**JSP**

1. Get the Req & Generates Dynamic Response
2. Why JSP?

It divides the presentation logic and Business logic.

1. Accessed the JSP using the File name.
2. Time taken for first request is more as compare to following request.

**Java Server Pages (JSP)**

* Its an API of J2EE which accepts the Request & generates “Dynamic Response”
* With Respect to functionality, both Servlets & JSP’s are one & the same because JSP get “translated to a Servlet” at runtime
* JSP Separates “Business Logic” from “Presentation Logic” however these two are tightly coupled in case of Servlets
* With this separation different people can work on different tasks

1. “Web Designer” can develop “presentation Logic”
2. “Web Developer” can develop “business Logic”

**Business Logic:**

**Logic/JavaCode to generate the Dynamic Data using JDBC/ Hibernate /Design Patterns, etc.,**

**Presentation Logic:**

It present the Dynamic Data in Browser using HTML, CSS, JavaScript, Jquery, AngularJS, etc.

* Whenever First Request comes to JSP,

1. Translate the JSP into Servlet (i.e. “.java” file)
2. Compiles the Servlet (i.e “.class” file)
3. This “.class” file is used to Generate the Response

All these things are done by Container & it happens behind the scene at runtime

* Hence time taken to generate the response for First Request is more compared to Subsequent Requests
* JSP File will have “.jsp” file extension & hence it should be created under “WebContent” folder of Dynamic Web Project.
* There are two ways to access JSP

1. BY using the JSP File Name directly in Web URL
2. By configuring URL for a JSP in “Web.xml”

& using this configured URL in Web URL

* Second Approach helps us to achieve “Security” & Maintainability
* “javax.servlet.jsp.\*” is the package representation of JSP API
* By default JSP handles ONLY Http/Https Protocols i.e. by default JSP API is “Protocol dependent” Where as Servlet API is “Protocol Independent”

**MyFirstPage.jsp**

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

**<%**

**//Java Code to Generate Current Date & Time**

**Dae dateRef= new Date();**

**String currDate = dateRef.toString();**

**%>**

**<html>**

**<body>**

**<h2>**

**Vuttrny Fsyr & Yimr id:**

**<font color = “red”> <%= currDate %> </font>**

**</h2>**

**</body>**

**</html>**

**Configuring URL for JSP in web.xml:**

**<servlet>**

**<servlet-name>MyFirstJsp</servlet-name>**

**<jsp-file>/MyFirstJsp.jsp</jsp-file>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>MyFirstJsp</servlet-name>**

**<url-pattern>/currentDateTimeJsp</url-pattern>**

**</servlet-mapping>**

**URLs to Access JSP:**

1. **Using JSP File Name**

[**http://localhost:8080/studentsApp/MyFirstJSP.jsp**](http://localhost:8080/studentsApp/MyFirstJSP.jsp)

1. **Using Configured URL**

[**http://localhost:8080/studentsApp/currentDateTimeJsp**](http://localhost:8080/studentsApp/currentDateTimeJsp%20)

**JSP LifeCycle**

**Lifecycle of a jsp is controlled by servlet container and it has following phases.**

1. **Translation & Instantiation Phases**
2. **Initialization Phase.**
3. **Service Phase.**
4. **Destruction Phase.**

Translation & Instantiation Phase:-

* **Whenever the First Request comes to JSP, Container**

1. **Parses the jsp (i.e. Syntax Check)**
2. **Translates JSP into a Servlet (i.e. “.java” file)**
3. **Compiles the Servlet (i.e “.class” file)**

* **Once Translation & Compilation is successful, Container creates an instance of the Translated Servlet by invoking “Public Default Constructor” ONLY.**

**Initialization Phase**

Syntax

Public void \_jspInit(){

//Initialization Code goes here

}

* After successful Instantiation, container invokes “init(ServletConfig)” method which in turn invokes “\_jspInit() and jspInit()” method
* jspInit() method is called only once in jsp lifecycle
* We may/ may not override this method. If we don’t override then default implementation from “HttpJspBase” is invoked.
* After successful initialization, container caches the Translated Servlet Instance.

