.

 <%@ page uri=”http://java.sun.com/jsp/jstl/core” prefix=”c” %>

Example:

 This example has 3 files.

 header.html

 divide.jsp

 error.jsp

 . header.html has a simple message Ashokit to be displayed as header.

. divide.jsp has a form to enter two input values and also a scriptlet to divide the two numbers.

 . if exception occurs in the divide.jsp then the control goes to error.jsp. This error.jsp will display the exception.

header.html

-----------

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

 <center>

 <h2> Ashokit </h2>

 </center>

</body>

</html>

divide.jsp

----------

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

 pageEncoding=*"UTF-8"* errorPage = *"error.jsp"* %>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

 <%@ include file = *"header.html"* %>

 <hr>

 <form action = *"./divide.jsp"*>

 First Number : <input type=*"text"* name = *"first"*> <br>

 Second Number : <input type=*"text"* name=*"second"*> <br>

 <input type=*"hidden"* name=*"x"* value=*"1"*>

 <button type=*"submit"*>Divide</button>

 </form>

 <%

 **if**( request.getParameter("x") != **null**) {

 **int** fno = Integer.parseInt(request.getParameter("first"));

 **int** sno = Integer.parseInt(request.getParameter("second"));

 **int** result = fno / sno;

 out.println("Result = " + result);

 }

 %>

</body>

</html>

error.jsp

---------

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

 pageEncoding=*"UTF-8"* isErrorPage = *"true"* %>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

 <font color = *'red'*>

 <%= exception %>

 </font>

</body>

</html>

implicit objects in JSP:

------------------------

. implicit objects are pre-defined objects that are automatically available for the developers, without creating them explicitly.

 object name type

 ------------ --------

 request HttpServletRequest

 response HttpServletResponse

 out JspWriter

 session HttpSession

 config ServletConfig

 application ServletContext

 exception Throwable

 page Object

 pageContext PageContext

Expression Language(EL):

--------------------

 . EL in JSP is used to simplify the accessibility of the data stored in the Java objects, request parameters, session attributes and application attributes.

 . This EL was introduced to reduce the java code for accessing the data from these objects and to make the JSP pages more cleaner and easy to maintain.

 . The basic syntax of EL is ${expression}.

 . The given expression is evaluated at runtime by the webcontainer.

 . The expression can use operators like,

1. arithmetic operators : +, -, \*, %, /
2. relational operators : ==(eq), !=(ne), <(lt), >(gt),

 <=(le), >=(ge)

1. logical operators: &&(and), ||(or), !(not)

User.java

---------

**package** com.example;

**public** **class** User {

 **private** **int** id;

 **private** String name;

 **private** String email;

 **public** User(**int** id, String name, String email) {

 **super**();

 **this**.id = id;

 **this**.name = name;

 **this**.email = email;

 }

 **public** **int** getId() {

 **return** id;

 }

 **public** **void** setId(**int** id) {

 **this**.id = id;

 }

 **public** String getName() {

 **return** name;

 }

 **public** **void** setName(String name) {

 **this**.name = name;

 }

 **public** String getEmail() {

 **return** email;

 }

 **public** **void** setEmail(String email) {

 **this**.email = email;

 }

}

index.jsp

---------

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

 pageEncoding=*"UTF-8"* import=*"com.example.User"* %>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

 <%!

 User user = **null**;

 %>

 <%

 user = **new** User(1001, "John", "john@gmail.com");

 request.setAttribute("user", user);

 %>

 <h2>

 id : ${user.id} <br>

 name : ${user.name} <br>

 email : ${user.email}

 </h2>

</body>

</html>

JSTL core tags:

--------------

 . JSTL --- JSP Standard Tag Library

 . JSTL core tags are provided to simplify the tasks related to iterations, conditions, creating variables, displaying the variables and so on.

 . To use, JSTL core tags in a JSP page, first you have to import the tag library, like below.

 <%@ taglib uri = “http://java.sun.com/jsp/jstl/core” prefix = “c” %>

1. <c:set> : It assigns a value to a variable.

 The variable can be stored in a scope like page or request

 or session or application, to make this variable visible or

 accessible across other pages.

<c:set var = “role” value = “admin” />

1. <c:out> : It displays the value of a variable.

 <c:out value = “${role}” />

1. <c:if> : It defines if condition. The body of this will be executed

 if the given condition is true.

 <c:if test = “${role == ‘admin’}”>

 <h3> Welcome Admin! </h3>

 </c:if>

1. <c:choose>, <c:when>, <c:otherwise> : These are used to define a switch statement.

