**JSP (Java Server Pages)**

* JSP is an API of J2EE which accept the user request & generate the dynamic response.
* **With respect to functionality both JSP & Servlets are one & the same because JSP’s get translated to Servlet at runtime**.
* JSP separates business logic from presentation logic. With JSP it is more convenient to write regular HTML than to have a lot of statements that generate the HTML. Also, With this separation different people can work on different tasks;

1. Web page design experts can build the presentation logic

2. web developer can write business logic for generating dynamic data

* Whenever the first request comes to JSP, the container translates the JSP into a servlet (i.e. Java File), compiles the servlet & generates the .class file. This class file is used for generating the response.
* Hence time taken for the getting the response for the first request as compared to subsequent request is more
* We can find the generated servlet & it’s class file in the below path

**<Tomcat Installation Directory>**\work\Catalina\localhost\**<Application Name>**\org\apache\jsp

**<Eclipse\_Workspace>**\.metadata (search for \*.java file inside this directory)

* There are two ways to access JSP’s

1. By typing the JSP file name directly in the URL
2. By Configuring a URL for a JSP in web.xml

Second approach helps us to achieve security & maintainability along with passing initialization information to JSP

Example:-

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

<html>

<body>

Current Date with Time is : <%= **new** Date() %>

<br>

Actress Name is (Config Param) : <%= config.getInitParameter("actress") %>

<br>

Actor Name is (Query Param) : <%= request.getParameter("actor") %>

<br>

Email is (Context Param) : <%= application.getInitParameter("email") %>

</body>

</html>

Web.xml Changes:-

<context-param>

<param-name>email</param-name>

<param-value>abc@xyz.com</param-value>

</context-param>

<servlet>

<servlet-name>myjsp</servlet-name>

<jsp-file>/CurrentDateTime.jsp</jsp-file>

<init-param>

<param-name> actress</param-name>

<param-value>Priyanka</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>myjsp</servlet-name>

<url-pattern>/myJsp</url-pattern>

<!-- <url-pattern>/CurrentDateTime.jsp</url-pattern> -->

</servlet-mapping>

URL’s:-

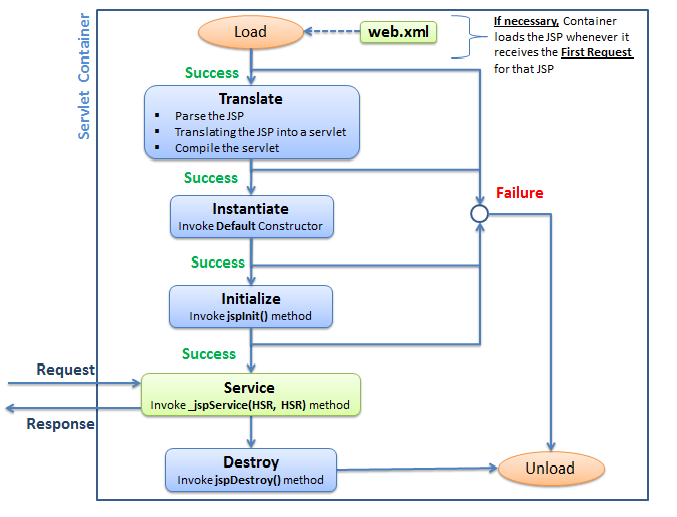
http://localhost:8080/studentsApp/CurrentDateTime.jsp?actor=Rajani

http://localhost:8080/studentsApp/myjsp?actor=Rajani

**Differences between JSP & Servlets:**

|  |  |  |
| --- | --- | --- |
| **No.** | **JSP** | **Servlets** |
| 1 | **By default JSP is protocol dependent i.e. it handles ONLY HTTP & HTTPS Protocol** | **Servlets are Protocol Independent** |
| 2 | **Container invokes \_jspService(HSR, HSR) method irrespective of the HTTP Method present in the request** | **Container invokes corresponding doXXX(HSR, HSR) method depending on the HTTP Method present in the request** |
| 3 | In case of JSP, it get converted into Servlet, it get compiled by servlet container & the compiled .class file is used to generate Dynamic Content  Time taken to generate the response is more for the first request as compared to subsequent request | Already compiled .class files are used to generate the response |
| 4 | Translated Servlet extends HttpJspBase which in turn extends HttpServlet | Servlets extends either GenericServlet or HttpServlet |
| 5 | Implicit Object are available | We have to explicitly define implicit objects in servlets |
| 6 | We can access the JSP either by using the file name directly in the URL or configuring a URL in web.xml | Servlets are accessed ONLY by configuring a URL in web.xml |
| 7 | Business logic is kept separate from presentation logic (Java inside the HTML) | Business logic is tightly coupled with presentation logic (HTML inside Java) |

