5.JSP(JavaServer Pages)

JavaServer Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags.

Using JSP, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

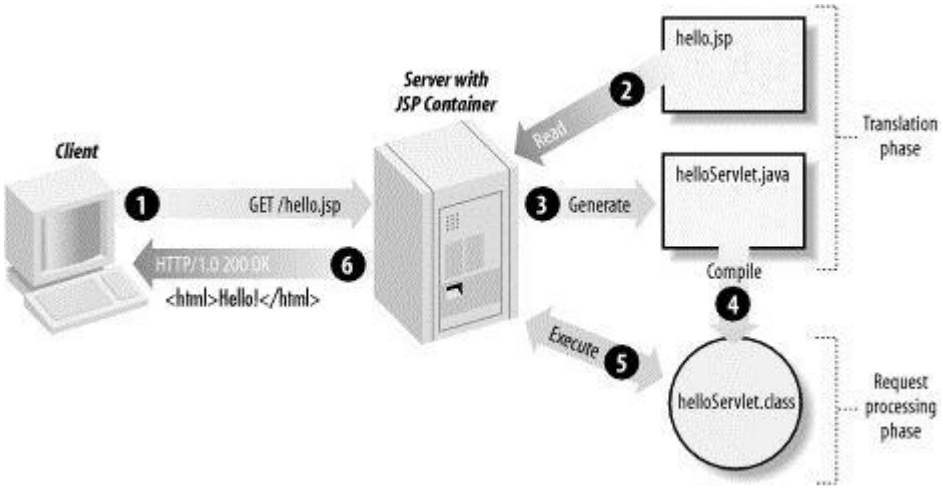
Advantages of JSP:

vs. Pure Servlets: It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

JSP Processing:

The following steps explain how the web server creates the web page using JSP:

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of .html.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.

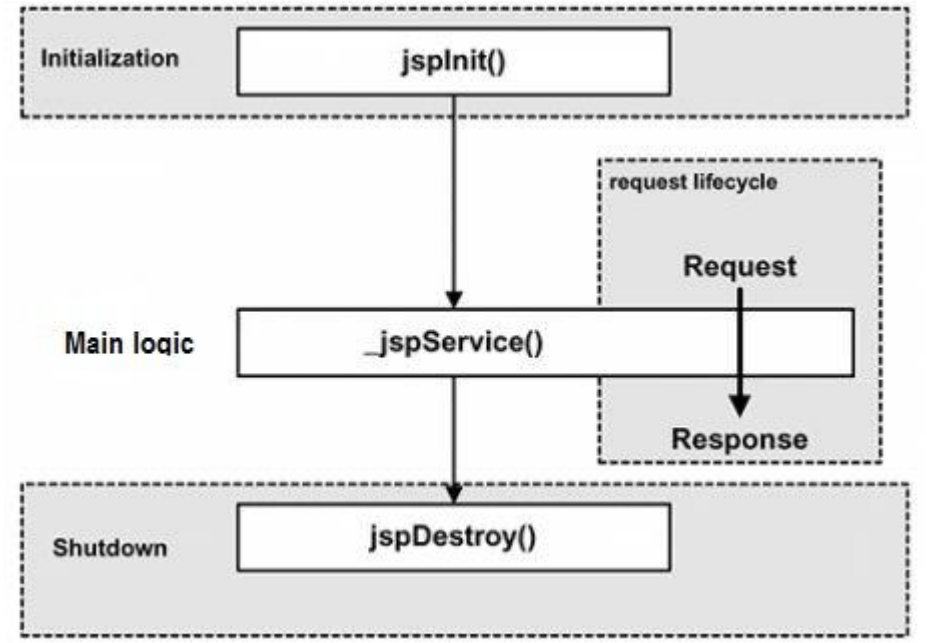


LIFE CYCLE:

A JSP life cycle can be defined as the entire process from its creation till the destruction which is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

The following are the paths followed by a JSP

- Compilation
- Initialization
- Execution
- Cleanup



JSP Compilation:

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps:

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

JSP Initialization:

When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method:

```
public void jspInit(){
  // Initialization code...
}
```

Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution:

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows:

```
void _jspService(HttpServletRequest request,
                 HttpServletResponse response)
{
  // Service handling code...
}
```

The _jspService() method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods ie. GET, POST, DELETE etc.

JSP Cleanup:

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form:

```
public void jspDestroy()
{
  // Your cleanup code goes here.
}
```

JSP Elements :

1. **JSP Comments**
2. **JSP Scriptlets**
3. **JSP Expression**
4. **JSP Directives**
5. **JSP Declaration**

1. JSP Comments

Since JSP is built on top of HTML, we can write comments in JSP file like html comments as

```
<-- This is HTML Comment -->
```

These comments are sent to the client and we can look it with view source option of browsers.

