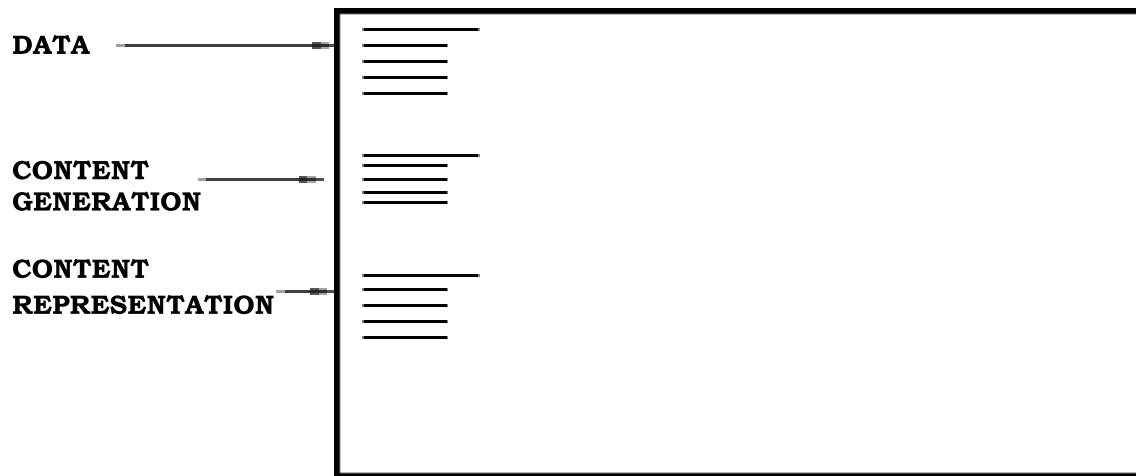


Java Server Pages

□ WHY JSP?

Servlet follows Component architecture, follows request response model.



- All the code goes inside **service()** / **doGet ()** / **doPost()**.
- This makes Servlet monolithic in nature.
- It is a tough job to write all the code in same method.
- Hence there emerged a need for a clear separation between **CONTENT GENERATION & CONTENT REPRESENTATION**.
- Using JSP Separation found was something like this.

□ Java Server Pages

- JSP stands for **Java Server Pages**.
- **JSP is a text document with any name which can contain 100% HTML tags & additional tags to embed java code.**
- It is a java technology which allows software developer to dynamically generate HTML, XML or other types of document in response to Web client request.
- This document is stored in WEB-ROOT of a WEB-APPLICATION.
- Whenever the request comes for any jsp for the first time, JSP ENGINE (which) is part of Container translates **.jsp** into a **.java** file.
- Translated **.java** file is nothing but a 100% HttpServlet.
- Whenever any changes are made in **.jsp** file. JSP ENGINE retranslates &

Recompile it.

□ **ADVANTAGES OF JSP**

1) WRITE ONCE, RUN ANYWHERE

- JSP technology is platform independent.
- JSP can be written on any platform, it can run from any platform & can be accessed from any web-browser on any platform.

2) HIGH QUALITY TOOL SUPPORT

- An explicit goal of Java Server Pages design is to enable the creation of high quality portable tools.

3) REUSE OF COMPONENTS & TAG LIBRARIES

- **JSP** uses reusable components such as
 - ❖ JAVA BEANS
 - ❖ ENTERPRISE JAVA BEANS
 - ❖ TAG LIBRARIES
- This save development time as well as gives cross-platform power & flexibility of Java programming.

4) SEPERATION OF DYNAMIC & STATIC CONTENT

- **JSP** enables separation of dynamic contents from static contents that is inserted in static template.
- This makes work easier & hence diminishes the dependency of WEB-DESIGNER over WEB-DEVELOPER & vice-versa.

5) SUPPORTS SCRIPTING & ACTIONS

- JSP supports scripting elements as well as actions.
- Action permits ENCAPSULATION.
- Scripting provides mechanism to glue-together functionalities.
-

□ **SERVLETS V/S JSP**

1) Servlet

- ___Servlet does not have any separation between dynamic & static contents.

JSP

- JSP has a clean separation between dynamic & static content.

2) Servlet

- ___Servlet does not have support for implicit objects.

JSP

- ___JSP supports implicit objects.