**Service Phase**

Syntax

Public void \_jspService(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

* Whenever request comes, container invokes “service(SR, SR) “ method which in turn invokes service(HSR,HSR) and it will invoke “\_jspService(HSR,HSR)
* This method is called one/ more times in JSP Lifecycle i.e equal to number of requests
* This method is also responsible for generating response for all HTTP methods.

**Destruction Phase:**

Syntax:

Public void jspDestroy(){

//clean up code goes here

}

* If container wants to remove the cached servlet instance , then it invokes destroy() Method which inturn invokes jspDestroy() Method
* jspDestroy() method is called ONLY once in JSP Lifecycle
* We may or may not override this method . if we don’t override then default implementation from “HttpJspBase” is invoked.

NOTE:

* Servlet get created by

1. Extending GenericSevlet
2. Extending HttpServlet or
3. Got created because of the JSP

* No Matter how Servlet get created, container always invokes below Lifecycle Methods in that Servlet

1. Public default constructor
2. Init(SC)
3. Service(SR,SR)
4. Destroy()

* In other words, from “Container Perspective” lifeCycle methods remains same.
* However from “Developer Perspective “ LifeCycle methods are different depending on what we are creating i.e Servlet/ JSP

MyFirstJSP.jsp with LifeCycle Methods:-

<% @ Page import = “java.util.Date %>

<%! public void jspInit(){

System.out.println(“Inside jspInit()….”);

}

public void jspDestroy(){

System.out.println(“Inside jspDestroyed()…..”);

}

%>

<%

System.out.println(“inside \_jspService()….”);

//java Code to Generate Date & Time

Date dateRef = new Date();

String currentDate = dataRef.toString();

%>

</html>

</body>

Current Date & Time is:

<font color = “green”>

<%= currentDate %>

</font>

</body>

</html>

**Difference Between Servlets & JSP**

|  |  |
| --- | --- |
| **Servlets** | **JSP** |
| Servlet are protocol Independent | By default JSP is protocol dependent  i.e it handles ONLY HTTP & HTTPS protocols. |
| In Case of Servlets , Business Logic is tightly coupled with presentation Logic i.e “HTML inside Java” | In Case of JSP Business Logic is kept separated From Presentation Logic i.e “Java inside the HTML” |
| In case of Servlets, Already compiled “.class” files are used to generate the response | In case of JSP, it get converted into Servlet, it get compiled & the compiled “.class “ file is used to generate the response. |
| Hence time to take generate the response for 1st Request is Less (as compared to JSP) | But in case of JSP , time to take generate the response for 1st Request is More. |
| Servlet are accessed ONLY by using the configured URL present in web.xml | JSP can be accessed   * Either by using the file name directly in the url * Using the configured URL present in web.xml |
| Servlet extends either “GenericServlet “ or “HttpServlet” | In case of JSP, “Translated servlet extends HttpJspBase “ which in turn extends “HttpServlet” |
| In Case of Servlets (Which extends HttpServlet,) Container invokes coresponding doXXX (HSR, HSR) methods depending on the http method present in the request. | Container invokes \_jspService (HSR, HSR) method irrespective of the Http Method present in the request. |
| Servlets has “\*.java” file extention | JSP has “\*. jsp” file extension |
| Implicitly objects are Not available we have to explicitly define them | Implicit objects are Available. |

**Comments in JSP:-**

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

**SyntaX: <%-- My JSP Comment --%>**

**Note:**

**<!--HTML Comment--> is a Html Comment.**

**Hence there is nothing called as “Hidden Comment” OR “Output Comment “ as Per JSP Specification.**

**JSP Objects:**

* We know that , JSP get translated to Java (Servlet) & in java we can create Objects
* Also, JSP’s are handled by 2 parties

1. Developer who creates the Jsp
2. Container who translates JSP into Servlet.

* Hence Developer as well as Container can create Objects in JSP
* Hence Objects in JSP are grouped into 2 Groups

1. Implicit Objects (Created by Container)
2. Explicit Objects (Created by Developer)

* Every Objects in JSP should have a Scope. There are 4 different types of scopes:

1. Page Scope (it’s Default)
2. Request Scope
3. Session Page
4. Application Scope
5. **Page Scope:**

JSP Object with “Page Scope” can be accessed only within the same JSP (like “private” variables in Java)

1. **Request Scope:**

* JSP Object with “Request Scope” can be accessed from ANY jsp which serves that request (More than one JSP can Serve a Single request)
* Once Response is given back this Objects get Garbage Collected

1. **Session Scope:**

* JSP Object with “Session Scope” can be accessed in ANY JSP’s within the same session.
* Once Session is Invalidated, This Object get Garbage Collected.