We can put comments in JSP files as:

```
<%-- This is JSP Comment--%>
```

This comment is suitable for developers to provide code level comments because these are not sent in the client response.

2. JSP Scriptlets

Scriptlet tags are the easiest way to put java code in a JSP page. A scriptlet tag starts with `<%` and ends with `%>`.

Any code written inside the scriptlet tags go into the `_jspService()` method.

For example:

```
<%

Date d = new Date();

System.out.println("Current Date="+d);

%>
```

3. JSP Expression

Since most of the times we print dynamic data in JSP page using *out.print()* method, there is a shortcut to do this through JSP Expressions. JSP Expression starts with `<%=` and ends with `%>`.

`<% out.print("Pankaj"); %>` can be written using JSP Expression as `<%= "Pankaj" %>`

Notice that anything between `<%= %>` is sent as parameter to `out.print()` method. Also notice that scriptlets can contain multiple java statements and always ends with semicolon (;) but expression doesn't end with semicolon.

4. JSP Directives

JSP Directives are used to give special instructions to the container while JSP page is getting translated to servlet source code. JSP directives starts with `<%@` and ends with `%>`

For example, in above JSP Example, I am using *page* directive to to instruct container JSP translator to import the Date class.

5. JSP Declaration

JSP Declarations are used to declare member methods and variables of servlet class. JSP Declarations starts with `<%!` and ends with `%>`.

For example we can create an int variable in JSP at class level as `<%!`
`public static int count=0; %>`

JSP Implicit Objects:

JSP supports nine automatically defined variables, which are also called implicit objects. These variables are:

Objects	Description
request	This is the **HttpServletRequest** object associated with the request.
response	This is the **HttpServletResponse** object associated with the response to the client.
out	This is the **PrintWriter** object used to send output to the client.
session	This is the **HttpSession** object associated with the request.
application	This is the **ServletContext** object associated with application context.
config	This is the **ServletConfig** object associated with the page.

pageContext	This encapsulates use of server-specific features like higher performance **JspWriters**.
page	This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
Exception	The **Exception** object allows the exception data to be accessed by designated JSP.

JSP Directives:

A JSP directive affects the overall structure of the servlet class. It usually has the following form:

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

There are three types of directive tag:

Directive	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
<%@ include ... %>	Includes a file during the translation phase.
<%@ taglib ... %>	Declares a tag library, containing custom actions, used in the page

JSP Action Tags:

JSP actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

There is only one syntax for the Action element, as it conforms to the XML standard:

```
<jsp:action_name attribute="value" />
```

Action elements are basically predefined functions and there are following JSP actions available:

Syntax	Purpose
jsp:include	Includes a file at the time the page is requested
jsp:useBean	Finds or instantiates a JavaBean
jsp:setProperty	Sets the property of a JavaBean
jsp:getProperty	Inserts the property of a JavaBean into the output
jsp:forward	Forwards the requester to a new page
jsp:plugin	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin
jsp:element	Defines XML elements dynamically.
jsp:attribute	Defines dynamically defined XML element's attribute.
jsp:body	Defines dynamically defined XML element's body.
jsp:text	Use to write template text in JSP pages and documents.

To give an example for a JSP code, first we are going to print the text "Hello Hiox". Try the the following syntax code.

Example :

```
<html>
<body>
<! -- This is the JSP file-->
<%
out.println ("Hello HIOX");
%>
</body>
</html>
```

Result :

```
Hello HIOX
```

JSTL

JSTL (JSP Standard Tag Library)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantage of JSTL

1. **Fast Developement** JSTL provides many tags that simplifies the JSP.
2. **Code Reusability** We can use the JSTL tags in various pages.
3. **No need to use scriptlet tag** It avoids the use of scriptlet tag.

JSTL Tags

There JSTL mainly provides 5 types of tags:

Tag Name	Description
Core tags	The JSTL core tag provide variable support, URL management, flow control etc. The url for the core tag is **http://java.sun.com/jsp/jstl/core** . The prefix of core tag is **c**.
Function tags	The functions tags provide support for string manipulation and string length. The url for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is **fn**.
Formatting tags	The Formatting tags provide support for message formatting, number and date formatting etc. The url for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**.
XML tags	The xml sql tags provide flow control, transformation etc. The url for the xml tags is **http://java.sun.com/jsp/jstl/xml**and prefix is **x**.
SQL tags	The JSTL sql tags provide SQL support. The url for the sql tags is **http://java.sun.com/jsp/jstl/sql** and prefix is **sql**.

AJAX:

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.

- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.