3) Servlet

- Servlet supports any protocol including Http.

JSP

- ___JSP supports only Http protocol.

4) Servlet

- ___Servlets are pure java program stored with extension (.java).

JSP

- JSPs are document-centric stored with extension(.jsp)

5) Servlet

- ___If any changes are made in **.class** file. It has to be redeployed

JSP

- ___Whenever any changes are made to (.jsp) file, JSP ENGINE
Retranslates & recompiles, hence no redeployment.

□ JSP ARCHITECTURE

--- Important Question. Expect for 6 or 8 marks.

- ___Whenever a request arises for **.jsp** file.

● JSP-ENGINE will :

- ___Translate **.jsp** file into a **.java** file.
- ___Compiles the file into a servlet file.

● Container will:

- ___Load the servlet class.
- ___Instantiate it using default constructor.
- ___Invokes `jspInit()`
Required that :
 - `jspInit()` does not throw any exception.
 - `jspInit()` returns in stipulated time.
- Put the servlet in service.
- Creates request & response object.
- Starts the thread, invoke `_jspService()` & passes request & response object as arguments.
- Retrieve the request parameter, perform processing, send responses to the client using writer object.
- Close the writer object.
- When Container shut downs `jspDestroy()` is invoked.

□ **JSP LIFE CYCLE METHODS**

public void jspInit()

- This method is invoked when JspPage is initialised.
- This method is called once in the life-time of JSP Page.
- To put Jsp in service, jspInit() must get completed successfully.

**public void _jspService(HttpServletRequest request,
 HttpServletResponse response) throws
 ServletException, IOException**

- The _jspService() corresponds to the body of JSP page.
- This method is defined automatically by the Container & this should not be defined by JSP page author.
- This method is called many a times in its life cycle .

public void jspDestroy()

- This method is invoked when Jsp Page is about to be destroyed.

□ **JSP Comments**

- Comments can be used in JSP.
- Two different types of comments are allowed in JSP.

i) HTML COMMENT

<!-- HTML COMMENT -->

- Container passes the straight to the client where browser intercepts this as a comment.
- Hence any attempt of client for view source on browser will end up showing comments.

ii) JSP COMMENT

<%-- JSP COMMENT --%>

- These are for page developers, and are stripped off the translated page.
- Hence a client can't see any of these comments.

□ JSP Tags

i) Directive

- a)Page Directive
- b)Include Directive

ii)Scripting Element

- a) Expression
- b) Data Declaration/Method Definition
- c) Scriptlets

a)Page Directive

- This tag is always given above <HTML> tag.
- Controls resultant translated page.
- No output.

Syntax:

```
<%@ page attribute="value" attribute="value" %>
```

Attribute of page directive.

i) language

- By default java is used.

```
<%@ page language="Java" %>
```

```
<html>
```

```
|
```

```
</html>
```

-It is used for future consideration, where JSP might support language other than java.

ii) import

-import package required by this JSP page.

-This is the only attribute which can be repeated more than once.

```
<%@ page language="Java" import="java.util.*" import="java.sql.*" %>
```

OR

```
<%@ page language="Java" import="java.util.*, java.sql.*" %>
```

iii) buffer

-JSP does not uses PrintWriter, but uses JSPWriter.

-Buffer can be "none", "8 kb", "12 kb" as per requirement.

```
<%@page buffer="15 kb" %>
```

iv) autoflush

-Defines whether the buffered output is flushed automatically.

-Default value is true.

-If false is passed, an IOException is generated.

```
<%@ page autoflush="true" %>
```

v)isErrorPage

-Defines whether current page represent JSP error page.

-Default value is "false",but if it's change to true,page has access to implicit exception object.

```
<%@ page isErrorPage="true" %>
```

vi) errorPage

-Defines the URL to which Exception should be sent.

-The target JSP's isErrorPage value should be true.

```
<%@ page errorPage="MyExceptionPage.jsp" %>
```

vii) session

-Defines whether page will have implicit session object.

-Default value is "true".

```
<%@ page session="true" %>
```

equivalent to getSession(true);

```
<%@ page session="false" %>
```

equivalent to getSession(false);

viii) isThreadSafe

-Defines whether the generated servlet need to implement the SingleThreadModel.

-Default value is "true".

b)include Directive

-include directive are given after <html> and before </html>.