**JSP Lifecycle:**



The lifecycle of a JSP is controlled by the Servlet Container. Lifecycle of a JSP consists of following phases,

1. Translation & Instantiation Phase
2. Initialization Phase
3. Service Phase
4. Destruction Phase
5. **JSP Translation & Instantiation Phase**:

* Whenever the first request comes to JSP, the container Parses the JSP (Syntax Check), Translates the JSP into a servlet (i.e. Java File) & Compiles the servlet
* Once translation & compilation is successful then container creates the instance of the translated servlet by invoking public default constructor of translated servlet

1. **JSP Initialization Phase :**

Syntax:

**public** **void** jspInit()

{

}

* + - After instanciation, container invokes ini(SC) method which in turn invoke jspInit() me
    - jspInit () method gives us a chance to initialize the translated servlet before handling any client requests. Like, Reading the data from property file etc.
* jspInit() method is called **only once** in JSP life cycle
* We may or may not override this method. If we don’t override, then **default implementation from “HttpJspBase” is invoked**

1. **JSP Service Phase:**

Syntax:

**public** **void** \_jspService(HttpServletRequest request,

HttpServletResponse response)

**throws** ServletException, IOException;

* Whenever client request comes, the container starts a new thread & it instructs the thread to run service(SR, SR) method which in turn calls the \_jspService() method of JSP
* This method is called **one / more times** in the translated servlet lifecycle.
* This method is also responsible for generating responses to **all seven of the HTTP methods**
* **We cannot override this method.** Whatever we write in the JSP becomes the part of this method at the time of translation
* If we do not write anything in the JSP (i.e. Blank JSP File) we won’t get any compile time / runtime exceptions. Instead we get the blank page as the response.

1. **JSP Destruction Phase:**

Syntax:

**public** **void** jspDestroy()

{

}

* When container wants to unload the translated servlet instance from service, it calls the jspDestroy() method. The decision of when to destroy a servlet instance rests on the shoulders of container. Servlet developers should not be concerned with these details, but instead focus on what should be done when the time comes.
* This method is called **ONLY Once** in the lifecycle of a JSP & this method is used to carry out any caretaking tasks (release any DB connection or close a file which is opened by init() method).
* **We may possibly override this method.** If you don’t override destroy(), then **default implementation from “HttpJspBase” is invoked**

**JSP Life Cycle Example:-**

<HTML>

<HEAD>

<TITLE>JSP Life Cycle</TITLE>

</HEAD>

<BODY bgcolor=*"grey"*>

<H1>JSP Life Cycle Example</H1>

<%!

**int** number;

**public** **void** jspInit() {

System.out.println("I am inside the Init method");

number = 50;

}

**public** **void** jspDestroy() {

System.out.println("I am inside the Destroy method");

number = 0;

}

%>

<% System.out.println("I am inside the Service method"); %>

The number is : <%= number %>

</BODY>

</HTML>

**JSP Comment:**

* There is **only one type** of JSP comment available in JSP API.
* Container ignores whatever we write inside this tag at the time of translation.

Syntax: <%-- My JSP Comment --%>

Note: <!-- My HTML Comment --> is not a JSP comment. This is HTML comment. Therefore, there is nothing called as hidden comment or output comment as per JSP specification.

// or /\* comment \*/ used inside the <% %> scriplet tag is also not a JSP comment. This is just a java comment and JSP container doesn’t reserve any special treatment for this comments usage.

