JSP (Java Server Page)

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

Java Server Pages often serve the same purpose as programs implemented using the **Common Gateway Interface (CGI)**. But JSP offers several advantages in comparison with the CGI.

Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.

JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.

JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including **JDBC, JNDI, EJB, JAXP,** etc.

JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

JSP Features

Coding in JSP is easy :- As it is just adding JAVA code to HTML/XML.

Reduction in the length of Code :- In JSP we use action tags, custom tags etc.

Connection to Database is easier :-It is easier to connect website to database and allows to read or write data easily to the database.

Make Interactive websites :- In this we can create dynamic web pages which helps user to interact in real time environment.

Portable, Powerful, flexible and easy to maintain :- as these are browser and server independent.

No Redeployment and No Re-Compilation :- It is dynamic, secure and platform independent so no need to re-compilation.

Extension to Servlet :- as it has all features of servlets, implicit objects and custom tags

JSP Elements

Comment

Comments are marked as text or statements that are ignored by the JSP container. They are useful when you want to write some useful information or logic to remember in the future.

Syntax: <%-- Set of comment statements --%>

Directive

These tags are used to provide specific instructions to the web container when the page is translated. It has three subcategories:

Syntax <%@ directive %>

Declaration

As the name suggests, it is used to declare methods and variables you will use in your Java code within a JSP file. According to the rules of JSP, any variable must be declared before it can be used.

Syntax <%! declarations %>

JSP Elements

Scriptlet

The scriptlet tag allows writing Java code statements within the JSP page. This tag is responsible for implementing the functionality of **_jspService()** by scripting the java code.

Syntax <% scriplets %>

Expression

Expressions elements are responsible for containing scripting language expression, which gets evaluated and converted to Strings by the JSP engine and is meant to the output stream of the response. Hence, you are not required to write **out.print()** for writing your data. This is mostly used for printing the values of variables or methods in the form of output.

Syntax<%= expression %>

JSP Directives

S.No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements. Syntax: <%@ page [attribute="value" attribute="value" ...] %> Attribute: Import, contentType, extends, info, buffer, language, isELIgnored, isThreadSafe, session, pageEncoding, errorPage, isErrorPage
2	**<%@ include ... %>** Includes a file during the translation phase. Syntax: <%@ include file="filename" %>
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page Syntax: <%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>

JSP Directives

import attribute

The import attribute defines the set of classes and packages that must be imported in servlet class definition.

language attribute

language attribute defines scripting language to be used in the page.

extends attribute

extends attribute defines the class name of the superclass of the servlet class that is generated from the JSP page.

session attribute

session attribute defines whether the JSP page is participating in an HTTP session. The value is either true or false.

isThreadSafe attribute

isThreadSafe attribute declares whether the JSP is thread-safe. The value is either true or false

JSP Directives

isErrorPage attribute

isErrorPage attribute declares whether the current JSP Page represents another JSP's error page.

errorPage attribute

errorPage attribute indicates another JSP page that will handle all the run time exceptions thrown by current JSP page. It specifies the URL path of another page to which a request is to be dispatched to handle run time exceptions thrown by current JSP page.

contentType attribute

contentType attribute defines the MIME type for the JSP response.

autoFlush attribute

autoFlush attribute defines whether the buffered output is flushed automatically. The default value is "true".

buffer attribute

buffer attribute defines how buffering is handled by the implicit out object.

JSP Actions

<jsp:action_name attribute = "value" />

S.No.	Syntax & Purpose
1	**jsp:include** Includes a file at the time the page is requested.
2	**jsp:useBean** Finds or instantiates a JavaBean.
3	**jsp:setProperty** Sets the property of a JavaBean.
4	**jsp:getProperty** Inserts the property of a JavaBean into the output.
5	**jsp:forward** Forwards the requester to a new page.

JSP Actions

<jsp:action_name attribute = "value" />

S.No.	Syntax & Purpose
6	**jsp:plugin** Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
7	**jsp:element** Defines XML elements dynamically.
8	**jsp:attribute** Defines dynamically-defined XML element's attribute.
9	**jsp:body** Defines dynamically-defined XML element's body.
10	**jsp:text** Used to write template text in JSP pages and documents.

JSP Implicit Object

the Implicit Objects in JSP. These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared. JSP Implicit Objects are also called **pre-defined variables**.

Following table lists out the nine Implicit Objects that JSP supports

S.No.	Object & Description
1	**request** This is the **HttpServletRequest** object associated with the request.
2	**response** This is the **HttpServletResponse** object associated with the response to the client.
3	**out** This is the **PrintWriter** object used to send output to the client.
4	**session** This is the **HttpSession** object associated with the request.

JSP Implicit Object

5	**application** This is the **ServletContext** object associated with the application context.
6	**config** This is the **ServletConfig** object associated with the page.
7	**pageContext** This encapsulates use of server-specific features like higher performance **JspWriters**.
8	**page** This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
9	**Exception** The **Exception** object allows the exception data to be accessed by designated JSP.