-It makes the static inclusion of .html/.jsp file.

-These files are include during translation.

Syntax:

```
<%@ include file="somename.html" %>
```

-Include .html/.jsp file should not contain <html> and </html>.

Header.html

```
<body>
```

```
<h6>Welcome To Tech Solution</h6><br>
```

```
<hr>
```

```
</body>
```

Footer.html

```
<body>
```

```
<hr>
```

```
<br>
```

```
<h6>This website is a copyright of Tech Solution</h6>
```

```
</body>
```

IncludeJSP.jsp

```
<%@ page language="Java" %>
```

```
<html>
```

```
<body>
```

```
<%@ include file="Header.html" %>
```

```
<p> Please login.....</p>
```

```
>%@ include file="Footer.html" %>
```

```
</body>
```

```
</html>
```

□ Scripting Elements

a)Expressions

b)Data Declaration and Method Definition

c)Scriptlets

a) Expressions

-JSP expression automatically print out whatever is being sent between the tags.

-Expressions are evaluated and output is sent to the browser.

-Expressions are later resolved and consumed by HTML itself.

```
<%= expression %>
```

[Anything which comes on right hand side int i=1;

```
int z=x+y;
```

```
Date dt =new Date();
```

Point To REMEMBER:

NO semi-colon at the end.

Ease of using Expressions.**Without expression:**

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

```
<html>
```

```
<body>
```

```
Date: <% out.println(new Date());%>
```

```
</body>
```

```
</html>
```

With expression:

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

```
<html>
```

```
<body>
```

```
Today's Date:<%=new Date() %>
```

```
</body>
```

```
</html>
```

-Whenever container encounters expression, Container

takes everything typed between the <%= %> and put it as an argument in out.println()

```
<%= new Date() %>
```

Becomes

```
out.println(new Date());
```

Tip for Students:

You people always by mistake put up a semi-colon at the end which result in

```
<%= new Date(); %>
```

Becomes

```
out.println(new Date());
```

Which becomes error then.

Hence, No semi-colon(;) at the end of Expression.

FirstJSP.jsp

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

```
<html>
```

```
<body>
```

```
<%@ include file="Header.html" %>
```

```
<b>
```

```
<Hello User!!!Welcome to the world of JSP !!
```

```
</b>
```

```
Current Date : <%= new Date() % >
```

```
<%@ include file="Footer.html" %>
```

```
</body>
```

```
</html>
```

b) Data Declaration and Method Definition

-There may arise a case when a programmer requires to use his own method.

-But till yet, we have seen that whatever we write goes in servlet method (service ()).

-As JSP Declaration doesn't provide any output they are used in conjunction with JSP Expression and JSP Scriptlets.

```
<%! Data Declaration; %>
```

```
<%! Method Definition()
```

```
{
}

%>
```

```
<%! int count=0; %>

<%! public String m1()
{
return "Hello";
}

%>
```

-Method returning some values are allowed in JSP Declaration.

<http://localhost:8080/TechWebApp/j2?name=Tom>

HelloDeclarationJSP.jsp

```
<html>
<body>
<%!
public String sayHello(String name)
{
return "Hello" + name;
}

%>
```

```
Message:<%= sayHello(request.getParameter("name")) %?>
```

```
</body>
</html>
```

c) Scriptlets

-Any piece of code written in scriptlet goes into service() method.

- Any complex or simple java code can be written in scriplet.
- Scriptlet are meant for embedding java code.
- Scriptlet never get translated.
- Hence a very simple but logical point to remember for scriplet is that Scriptlet NEVER CAUSE TRANSITION ERRORS.
- Scriptlet CAN BE BROKEN by STATIC content.

```
<% Java Code %>
```

```
<%-----
```

```
-----
```

```
-----%>
```

- Nothing but pure java code goes in the dotted area.
- Any data declared become local to service().