**JSP Objects**

* Objects in JSP are grouped into two groups & they are

1. Implicit Objects
2. Explicit Objects

* Every object created in a JSP page should have a scope. There are 4 types of Object Scopes in JSP

1. Page Scope (It’s default) :

The JSP object with ‘Page’ scope can be accessed only within the same JSP (like private variables in java class)

1. Request Scope:

The JSP object with ‘Request’ scope can be accessed from any page that serves the request (more than one page can serve a single request). Functionality is similar to Request Attributes

1. Session Scope:

The JSP object with ‘Session’ scope can be accessed in different pages with in the same session. Functionality is similar to Session Attributes

1. Application Scope:

The JSP object with ‘Application’ scope can be accessed from any page across the application, till the application goes down. Functionality is similar to Context Attributes

**Implicit Objects**

* Implicit objects are the objects that are created by the container
* **These objects use standard variable names. Hence we should use same varia**
* **The implicit objects are available only within the \_jspService() method**.

Our own methods (user defined methods) can't access them as they are local to the service method. But we can pass them to our own method if we wish to use them locally in those functions. Thus for example the following code will cause a compile time error.

<%!

**public** **void** myMethod()

{

out.print("Hello");

}

%>

* There are **9 implicit objects**. Here is the list of all the implicit objects **CODE: SOP2 R2ACE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Variable** | **of Type** | **Scope** | **Description** |
| 1 | out | javax.servlet.jsp.JspWriter | Page | Out object is used to write dynamic data to the output stream. |
| 2 | request | iavax.servlet.http.  HttpServletRequest | Request | Operation is similar to Servlet’s HttpServletRequest Object |
| 3 | response | iavax.servlet.http.  HttpServletResponse | Page | Operation is similar to Servlet’s HttpServletResponse Object |
| 4 | config | iavax.servlet.ServletConfig | Page | Operation is similar to Servlet’s ServletConfig Object |
| 5 | application | iavax.servlet.ServletContext | Application | Operation is similar to Servlet’s ServletContext Object |
| 6 | session | iavax.servlet.http.  HttpSession | Session | Operation is similar to Servlet’s HttpSession Object |
| 7 | page | java.lang.Object | Page | This is as same as “this” |
| 8 | pageContext | javax.servlet.jsp.PageContext | Page | Pagecontext acts like a single API to manage all the other implicit objects.  out = pageContext.getOut();  request = pageContext.getRequest()  response= pageContext.getResponse()  config = pageContext.getServletConfig();  application = pageContext.getServletContext();  session = pageContext.getSession();  exception = pageContext.getException()  page = pageContext.getPage()   * This API is extensively used if we are implementing JSP custom tag handlers. * A typical use of the pageContext is to include another resource or forward the request to another resource. Thus the following would forward from the current page to menu.jsp   <% pageContext.forward("menu.jsp"); %> |
| 9 | exception | java.lang.Throwable | Page | This object is **only available to pages that have isErrorPage set to true** with the directive  <%@ page isErrorPage='true' %> |

**Example for Implicit Objects:**

<html>

<body>

<%-- Implicit Object : request --%>

Request Param : <%= request.getParameter("name") %>

<%-- Implicit Object : config --%>

Config Param : <%= config.getInitParameter("name") %>

<%-- Implicit Object : application --%>

Context Param : <%= application.getInitParameter("email") %>

<%-- Implicit Object : page --%>

Servlet Info : <%= ( (Servlet)page ).getServletInfo() %>

<%-- Implicit Object : response, out & pageContext --%>

<%

out.println("Creating the Cookie ...");

Cookie cookie = **new** Cookie("name1", "Praveen");

response.addCookie(cookie);

/\* HttpServletResponse res = (HttpServletResponse)pageContext.getResponse();

res.addCookie(cookie); \*/

JspWriter jspOut = **null**;

jspOut = pageContext.getOut();

jspOut.println("First Cookie Created");

%>

</body>

</html>

**JSP Tags:-**

* **We know that JSP is like a Java inside HTML & JSP tags helps us to write Java code inside JSP**
* Every JSP tag has its own functionality defined & each tag has its own specific purpose. **We cannot make use of one tag inside an another tag**
* Tags are in JSP are grouped in to 5 groups. They are **CODE : DEADS**