JSP Implicit Object

The request Object

The request object is an instance of a **javax.servlet.http.HttpServletRequest** object. Each time a client requests a page the JSP engine creates a new object to represent that request.

The response Object

The response object is an instance of a **javax.servlet.http.HttpServletResponse** object. Just as the server creates the request object, it also creates an object to represent the response to the client.

The out Object

The out implicit object is an instance of a **javax.servlet.jsp.JspWriter** object and is used to send content in a response.

JSP Implicit Object

The session Object

The session object is an instance of **javax.servlet.http.HttpSession** and behaves exactly the same way that session objects behave under Java Servlets.

The application Object

The application object is direct wrapper around the **ServletContext** object for the generated Servlet and in reality an instance of a **javax.servlet.ServletContext** object.

The config Object

The config object is an instantiation of **javax.servlet.ServletConfig** and is a direct wrapper around the **ServletConfig** object for the generated servlet.

The pageContext Object

The pageContext object is an instance of a **javax.servlet.jsp.PageContext** object. The pageContext object is used to represent the entire JSP page.

JSP Implicit Object

The page Object

This object is an actual reference to the instance of the page. It can be thought of as an object that represents the entire JSP page.

The page object is really a direct synonym for the **this** object.

The exception Object

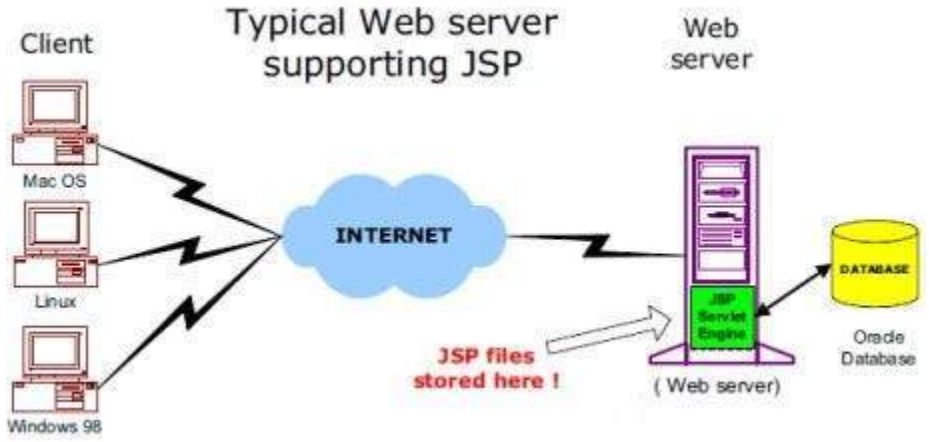
The exception object is a wrapper containing the exception thrown from the previous page. It is typically used to generate an appropriate response to the error condition.

JSP Architecture

The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. IDE makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.



JSP Processing

The following steps explain how the web server creates the Webpage 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. This code implements the corresponding dynamic behavior of the page.

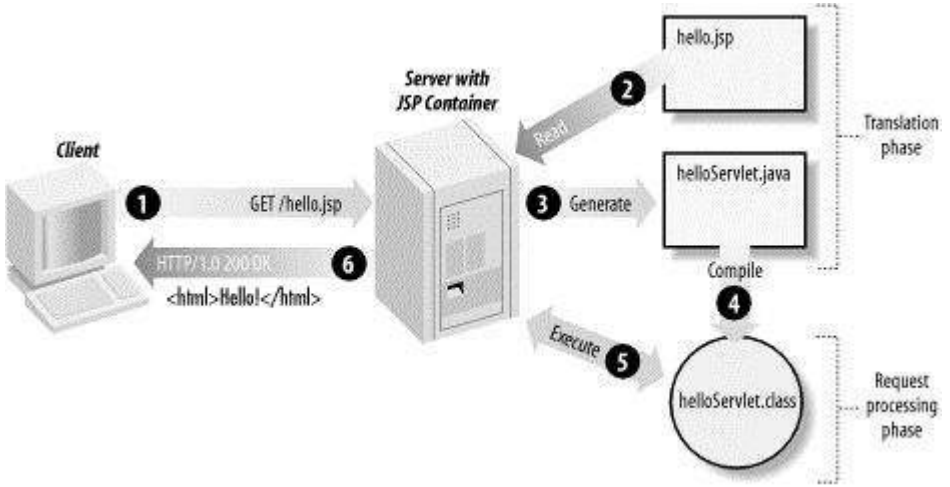
The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.

JSP Processing

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. The output is furthur passed on to the web server by the servlet engine inside an HTTP response.

The web server forwards the HTTP response to your browser in terms of static HTML content.

Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.