TomJSP.jsp/j3

```
<html>
```

```
<body>
```

```
<%
```

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

```
if(name.equalsIgnoreCase("Tom"))
```

```
{
```

```
    out.println("Hi Tom!!!");
```

```
    out.println("How are You ?");
```

```
}
```

```
else
```

```
{
```

```
    out.println("Who You ?");
```

```

        out.println("I don't know You");
    }
%>

```

```

</body>
</html>

```

URL: <http://localhost:8080/TechWebApp/j3 ?name=Tom>
ScriptletTomJSP.jsp/j4

```

<html>
<head> Scriptlet Tom </head>
<body>

```

```

<b>
    A JSP to demonstrate that SCRIPTLET can be broken by static content

```

```

</b>
<% String name=request.getParameter("name");
    if(name.equals("tom"))
    {
%>
<b>
Hi Tom! How are you ?
</b>
<br>
<%
    }

```

```

    else
    {
%>

<b>
Who you ? I don't know you ? <%= name %>

<%
    }
%>

</body>

</html>

```

□ **OBJECT SCOPE**

- Scope of an object define
- 1) How long the object is available.
- 2) Form on which JSP object will be available

There are four scopes

- 1) request scope**
- 2) session scope**
- 3) page scope**
- 4) application scope**

1) request scope

- request scope indicates that object is bound to **javax.servlet.http.HttpServletRequest**.
- Hence, object in request scope can be accessed using `getAttribute()` on implicit request object.
- The object is distinct for every client request.

- Scope of such objects is available, till **HttpServletRequest** object exists.

Implicit objects coming under request scope: **request**.

2) session scope:-

- session scope indicates that object is bound to **javax.servlet.http.HttpSession**
- Object in session scope can be accessed using `getValue()` on implicit object session.
- Object is different for different client.
- Object in this scope helps in session binding.
- Scope of such object is available, till the client's session valid.

Implicit objects coming under session scope : session

3) page scope

- page scope indicates that object is bound to **javax.servlet.jsp.PageContext** .
- * Object in page scope can be accessed using `getAttribute()` of implicit `pageContext` object.
- * Object in page scope are created & destroyed for each client request to the page.
- * scope of such object can be considered as long as page is responding to the current request.

Implicit objects coming under page scope are:
response,out,config,page,pageContext,exception

4) application scope :-

- application scope indicates that the object is bound to **javax.servlet.ServletContext**.
- object in application scope can be accessed using `getAttribute()` of implicit application object.

- scope of this objects are very persistent, they are destroyed when Container is put down.

Implicit objects coming under application scope: **application**

□ IMPLICIT OBJECTS

- JSP simplifier authoring & provides certain objects implicitly to be accessed within a JSP page without any explicit declaration.
- These objects are called as implicit objects .
- These objects are not declared explicitly but they are provided by Container during translation phase.

There are in all 9 implicit objects

TIP FOR STUDENT:it is always recommended to remember the name of this implicit object & their classes in context to interviews.....

There are 9 implicit objects:-

1) request.

2) response

3) out

4) session

5) application

6) config

7) page

8)pageContext

9)exception

1) request:-

- request object represents all the information about HttpRequest.
- Instance of **javax.servlet.http.HttpServletRequest**
- **Scope:request scope.**
- Using this object we can access headers cookies ,etc.

2) response:-

- response object represents response to the client.
- Instance of : **javax.servlet.http.HttpServletResponse.**

- Scope:page scope

- response object is used for adding cookies, redirecting the calls,etc

3) out:-

- out object represents output stream.
- out objects are used for sending arguments to print methods.
- Instance of:**javax.servlet.jsp.JSPWriter**

-Scope:page scope**4) session:-**

- session object is used to track client's information across the session.
- Instance of: **javax.servlet.http.HttpSession**

- Scope:session scope.**5) application:-**

- application object represent the context for the JSP page.
- application object is accessible to any object used within JSP page.
- Instance of:**javax.servlet.ServletConfig**

- Scope:page scope.**6) config:-**

- config object provides access to the initialization parameters
- Instance of : **javax.servlet.ServletConfig**

- Scope : page scope**7) page:-**

- page refers to the instance of JSP implementation class, i.e the JSP itself.