1. Declaration Tag
2. Expression Tag
3. Scriptlet Tag
4. Action Tag
5. Directive Tag

**Declaration Tag [ <%! %> ]:**

* This Tag helps us **to declare variables, methods (non abstract; if used fails at Compilation Phase), inner classes or blocks (static or non-static)**
* Code placed within this will be present outside of \_jspService() method of translated servlet.
* Hence if we declare a variable using this tag then it becomes class level variable
* Declaration tag can be present anywhere inside JSP also we can have any numbers of declaration tags

**Expression Tag [ <% = %> ]:**

* This Tag allows us to print dynamic value to output stream. The dynamic data automatically get converted to String when it is stuffed into output stream
* Code placed with in this tag **should not** end with Semicolon

**Scriptlet Tag [ <% %> ]:**

* This Tag allows **to write any amount of valid java code inside JSP**
* Code placed with in this tag will become part of \_jspService() method of translated servlet.
* Hence if we declare a variable using this tag then it becomes \_jspService() method level variable

**Example for** **Declaration, Expression & Scriptlet Tag**:

<%!

**public** **int** age = 99;

%>

<html>

<body>

<%!

**public** String getName()

{

**return** "Praveen";

}

**public** **int** getAge()

{

**return** age;

}

**public** String getName(String str)

{

**return** str;

}

%>

Name : <%= getName() %><BR>

Age : <%= getAge() %>

<BR><BR><BR>

<%

out.println("Another Name : "+getName("Amir"));

out.println("<BR>");

out.println("Age : "+age);

%>

</body>

</html>

Revisit the JSP Lifecycle Example

**Action Tag [ <jsp:action\_name action\_*attributes* /> ]:**

Action tags in JSP are used to perform action on particular page. We can perform following actions on JSP

1. Include Action

2. Forward Action

3. UseBean Action

**Forward Action [ <jsp:forward page=”relativeURL” /> ] :**

The forward action tag forwards the request from JSP to another resource (static / dynamic)

Example:

<%-- Static Forward (HTML File) --%>

<jsp:forward page=*"index.html"* />

<%-- Static Forward (Text File) --%>

<jsp:forward page=*"MyText.txt"* />

<%-- Dynamic Forward (JSP) --%>

<jsp:forward page=*"Second.jsp?name=praveen"* >

<jsp:param name=*"age"* value=*"999"*/>

</jsp:forward>

<%-- Dynamic Forward (Servlet) --%>

<jsp:forward page=*"/myExample?name=praveen"* >

<jsp:param name=*"age"* value=*"999"*/>

</jsp:forward>

**Include Action [ <jsp:include page=”relativeURL” /> ] :**

This tag is used to include response of another resource (static / dynamic) into the JSP

Example:

<%-- Static Include (HTML File) --%>

<jsp:include page=*"index.html"* />

<%-- Static Include (Text File) --%>

<jsp:include page=*"MyText.txt"* />

<%-- Dynamic Include (JSP) --%>

<jsp:include page=*"Second.jsp?name=praveen"* >

<jsp:param name=*"age"* value=*"999"*/>

</jsp:include>

<%-- Dynamic Include (Servlet) --%>

<jsp:include page=*"/myExample?name=praveen"* >

<jsp:param name=*"age"* value=*"999"*/>

</jsp:include> **DONOT close the Output Stream in the Above Servlet like out.close**

Show the translated servlet to students. Highlight the point “**In all the above cases if the included resource content changes then it will have immediate effect in the response**”.

**Use Bean Action:**

<jsp:useBean id=*"referenceName"*

class=*"package\_nm.class\_nm"* scope=*"page|request|session|application"* />

* This action tag helps us to create explicit objects in JSP
* Container first searches for an existing object by using the id and scope attributes present in this tag.

If an object is found then it makes use of that object.

If an object is not found, it then tries to create the new object.

Example:

**public** **class** Students

{

**private** **int** regNo;

**public** Students(){ }

**public** **int** getRegNo() {

**return** regNo;

}

**public** **void** setRegNo(**int** regNo) {

**this**.regNo = regNo;

}

}

First JSP:

<html>

<body>

<jsp:useBean id=*"students"*

class=*"com.jspiders.studentsapp.beans.Students"*

scope=*"session"* />

Reg. No : <jsp:getProperty name=*"students"* property=*"regno"* />

Changing the Reg. No.