<c:choose>

 <c:when test = “${role eq ‘admin’}”> Welcome admin</c:when>

 <c:when test = “${role eq ‘manager’}”>Welcome manager</c:when>

 <c:otherwise>Welcome guest</c:otherwise>

</c:choose>

1. <c:forEach>: Iterates over a range of values like 1 to 10, or iterates over a collection or iterates over an array.

<c:forEach var=”i” begin=”1” end=”10” step=”1”>

 <c:out value=”${i}”/>

</c:forEach>

<c:forEach var=”e” items=”${list}”>

 <c:out value=”${e.empno}”/> <br>

 <c:out value=”${e.ename}”/> <br>

 <c:out value=”${e.sal}”/>

</c:forEach>

1. <c:import> : imports the contents from another file or from an external resource.

 <c:import url = “header.html”/>

 <c:import url=”http://www.ashokit.in/index.html”/>

1. <c:redirect> : It redirects the user to a different URL from the current JSP page.

<c:redirect url = “http://www.ashokit.in”/>

MVC architecture:

------------------------

 . MVC -- Model View Controller

 . We can directly develop web applications with servlets and JSP’s.

 . Using MVC architecure, if you develop web applications then you will

 serveral advantages like a good structure for the project,

 modularity, improves maintainability, separation of code etc.

 . If you develop a web application, with a servlet, then you have to

 write the application logic and view related logic in the same file.

 This leads to large complex classes and it is difficult to maintain

 the classes.

 . For example, You want to create a simple online store application,

 the servlet should contain the code to validate the user input,

 executing the database queries for fetching the products, applying

 the discounts, calculating the total costs and also html code to

 display the products. This approach tightly couples the business

 logic and presentation logic.

 . With MVC, the responsibilites can be separated into 3 components.

1. Model : Model manages the data and the business logic of the

 application.

 There can be multiple model classes also in application.

 For ex, in online store application, the model classes

 would be like Product, Order , Cart etc. They also

 include database interactions and the calculations.

1. View : View is responsible for rendering the UI elements.

 View contains presentation logic.

 There can be multiple views in an application.

 For ex, a JSP file, diplaying the products is a view, or

 a JSP file, displaying the images and other details of

 a product is a view.

1. Controller : Controller manages the flow of a request in an

 application.

 It captures the HTTP request, performs input

 validations, interacting with models and selects

 appropriate views for the response.

 Controller acts as a mediator between models and

 views.

 . In MVC, each component has a dedicated role, so making changes is

 easy and it do not distrub the other components.

 . For example, I want to modify the view to display the products from

 horizontal to vertical format. I can only make changes to the View,

 with out modifying the Model or Controller.

 User Login process example with MVC:

-------------------------------------

1. a user fills login form and then clicks on “Login”
2. The controller receives the request. It extracts the credentials and then perfoms input validation. After that, it will pass the details to the model.
3. The model interacts with the database, checks for the matching username and password. If valid, the model might return user’s profile. If not valid, the model might return an error message.
4. Based on the model’s output, the controller will select the appropriate view. If success, the controller forwards request to dashboard view, if fails, the controller forwards request to failure view. The view will provide the response back to the user.



MVC Application example:





Employee.java

**package** com.example.model;

//model class

**public** **class** Employee {

 **private** **int** empno;

 **private** String ename;

 **private** **double** sal;

 **private** **int** deptno;

 **public** Employee() {

 }

 **public** Employee(**int** empno, String ename, **double** sal, **int** deptno) {

 **super**();

 **this**.empno = empno;

 **this**.ename = ename;

 **this**.sal = sal;

 **this**.deptno = deptno;

 }

 **public** **int** getEmpno() {

 **return** empno;

 }

 **public** **void** setEmpno(**int** empno) {

 **this**.empno = empno;

 }

 **public** String getEname() {

 **return** ename;

 }

 **public** **void** setEname(String ename) {

 **this**.ename = ename;

 }

 **public** **double** getSal() {

 **return** sal;

 }

 **public** **void** setSal(**double** sal) {

 **this**.sal = sal;

 }

 **public** **int** getDeptno() {

 **return** deptno;

 }

 **public** **void** setDeptno(**int** deptno) {

 **this**.deptno = deptno;

 }

}

EmployeeDAO.java

package com.example.model;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

//Model class

public class EmployeeDAO {

 private static Connection conn;

 private static PreparedStatement pstmt;

 private static Statement stmt;

 static {

 try {

 Class.forName("com.mysql.cj.jdbc.Driver"); //optional

 conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root","root");

 stmt = conn.createStatement();

 pstmt=conn.prepareStatement("INSERT INTO EMP(EMPNO,ENAME,SAL,DEPTNO) VALUES(?, ?, ?, ?)");