JSP Life Cycle

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

Paths Followed By JSP

The following are the paths followed by a JSP −

Compilation

Initialization

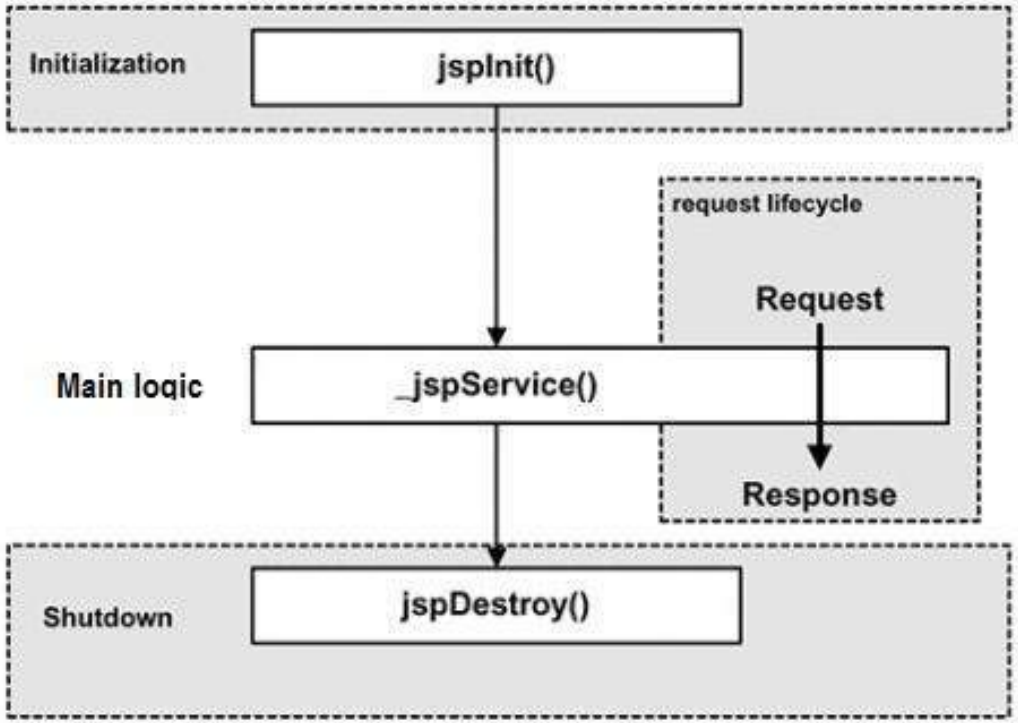
Execution

Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below −

JSP Life Cycle

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below −



JSP Life Cycle

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 Life Cycle

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 Life Cycle

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 on request basis. This 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, i.e, GET, POST, DELETE, etc.

JSP Life Cycle

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.
}
```

Creating Static content

You create static content in a JSP page simply by writing it as if you were creating a page that consisted only of that content.

Static content can be expressed in any text-based format, such as HTML, Wireless Markup Language (WML), and XML. The default format is HTML.

If you want to use a format other than HTML, at the beginning of your JSP page you include a page directive with the contentType attribute set to the content type.

The purpose of the contentType directive is to allow the browser to correctly interpret the resulting content.

So if you wanted a page to contain data expressed in WML, you would include the following directive:

<%@ page contentType="text/vnd.wap.wml"%>

Creating Dynamic content

You create dynamic content by accessing Java programming language object properties.

Using Objects within JSP Pages

You can access a variety of objects, including enterprise beans and JavaBeans components, within a JSP page. JSP technology automatically makes some objects available, and you can also create and access application-specific objects.

Using Implicit Objects

Implicit objects are created by the web container and contain information related to a particular request, page, session, or application. Many of the objects are defined by the Java servlet technology underlying JSP technology. The section Implicit Objects explains how you access implicit objects using the JSP expression language.

Creating Dynamic content

Using Application-Specific Objects

When possible, application behavior should be encapsulated in objects so that page designers can focus on presentation issues. Objects can be created by developers who are proficient in the Java programming language and in accessing databases and other services. The main way to create and use application-specific objects within a JSP page is to use JSP standard tags (discussed in JavaBeans Components) to create JavaBeans components and set their properties, and EL expressions to access their properties.

Using Shared Objects

The conditions affecting concurrent access to shared objects (described in Controlling Concurrent Access to Shared Resources) apply to objects accessed from JSP pages that run as multithreaded servlets. You can use the following page directive to indicate how a web container should dispatch multiple client requests:

<%@ page isThreadSafe="true|false" %>

Creating Dynamic content

When the isThreadSafe attribute is set to true, the web container can choose to dispatch multiple concurrent client requests to the JSP page. This is the default setting. If using true, you must ensure that you properly synchronize access to any shared objects defined at the page level. This includes objects created within declarations, JavaBeans components with page scope, and attributes of the page context object (see Implicit Objects).

If isThreadSafe is set to false, requests are dispatched one at a time in the order they were received, and access to page-level objects does not have to be controlled. However, you still must ensure that access is properly synchronized to attributes of the application or session scope objects and to JavaBeans components with application or session scope. Furthermore, it is not recommended to set isThreadSafe to false. The JSP page's generated servlet will implement the javax.servlet.SingleThreadModel interface, and because the Servlet 2.4 specification deprecates SingleThreadModel, the generated servlet will contain deprecated code.