<jsp:setProperty name=*"students"* property=*"regno"* value=*"999"* />

New Reg. No : <jsp:getProperty name=*"students"* property=*"regno"* />

</body>

</html>

Second JSP:

<html>

<body>

<jsp:useBean id=*"students"*

class=*"com.jspiders.studentsapp.beans.Students"*

scope=*"session"* />

Reg. No : <jsp:getProperty name=*"students"* property=*"regno"* />

</body>

</html>

**Directive Tag [ <%@ directive\_name directive\_attributes %> ]:**

There are three types of directive tag

1. Include Directive
2. Page Directive
3. Tag Lib Directive

**Include Directive <%@ include file=”resource\_file\_name” %>:**

* Include directive is used to include the contents of physical file or static resource into JSP at translation time.
* Hence compared to Include Action (in case of Static Resource), Include directive is faster in nature.

Example:

<html>

<body>

<%@include file=*"MyJSP.jsp"* %>

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

<%@include file=*"MyText.txt"* %>

<!-- We Cannot include the JSP along with Query Parameters -->

<%-- <%@include file="MyJSP.jsp?name=praveen" %> --%>

<!-- We Cannot include the Servlet Contents using Include Directive -->

<%-- <%@include file="currentDateTime" %> --%>

</body>

</html>

**Difference between Include Directive & Include Action Tag:**

|  |  |  |
| --- | --- | --- |
| **No** | **Include Directive** | **Include Action Tag** |
| 1 | Include directive is used to include the contents of physical file into JSP | This tag is used to include response of another resource (static / dynamic) into the JSP |
| 2 | In Include Directive we specify the physical file name of the resource | In Include Action we specify the relative URL of the resource |
| 3 | Content of the resource get included into JSP ONLY at JSP translation phase | Content of the resource get included into JSP for every request |
| 4 | Faster in operation | Slower in operation |
| 5 | We cannot use <jsp:param> & Query Parameters with include directive | We can use <jsp:param>& Query Parameters with include action |

**Taglib Directive:**

<%@ taglib uri="*URI of the Tag Library*" prefix="*prefix of the Tag Library*" %>

* The Tag Lib directive helps us to make use of "Custom Tags" in JSP
* The functionality of these custom tags are defined in TLD (Tag Library Descriptor) file

**Page Directive <%@ page attributes %>:**

Page directive has 11 optional attributes that provide the container with special processing information.

**CODE: L I4C BASE2**

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Attribute** | **Syntax** | **Description** |
| 1 | language | <%@ page language = "java" %> | Language denotes the scripting language used in scriptlets, declarations, and expressions in the JSP  Default value is “java” |
| 2 | info | <%@page info="This is the example of info attribute of the page directive." %> | This attribute is used to provide documentation information for a JSP.  Details such as author, version, copyright and date are placed in this attribute.  This information can be later retrieved by using Servlet.getServletInfo() method. |
| 3 | **contentType** | <%@page contentType="text/html" %>  OR  <%@page contentType="text/html;charset=ISO-8859-1" %> | 1. This attribute specifies the Content type and the character encoding for JSP response.  2. The default MIME type is "text/html" and the default character set is "ISO-8859-1". |
| 4 | **import** | <%@ page import = "package1.\*, package2,\*,..." %> | This attribute functionality is similar to java import |
| 5 | extends | <%@ page extends = "package\_name.class\_name" %> | This attribute functionality is similar to java extends. **This attribute is used if you declare any inner class which extends some other class inside a JSP** |
| 6 | Buffer | <%@ page buffer = "none|sizekb" %> | 1. This attribute sets the buffer size in KB to handle output generated by the JSP page which is sent to the client in packets of data.  2. The default size of the buffer is 8KB  3. To turn off buffered output the value should be “none” |
| 7 | autoFlush | <%@ page autoFlush = "true|false" %> | 1. This attribute control the behaviour of output buffer.  2. If true (it’s default), the buffer will be flushed automatically when it is full.  3. If false, then an exception is thrown and results in overflow. |
| 8 | **session** | <%@ page session="true|false" %> | 1. If true (it’s default), then the session implicit  object will be available in JSP & it refers to the current or a new session.  2. If false then session implicit object is unavailable |
| 9 | **isThreadSafe** | <%@ page isThreadSafe ="true|false" %> | 1. If true (it’s default), JSP becomes Multi threaded in nature.  2. If false, JSP becomes Single Threaded. |
| 10 | **isErrorPage** | < %@ page isErrorPage="true|false" %> | 1. If true, then implicit object “exception” is available in the JSP  2. If false (it’s default) then implicit object “exception” is unavailable in the JSP |
| 11 | **errorPage** | <%@ page errorPage = "relativeURL" %> | This attribute is used to define the error page.  If an exception occurs in the current page, then request automatically get forwarded to the specified error page |
| 12 | pageEncoding | NA | NA |
| 13 | isELIgnored | NA | NA |