 }

 catch(Exception ex) {

 ex.printStackTrace();

 }

 }

 public List<Employee> getAllEmployees() {

 List<Employee> empList = new ArrayList<>();

 try {

 ResultSet rs = stmt.executeQuery("SELECT EMPNO,ENAME,SAL,DEPTNO FROM EMP");

 while( rs.next() ) {

 Employee e = new Employee();

 e.setEmpno(rs.getInt(1));

 e.setEname(rs.getString(2));

 e.setSal(rs.getDouble(3));

 e.setDeptno(rs.getInt(4));

 empList.add(e);

 }

 rs.close();

 }

 catch(Exception ex) {

 ex.printStackTrace();

 }

 return empList;

 }

 public void addEmployee(Employee emp) {

 try {

 pstmt.setInt(1, emp.getEmpno());

 pstmt.setString(2, emp.getEname());

 pstmt.setDouble(3, emp.getSal());

 pstmt.setInt(4, emp.getDeptno());

 pstmt.executeUpdate();

 }

 catch(Exception ex) {

 ex.printStackTrace();

 }

 }

}

EmployeeController.java

package com.example.controller;

import java.io.IOException;

import java.util.List;

import com.example.model.Employee;

import com.example.model.EmployeeDAO;

import jakarta.servlet.RequestDispatcher;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

//Controller class

@WebServlet( value = "/controller")

public class EmployeeController extends HttpServlet {

 @Override

 protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

 // TODO Auto-generated method stub

 String action = req.getParameter("action");

 if ( "new".equals(action) ) {

 RequestDispatcher dispatcher = req.getRequestDispatcher("employee-form.jsp");

 dispatcher.forward(req, resp);

 }

 else {

 RequestDispatcher dispatcher = req.getRequestDispatcher("employee-list.jsp");

 EmployeeDAO dao = new EmployeeDAO();

 List<Employee> empList = dao.getAllEmployees();

 req.setAttribute("employees", empList);

 dispatcher.forward(req, resp);

 }

 }

 @Override

 protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

 // TODO Auto-generated method stub

 int empno = Integer.parseInt(req.getParameter("empno"));

 String ename = req.getParameter("ename");

 double sal = Double.parseDouble(req.getParameter("sal"));

 int deptno = Integer.parseInt(req.getParameter("deptno"));

 Employee newEmp = new Employee();

 newEmp.setEmpno(empno);

 newEmp.setEname(ename);

 newEmp.setSal(sal);

 newEmp.setDeptno(deptno);

 EmployeeDAO dao = new EmployeeDAO();

 dao.addEmployee(newEmp);

 resp.sendRedirect("index.jsp");

 }

}

index.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

 pageEncoding=*"UTF-8"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Index Page</title>

</head>

<body>

 <h1>Welcome to Employee Management</h1>

 <a href=*"controller"*>View Employees</a>

</body>

</html>

employee-list.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

 pageEncoding=*"UTF-8"*%>

<%@ page import = *"java.util.List"* %>

<%@ page import = *"com.example.model.Employee"* %>

<%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

 <h2> Employees List </h2>

 <hr>

 <table border = *"1"*>

 <tr> <th>EMPNO</th> <th>ENAME</th> <th>SAL</th> <th>DEPTNO</th> </tr>

 <c:forEach var=*"e"* items=*"*${employees}*"*>

 <tr>

 <td> <c:out value=*"*${e.empno}*"*/> </td>

 <td> <c:out value=*"*${e.ename}*"*/> </td>

 <td> <c:out value=*"*${e.sal}*"*/> </td>

 <td> <c:out value=*"*${e.deptno}*"*/> </td>

 </tr>

 </c:forEach>

 </table>

 <a href=*"controller?action=new"*>Add New Employee</a>

</body>

</html>

employee-form.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

 pageEncoding=*"UTF-8"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<h2> ADD NEW EMPLOYEE </h2>

<form action=*"controller"* method=*"post"*>

 Empno : <input type=*"text"* name=*"empno"*> <br>

 Ename : <input type=*"text"* name=*"ename"*> <br>

 Sal : <input type=*"text"* name=*"sal"*> <br>

 Deptno : <input type=*"text"* name=*"deptno"*> <br>

 <button type=*"submit"*>Submit</button>

</form>

</body>

</html>