1. **Application Scope:**

* JSP Object with “Application Scope” can be accessed in ANY JSP’s across the application till the application up & running
* Once Application/Web Server goes down, this object get Garbage Collected.

**Implicit Objects:**

* Implicit Objects are the objects that are “created by the container”
* Since container creates these objects it uses standard variable names
* Hence we should make use of the same variable names while using implicit objects in JSP
* There are 9 Implicit Objects.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Of Type** | **Scope** | **Description** |
| **Request** | **javax.servlet.http.HttpServletRequest** | **request** |  |
| **response** | **javax.servlet.http.HttpServletResponse** | **request** |  |
| **Session** | **javax.servlet.http.HttpSession** | **session** |  |
| **Config** | **javax.servlet.ServletConfig** | **page** |  |
| **Application** | **javax.servlet.ServletContext** | **Application** |  |
| **Out** | **javax.servlet.jsp.JspWriter** | **page** | **Functionality wise this is similar to “java.io.PrintWriter” Object which we use in Servlet to print the data in Browser.** |
| **Page** | **Java.lang.Object** | **page** | * **This Object holds the Objects reference of Translated Servlet Instance.** * **Functionality is as same as “this”.** |
| **pageContext** | **Javax.servlet.jsp.PageContext** | **page** | **PageContext acts like a Single API to manage All the impl obj** |
| **Exception** | **Java.lang.Throwable** | **Page** | **This Implicit Object is only available to the pages where “isErrorPage” page directive is set to “true”**  **<%@ Page isErrorPage=”true”%>** |

**These Objects functionality is as same as corresponding Servlet objects.**

**Note: Apart From “pageContext” rest of the variable names are in “lower Case”.**

**SOP2 R2ACE**

**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. Also, we cannot make use of one tag inside an another tag. (excluding Action Tag)
* JSP comes with set of “Pre-defined” tags & they are grouped into 5 groups.

1. Declaration Tag --- > **<%! %>**
2. Expression Tag -- > **<%= %>**
3. Scriptlet Tag --> **<% %>**
4. Action Tag--> **<jsp:action\_name action\_attributes />**
5. Directive Tag. --> **<%@ directive-nm directive-attribute%>**

Code Words: **Deads**

1. **Declaration Tag [ <%! %> ]**

* Declaration Tag helps us to declare

1. Variable (static / non-static)
2. Block of Code (static / non-static)
3. Methods (static / non-static but non-abstract) &
4. Inner Classes inside a JSP

* We can have “Zero / One/ More” Declaration Tag & Declaration Tag can be present “Anywhere inside a JSP” (Preferred is to keep at the beginning of the JSP>
* Code present inside this tag will be kept as a first lines of translated servlet at the time of Translation.
* Hence code present within this tag will be present outside of “\_jspService()” Method.
* So if we declare a variable using Declaration Tag then it becomes “Class Level Variable”

1. **Expression Tag** [ <%= %> ]

* This tag helps us to print Dynamic Value in the Browser
* Code placed with in this tag should not end with Semicolon.
* Functionality of this tag is similar to printing the dynamic data using out.print() in case of Servlets.

1. **Scriptlet Tag [ <% %> ]**

* In Servlets we write business logic (Java Code) inside service() method
* We also know that, JSP get tanslated to Servlet at runtime & “we can also write business logic (Java Code) inside JSP using Scriptlet Tag”
* Code placed within this tag becomes part of \_jspService() Method at runtime and get executed for every request.
* If we declare a variable using this tag then it becomes “Method Level Variable” and accessible by only within \_jspService() Method.