**NOTE:**

* We can have zero/more page directives in JSP & they can be present anywhere inside JSP
* We can club the multiple page attribute into one page directive as shown below

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

Example:

<%@ page language=*"java"* %>

<%@ page info=*"My First JSP"* %>

<%@ page contentType=*"text/html; charset=ISO-8859-1"* %>

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

<%@ page extends=*"org.apache.jasper.runtime.HttpJspBase"*%>

<%@ page buffer=*"16kb"* %>

<%@ page autoFlush=*"true"* %>

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

<%@ page isThreadSafe=*"true"* %>

<%@ page isErrorPage=*"false"* %>

<%@ page errorPage=*"ErrorJsp.jsp"* %>

<html>

<body>

<h1>Page Directive Example </h1>

<br>

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

<BR>

About this JSP :

<%= ((javax.servlet.jsp.HttpJspPage)page).getServletInfo() %>

</body>

</html>

**Exception Handling in JSP:**

We can handle exception scenerios in JSP by using "errorPage" & "isErrorPage" attributes of page directive

MyJsp.jsp Code:

<%@page errorPage=*"ErrorJsp.jsp"* %>

<html>

<body>

<% **int** i = 100/0; %>

</body>

</html>

Error.jsp Code:

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

<html>

<body>

<font color=*"red"*> Error is : <%=exception.getMessage() %> </font>

</body>

</html>

**Creating Single Threaded JSP :**

We can create JSP which is Single threaded in nature by using "isThreadSafe" attribute of page directive.

JSP Code:

<%@ page isThreadSafe=*"false"* %>

<html>

<body>

Single Thread Model Example

</body>

</html>

Translated Servlet Code:

**public** **final** **class** MyJSP\_jsp

**extends** org.apache.jasper.runtime.HttpJspBase

**implements** org.apache.jasper.runtime.JspSourceDependent, **SingleThreadModel**

{

//Translated Servlet code goes here

}

**Redirect in JSP:**

<html>

<body>

<%

String redirectURL = "http://www.google.com";

response.sendRedirect(redirectURL);

%>

</body>

</html>

**Protocol Independent JSP:**

**Include Using PageContext:**

My JSP

<%-- Static Include (HTML File) --%>

<% pageContext.include("index.html"); %>

<%-- Static Include (Text File) --%>

<% pageContext.include("MyText.txt"); %>

<%-- Dynamic Include (JSP) --%>

<% pageContext.include("Second.jsp?name=praveen"); %>

<%-- Dynamic Include (Servlet) --%>

<% pageContext.include("/myExample?name=Amir"); %>

END of JSP

**Forward Using PageContext:**

<%-- Dynamic Forward (JSP) --%>

<% pageContext.forward("/Second.jsp?name=praveen"); %>

<%-- Dynamic Forward (Servlet) --%>

<% pageContext.forward("/myExample?name=praveen"); %>

<%-- Static Forward (HTML File & Text File) --%>

<!-- NOT Allowed : You Cannot do Static Forwards using pageContext -->

<% pageContext.forward("/index.html"); %>

<% pageContext.forward("/MyText.txt"); %>