- accessed using 'this' reference.
- Instance of : **javax.lang.Object**

- **Scope:page scope.**

8) **pageContext:-**

- pageContext encapsulates other implicit objects.
- If a class is given pageContext reference it get access to all the objects from all the scope
- Instance of : **javax.servlet.jsp.PageContext**
- **Scope:page scope.**

9) **exception:-**

- Refers to runtime exception that resulted in invoking the errorPage.
- Available only in error page.
- errorPage:- a JSP which has is errorPage attribute true in page directive.
- Instance of :**java.lang.Throwable**
- **Scope:page scope.**

□ JSP DOCUMENT

- JSP document is an XML document.
- Hence it must comply with xml standards.
- JSP documents must be a well formatted file,with each start tag having an end tag & document must have only one root element.
- Most of the JSP syntax are xml –complaint
- Those elements that are not complaint are summarized in following table.
-

| SYNTAX ELEMENTS | STANDARD SYNTAX | XML SYNTAX |
|------------------------|---|---|
| Comments | <%..... ..%> | <!.....> |
| Declaration | <%!..... ..%> | <jsp:declaration>.. ...</jsp:declaration> |
| Directives | <%@ include....%> <%@ page....%> | <jsp:directive.include.../> <jsp:directive.page...../> |

| | | |
|------------|------------------|--|
| | <%@ taglib....%> | xmlns:prefix="tag library url" |
| Expression | <%=%> | <jsp:expression>..... ...<jsp:expression> |
| Scriptlet | <%.....%> | <jsp:scriptlet>...<jsp:scriptlet> |

□ CHARACTER QUOTING CONVENTIONS

| SYNTAX | PURPOSE |
|--------|---|
| <\% | Used in static html where we want to use "<%" |
| %> | Used in static html where we want to use "%>" |
| \' | Used to insert single quote in static HTML page |
| \" | Used to insert double quotes in static HTML page. |

□ TECHNICAL CONSIDERATION FOR JSP

Configuring **web.xml** for **jsp** file

```

<web-app>
<servlet>
    <servlet-name>MyFirstJSP_jsp</servlet-name>
    <jsp-file>FirstJSP.jsp</jsp-file>
</servlet>

<servlet-mapping>
    <servlet-name>MyFirstJSP</servlet-name>
    <url-pattern>FirstJSP.jsp</ url-pattern >
</servlet-mapping>
</web-app>

```

Where to keep JSP File.

e.g. I want to keep my file.

MyFirstJSP.jsp in WebApp.

- **ALWAYS remember .jsp file are kept in WEB-ROOT folder of a Web-App**
-

PROGRAMS

A JSP FOR REQUESTING VALUES FROM A HTML FORM AND REPLYING BACK.

NameAgeHtml.html

```
<html>

<body>

<form action="http://localhost:8080/ZWebApp/NameAgeJSP.jsp" method="POST" >

Name : <input type="text" name="name">

<br>

AGE : <input type="number" name="age">

<br>

<input type="submit" value="submit">

</form>

</body>

</html>
```

NameAgeJSP.jsp

```
<html>

<body>

<%

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

int age=Integer.parseInt(request.getParameter("age"));

%>
```

```
<B>Hi ! <%= name %> You are <%= age %> years old.
```

```
<br>
```

```
Next year you will be <%= age+1 %> years old.
```

```
</body>
```

```
</html>
```

● **PROGRAM**

JSP WITH JDBC.

A JSP TO INSERT A NEW USER IN LOGIN TABLE

Insert.html

```
<html>
```

```
<body>
```

```
<form action="http://localhost:8080/TechWebApp/InsertUserJSP.jsp" method="POST" >
```

```
UserName : <input type="text" name="user">
```

```
<br>
```

```
Password: <input type="text" name="password">
```

```
<br>
```

```
<input type="submit" value="register">
```

```
</form>
```

```
</body>
```

```
</html>
```

InsertUserJSP.jsp

```
<%@ page import="java.sql.*" %>
```

```
<html>
```

```
<body>
```

```
<%
```

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

```

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

%>

<B>Inserting a new User in Login Table.... <br>Process Going On..

<br>Please wait for the notification.....

<hr>

<%

try

    {

        Class.forName("com.mysql.jdbc.Driver");

        Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/Tech","root","tech");

        String query="Insert into Login values(?,?);";

        PreparedStatement pstmt=con.prepareStatement(query);

        pstmt.setString(1,username);

        pstmt.setString(2,password);


        pstmt.executeUpdate();

    }

%>

    <hr> Values Inserted Successfully...

    <BR> Welcome <%= username %>

<%

    con.close();

    }

    catch (Exception e)

    {

        System.out.println("DB : "+e.getMessage());

    }

}

```

%></body>

</html>