Example for Declaration, Expression & Scriptlet Tags:

<%!

private int age = 100;

public String name = “RajniKant”;

public int getAge(){

return age;

}

public String getName(){

return name;

}

public String getName(String givenNM){

return “Given name is – “+givenNM;

}

%>

<html>

<body>

<h1>JSP Tags Example</h1>

Name 1: <%= name %>

Name 2: <%= getName() %>

Name 3: <%= get Name() %>

<%

Int j = 100;

for(int I = 0;i<5;i++){

%>

Name: <%= name %>

<%

}

%>

</body>

</html>

1. **Action Tag <jsp:action\_name action\_attributes />**

Action Tag helps us to perform following actions in JSP

1. Forward Action
2. Include Action
3. UseBean Action
4. Forward Action: [ <jsp: forward page=”relativeUrl” /> ]

This tag helps us to Forward the Request to an another Internal Resource (static/ Dynamic Servlet or JSP)

Example:

<%-- Forward: Static Resource --%>

<jsp: forward page=”index.html”/>

<%-- Forward : Dynamic Resource – Servlet --%>

**<jsp:forward page=”currentDate” />**

**<jsp:forward page = “currentDate?fnam=ABC&lname=XYZ”/>**

**<jsp: forward page= “currentDate”>**

**<jsp:param name=”fname” value=”123”/>**

**<jsp:param name=”lname” value=”456”/>**

**</jsp:forward>**

Note:

* We can Forward the request to only one resource at a time
* Only in case of Action Tag we can make use of other JSP Tags.

For Example:

<%

String url = “currentDate”;

%>

<jsp:forward page =”<%=url%>” />

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

This tag helps us to Include the Response of an another Internal Resource (static /dynamic i.e Servlet or JSP)

Example:

1111111111111

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

2222222222222

**<jsp:inlude page=”currentDate?fname=AAA&lname=BBB”/>**

33333333333333

**<jsp:inlcude page=”currentDate”>**

**<jsp:param name = “fname” value=”xxx” />**

**<jsp:param name=”lname” value =”yyy”/>**

**</jsp:include>**

444444444444444

**<jsp:inlcude page=”currentDateJsp”>**

**<jsp:param name=”fname” value=”123” />**

**<jsp:param name=”lname” value=”456” />**

**</jsp: include>**

55555555555555555555

1. **UseBean** Action

**Syntax:**

<jsp:useBean

Id=”referenceNM”

class =”pkgNM.classNM”

scope=”page|request|session|application”/>

* This tag helps us to create Explicit Objects in JSP
* Container first search for an Existing object by using “id” & “scope” attribute present in this tag
* If found, then it makes the use of that object
* Otherwise,

\*it creates a “New Object” using “Public Default Constructor” &

\* set that object to the corresponding scope

* Since web application is Multi Threaded in Nature, the above logic is “synchronized” in Translated Servlet.
* We know that , Java Bean helps us to transfer the data in the form of Object from one Java program to other.
* Hence using this tag “we can Only create an instance of JavaBean “ & for normal Java Classes we should use “Scriptlet Tag”

Example:

<jsp:useBean

Id=”ref”

Class =”com.jspiders.studentsapp.dao.StudentInfoBean”

Scope=”application”

/>

Reg. no. Value is:

<jsp:getProperty name=”ref” property=”regno” />

Changing the Value of Reg. no.

<jsp:setProperty name=”ref” property=”regno” value=”10” />

New reg no value is:

<jsp:getProperty name= “ref” property=”regno”/>

5- **Directive Tag [ <%@directive\_name directive\_attribute %> ]**

**There are 3 Types of Directive Tags**

1. **Include directive**
2. **TagLib Directive**
3. **Page Directive**
4. Include Directive- <%@include file=”resourceFileNM”%>

* Include Directive includes the content of Physical File OR Static Resource into JSP at the time of translation
* Hence compared to Include Action (for static Resource Include Directive is Faster in nature

**Note:** Use include Action for Dynamic Resources & Use Include Directive for static Resources

Example:

11111111111111111

<[%@include file = “index.html” %](mailto:%25@include%20file%20=%20)>

2222222222222222222

<%--

Not Possible !!! We Cannot include Dynamic Resource using Include Directive

--%>

<%-- <%@include file=”currentDate” %> --%>

<% -- <%@include file=”currentDateJsp” %> --%>

ii. **Taglib Directive**

**Syntax:**

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

* Taglib Directive helps us to make use of “Custom Tags” in JSP
* The functionality of these custom tags are defined in “Tag Library Descriptor (TLD)” file
* Tag Library Description file will have “.tld” file extension & it will present inside WEB-INF folder.

**Page Directive [ <%@page attributes%> ]**

* Page Directive has 11 Optional Attribute that provides some processing information to Servlet Container.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Syntax** | **Description** |
| Info | <%@page info=”My First JSP” %> | * This attribute is used to provide documentation information for a JSP * Details Such as Author, Version, Copyright, Date, etc., are placed inside this tag. |
| Language | <%@page language=”java” %> | * Language denotes the “Scripting Language” used in Declaration, Expression & Scriptlet Tags. * Default language is “Java” |
| **contentType** | <%@page contentType=”text/html” %>  Or  <%@ page contentType=”text/html; charset=ISO-8859-1”%> | * This attribute specifies the Content Type and Character Set for JSP Response. * Default Content Type is “text/html” &Default Character Set is”ISO-8859-1” |
| Import | <%@page import=”pkg1.class1, pkg2.\*,.. “%>  Or  <[%@page import=”pkg1.class1”%](mailto:%25@page%20import=)>  <%@page import=”pkg2.\*”%> | * Functionality is similar to Java “import” statement. |
| Extends | <%@page extends=”pkgNM.classNM”%> | * Functionality is similler to Java “extends” statement |
| Buffer | <%@page buffer=”none | sizeInKB” %> | * This attribute set the Buffer Size in KB * The default size is 8KB * To turn off the buffered output the value should be “none”. |
| autoFlush | <%@page autoFlush=”true | false”%> | * This attribute controls the behavior of Buffer * If “true” (it’s default), the buffer will be flushes automatically when it’s full * If “false” then an exception is thrown when the buffer is full. |
| Session | <%@page session=”true” | false %> | * If “true” (it’s default), then “session” implicit object is available in JSP % it refers to Current/ New Session * If “false”, then “session” implicit object will NOT be available in JSP |
| isThreadSafe | <%@page isThreadSafe =”true” | false”%> | * If “true” (it’s default), then JSP becomes “Multi Threaded” in nature * If “false”, then JSP becomes “Single Threaded” in nature. |
| isErrorPage | <%@page isErrorPage=”true” |”false” %> | * If “true”, then “exception” implicit object is available in JSP * If False (it’s default), then “excepiton” implicit object will Not be availble in JSP |
| errorPage | <%@page errorPage=”relativeURL”%> | * This attribute defines the “Error Page” for a JSP * While processing the request for a JSP , if an exception occurs then request automatically get forwarded to the “Specified Error Page” * The value of “errorPage” attribute is URL of the Static/ Dynamic Resource. |

* Info
* Language
* Content type
* Buffer
* Autoflush
* isThreadSafe
* isErrorPage
* ErrorPage
* Import
* Extends
* session

Code words: LI4C BASE2

NOTE:

1. We can have Zero / More Page Directive in JSP & they can be present anywhere inside a JSP (Preferred is to keep at the beginning of the JSP)
2. We can club multiple Page Directive Attributes into one page directive as shown below

<%@ page errorPage=”JSPErrorPage.jsp” import=”java.util.Date” %>

Which is equal to

<%@page errorPage=”JSPErrorPage.jsp”%>

<%@page import=”java.util.Date” %>

1. Declaration Tag & Page Directive Tag Can be present at the beginning of the JSP (preferred ). So Between these two, page directive tag should be present first followed by Declaration Tag.

Exception Handling in JSP

In JSP we can handle exception scenarios with the help of “errorPage” & “isErrorPage” page directive attributes.

Example:

MyJSP.jsp

<%@page errorPage=”JSPErrorPage.jsp”%>

<html>

<body>

Exception Simulation…

<%

Int i = 100/0;

%>

</body>

</html>

JSPErrorPage.jsp

<%@page isErrorPage=”true”%>

<html>

<body>

<h1>Exception Occurred!!! Exception is: </h1>

<%= exception.getMessage() %>

</body>

</html>

Single Thread JSP:

We can create JSP, single threaded in nature with the help of “isThreadSafe” page Directive attribute.

Example:

<%@page isThreadSafe=”false”%>

<html>

<body>

<h1>My Single Threaded JSP</h1>

</body>

</html>

<%-- Don’t Generate the Response, Instead Redirect the Request =%>

<%

String url = “http://www.gmail.com”;

response.sendRedirect(url);

%>